COMPUTER INTEGRATED MANUFACTURING

MANUFACTURING TECHNOLOGY

Manufacturing technology is the technology of selecting the appropriate material, best configuration, and optimal process control to produce competitive products. It is the technology of integrating machines, operators, and the management of work to control a manufacturing process. The manufacture of industrial electronics devices, components, subsystems, and systems involves hundreds of operations and processes that are amenable to automation and computerization. Competitiveness in manufacturing is a global challenge, and every manufacturing organization is striving to enhance its competitive posture by employing the best and most effective tools and methods to produce products at competitive prices. The competitive environment is characterized by a number of simultaneous developments. Products are more functional, variable, and complex than ever before. Delivery lead times are shorter. The rate of product development has intensified. Products that used to stay in the market for 5 years are now retired after 1 or 2 years. Customers are increasingly concerned about quality, dependability, cost, safety, maintainability, and environmental compliance. These developments are forcing manufacturers to look critically at their design and manufacturing activities, searching for opportunities to make serious improvements over the normal "business as usual." The barrier between design and manufacturing must

be removed to allow for concurrent engineering and multifunctional teamwork. Information flow from customer order to engineering to production preparation to manufacture must be streamlined (1).

Manufacturing technology started about 200 years ago with the English system. There have been six major events in the history of manufacturing. Each manufacturing event has been triggered by the development of new technology that represented a major milestone in the solution of a particular problem. The new technology required changes in the nature of the organization of manufacturing and in the machines used to make these changes (2,3).

The first major manufacturing event was the English system of manufacturing, originated in the late 1700s with the invention of general-purpose machine tools, such as lathes, that could be used to fabricate a variety of work pieces. This development was a breakthrough because it separated the product's function from the process used to make it. As a result, improvements in manufacturing were made independently of the products. Consequently, the manufacturing technology became free from the constraints of the product. The English system emphasized craftsmanship, a one-at-a-time method.

The American system of manufacturing emerged in the mid-1800s, emphasizing mass production and interchangeability of parts. The English system emphasized the best possible fit among components, whereas the American system aimed at the greatest possible tolerance without loss of functionality. The American system is a mass production method of making many parts of one kind and assembling a product from a set of these parts. The manufacture of a rifle is a good example of the American system of manufacturing. Managing variation became the hallmark of the American system.

The third major manufacturing event is called the period of Scientific Management, which began in the late 1800s. Scientific Management was based on the work of Frederick Taylor, an American mechanical engineer. Taylor advocated the idea that workers were limiting the speed and efficiency of machines. Using job analysis and time study, he determined a standard rate of output for each job. The approach of standardization and control of machines narrowed the scope of work, and left nothing to the worker's decision. Taylorism placed the control of manufacturing in the hands of management, which could monitor a worker's productivity by comparing it against a standard.

The fourth period occurred in the mid-twentieth century. It is based on statistical process control (SPC), which was invented in the United States. In SPC it is assumed that machines intrinsically produce definable variations. SPC emphasis is on out-of-control situations rather than on mean performance; therefore, it directs management's attention away from the worker and toward man–machine variations.

The fifth major event in the history of manufacturing is the introduction of numerical control (NC) in the mid-1940s. NC became more mature in the late 1970s with the embedded microprocessor-based controller. NC emphasizes monitoring and controlling machines and encourages experimentation and learning (2,3).

In the late 1980s, manufacturing entered a new era, computer integrated manufacturing (CIM) and flexible manufacturing (FM). CIM is based on the use of information and models of functional expertise that make it possible to examine and optimize the interactions between tasks and functional blocks. Functional blocks or modules are machines, workstations, workcells, or software modules where complex operations, or processes are performed. CIM deals with the effective use of computers and automation in manufacturing to improve production (quality, yield, cycle time) mix of products running concurrently, and reduce the unit product cost (UPC) by reducing the recurring and nonrecurring engineering (NRE) costs. The recurring manufacturing cost of a product consists of the cost of material, labor, and overhead (MLO). NRE cost is the cost of product design and development and the investment in manufacturing, tooling, machinery, and processes. It should be mentioned that automation, in and of itself, is not the answer to low cost, and high-flexibility manufacturing. Automation is a tool, and can be used correctly or incorrectly. FM is the application of CIM to increase the range of products and to decouple UPC from volume (i.e., produce products at low cost and low volume). CIM/FM are appropriate manufacturing technologies for cost-effective manufacturing. Technologies to manufacture affordable devices and systems are of significant interest to industry and government. With CIM, flexibility in the form of new products and processes is achievable. CIM provides the capability of performing sequential and parallel tasks simultaneously, which increases the versatility and productivity of a CIM factory (1). Process equipment that is compatible with CIM contains embedded computers (hardware and software) in the form of micro-controllers, field-programmable-gate-arrays (FPGA), digital signal processors, and modems to communicate with and control other equipment. Process equipment used in conventional factories is not suitable for CIM. From a system engineering point of view, a CIM factory is network-centric, meaning that machines, operators, processes, and operations are under computer control through the factory local-area-network (LAN).

Over the last four decades, drivers in manufacturing for competitive success have been:

1. Cost advantage: characterized by highly productive labor, high volume, and low mix of products.
2. Quality: emphasizing statistical process control, variability reduction, and customer satisfaction.
3. Time-to-market: emphasizing cycle time reduction, integrating product and process development, using production and process simulation, and requiring a highly skilled and adaptable work force.
4. Product variety: characterized by UPC, which is independent of volume, flexibility, and simulation of product and factory.
5. Company goodness: emphasizing environmental impact.

Businesses are using several computer-aided techniques in conjunction with CIM for accelerating the design and production of products. Among those that have proven successful are concurrent engineering, just-in-time (JIT) and just-in-sequence (JIS) inventory control in manufacturing and electronic design automation (EDA) tools (i.e., rapid prototyping and virtual prototyping). Computer simulation of manufacturing processes has proven to be a very useful tool in complementing the practical know-how of manufacturing planners

and designers. In the late 1980s, the Defense Advanced Research Projects Agency (DARPA) and the U.S. Air Force sponsored the Microelectronics Manufacturing Science and Technology (MMST) program at Texas Instruments. The program was a detailed approach to reduce the principles of CIM to practice on a factory floor. The objective of the program was to create a CMOS Integrated Circuits factory with flexibility to process each wafer using different design, with all scheduling and processing fully under the control of a centralized system.

SYSTEM ENGINEERING IN MANUFACTURING

Recently, the design, development, and manufacture of systems has become more complex and has increased the need for systematic techniques, principles, and methodologies. System designers and integrators have been faced simultaneously with increasingly complex technical demands under challenging schedule and budget conditions, acutely limited resources and test data, and fast-changing technologies. Performance requirements invariably include severe reaction and response time constraints that cannot be met without the proper integration and resource allocation of personnel, hardware, software, and procedures. At the same time, system development cost reductions, accelerated schedules, and lack of complete system tests prior to operational usage have combined to reduce the availability of adequate operational data bases, either in quality or quantity, and increase the challenge of system integration. The present business environment involves global competition, and the customer is focused on acquiring cost-effective (affordable) systems. Accordingly, it is necessary to have a comprehensive methodology of system design, integration, management, and analysis using all available information to pinpoint problem areas, produce competitive and affordable products, and provide a numerical estimate of system effectiveness during all phases of the system life cycle (4,5). The production of competitive products requires a system engineering approach to manufacturing.

In today's environment, everyone seems to be working on systems or subsystems of some kind. There are communication systems, political systems, transportation systems, economic systems, radar systems, information systems, manufacturing systems, and production systems, to mention but a few. One might ask the question, Can all these truly be called systems? An attempt to answer this question requires examining the real world, which is made of material things, organisms, and ideational constructs that are variously related to one another, structurally and functionally. Both the functional and structural relationships tend to span all three spatial dimensions and time. The multidimensionality of systems leads to a hierarchy of systems. Hierarchy of systems means there is a vertical and horizontal structure of systems. The vertical structure consists of systems embedded within systems. The horizontal structure is characterized by mutual interdependence among the units at each tier, where the units are operationally dependent on those of the neighboring tiers. Consequently, the term *system* has broad application.

System engineering is a multidisciplinary subject; it uses all the engineering disciplines, operation research, management science, and economics to provide a systematic approach and rationale for building systems. A systematic approach follows a disciplined series of steps to focus on the objectives and the desired attributes of a product prior to building it. One aspect of the system engineering process consists of a logical sequence of activities and decisions that transforms a customer specification into a product performance specification, a design specification, and a preferred product configuration. The design concept aspect is concerned with system design and analysis, system integration, and tradeoffs between factors such as performance, reliability, safety, cost and availability. Another aspect of the system engineering process is concerned with the system engineering management activities, which deal with the application of business disciplines such as planning, budgeting, costing, schedule control, policy direction, acquisition strategy, resources management, and the evaluation of program performance. A comprehensive system engineering process deals with both aspects (5–7).

During the conceptual phase of system development, a methodology must be used to determine the system requirement, tradeoffs, input/output, and operational environment. System engineering is the process dealing with all aspects of system design. The system engineering process consists of several phases: (1) requirements/objective determination, (2) system performance specification, (3) establishment of system architecture and tradeoffs analysis, (4) detailed design and tradeoff of building blocks, (5) integration, costing, testing, and design specification of the building blocks, and (6) development of metrics to evaluate these activities. The development and design of a CIM factory follows a system engineering approach. The CIM factory is the system, and the building blocks of the factory are those of the system.

After the CIM system architecture is established, the next step is synthesizing the required functionality to design the building blocks. The design process starts by simulating the system using the performance specification and generating several designs with different hardware and software configurations. During the design process, tradeoffs are made between performance, cost, safety, reliability, and supportability. The end product of the detailed design process is the design specification of the hardware and software building blocks.

System integration is a multidimensional process. At a given partitioning level, system integration is the process of blending the hardware and the software building blocks into a functioning whole. System integration is the process of combining the building blocks, functionally, logically, physically, and socially. The integration of the system with other systems and into the operational environment involves supportability, interoperability, and readiness activities, such as, repair, maintenance, logistics, training, and dealing with interfaces. Vertical system integration is the process of uniting building blocks at different levels—system within a system, wheels within wheels. System integration represents a major part of system engineering (1,5). The design of a CIM factory requires a methodology for integrating and controlling the diverse equipment in a manufacturing facility. The methodology must account for the hardware and software interfaces. Each manufacturing domain will have a generic interface model (GIM) to determine the common interfaces to the diverse equipment, and a framework for the development of machine control software. The basic hardware interfaces deal with sensors and actuators. However, there are several categories of software interfaces called middlewares.

System effectiveness is the source for the metrics to be used in evaluating performance, affordability, progress, and products of systems. Effectiveness is a desired result, outcome, or operation. System effectiveness is a measure of the ability of the system to accomplish its objective. It is a measure of the extent to which a system can be expected to achieve a set of specific goals. It can also be thought of as the probability P_{se}, that the system will achieve its objective. In this framework, the operator is considered part of the system. In manufacturing, the aggregation of the effectiveness factors are grouped into two sets of metrics, one at the factory level and the other at the process level (5).

Partitioning Concepts

In solving problems, the process of advancing from a coarse analysis to a fine analysis of a set of logical possibilities is achieved by means of partitioning. Similarly, the process of determining what should be in a subsystem, in an electronic package, in a software application, in a factory space, or on a solid-state wafer is a partitioning problem. For example, thermal management of an integrated circuit wafer, minimization of the number, and handling of signal lines are partitioning objectives. Also, given a system architecture, the issue of determining what should be performed manually, automatically by the hardware, or through software functions is a partitioning problem (1,5,6).

Partitioning involves several aspects of system design. Some of its pertinent objectives in CIM follow:

1. partition to identify functions and groupings of functions;
2. partition to optimize the interaction of subsystems;
3. partition to minimize the required hardware and/or software to implement a given function with a set of design constraints such as cost, reliability, thermal management, and maintainability;
4. partition to minimize the number of pins per package with a given set of constraints (i.e., select partitions that contain a maximum number of circuits with a limitation on the number of pins per package and chip area);
5. partition to minimize the time required to diagnose and locate a fault and the number of tests required to check a system;
6. partition to optimize the organization of interconnected packages, so that back panel wiring and the length of interconnecting cables are minimized;
7. partition to optimize the allocation of hardware and software in the factory;
8. partition to optimize the flow of material or operations in a factory.

In some cases, these objectives are overlapping or conflicting, and good engineering judgment is needed to select a compromise. For example, the thermal management of the silicon wafer must deal with the conflicting requirements of circuit density for a given wafer size. There are many types of systems, and each system must be considered on its own merits; it is expected that there will be compromises and tradeoffs to achieve the partitioning objectives and meet the constraints.

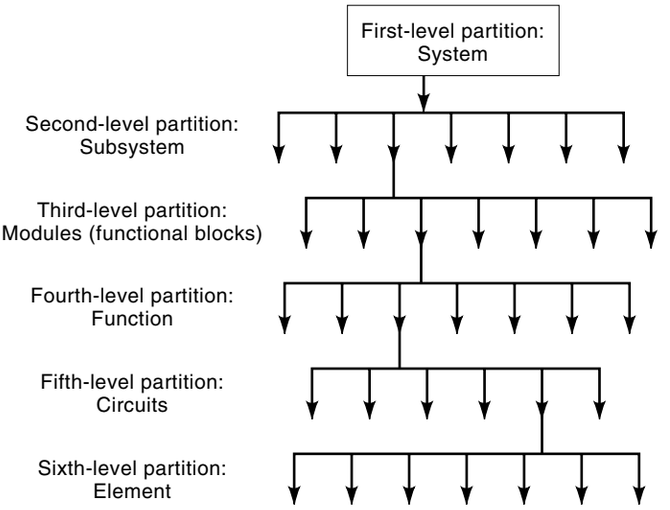

Figure 1. Partitioning levels of hardware systems representing a CIM factory. Figure shows six complexity levels of functionality and structure: system, subsystem, module, function block, circuit, and element.

Physical, fiscal, social, performance, or operational constraints are very useful in reducing the number of alternatives that must be considered (6).

The objective of system partitioning is to optimize functional and structural interaction of the system's building blocks with respect to design objectives. A software architecture requires functional partitioning and connectivity to generate the "instruction set." On the other hand, a hardware system architecture emphasizes functional and structural (physical) partitioning. Total system integration deals with both functional and structural partitioning. A given system can be partitioned into different functional levels according to topology or function, or both. The selected partitioning levels are based on complexity of function and structure. The following complexity levels are considered: system, subsystem, module (functional block), function block, circuit, and element, as shown in Fig. 1 (5,6).

In an integrated circuit (IC) CIM factory, for example, the system (level one) corresponds to the total factory consisting of machines, operators, managers, materials, computers, networks, and software. The subsystems or minifactories (level two) of the CIM factory are identified by the groupings of major operations that the factory must perform. The groupings can be the design and development of a product; integration of parts that make up the product; and testing of the product. The machinery, operators, managers, computers, databases, networks, software, and all other things that are needed to perform these operations make up the subsystems of the factory. The number of operations that forms a subsystem varies and can be large (e.g., 50–100 or more). The operations cover the total spectrum of manufacturing processes such as cutting, polishing, heating, slicing, etching, growing, moving, and testing (1).

The third or the module level of the factory hardware hierarchy is made of modules or functional blocks, where grouping of these modules make up a subsystem. The hardware modules can be a computer-controlled milling machine,

server, workstation, or database or a melting reactor for doping and growing materials. The fourth level in the hardware hierarchy is made up of function blocks. A grouping of function blocks makes up a module. A function block consists of machines, devices, or components that perform specific and simple functions such as drilling, washing, polishing, slicing, etching, adding, dividing, and multiplying. The circuit level (fifth level) is characterized by the interconnection of devices that perform simple processes such as amplification, rotation, linear motion, switching, and controlling. The aggregate of these simple processes contributes toward the realization of functions (e.g., circuits that control functions of pressing, heating, or speed control). Also included are logical gates such as electronics or fluidics gates (e.g., AND, OR, NOT, NAND, and NOR gates). Similarly, mechanical or pneumatic gates controlling valves, pumps, or heat exchangers in a reactor are examples of circuit level building blocks. The sixth or the element level in the hierarchy deals with devices, simple processes, or components that, when interconnected, perform the circuit-level operations. For example, the heating elements in a furnace, the valves of a heat exchanger, the pumps in a pneumatic circuit, or the electronic devices such as resistors, capacitors, ICs, ASIC (application specific integrated circuit) chips, microprocessors, motors, and generators, all are examples of components of a factory hardware hierarchy.

The vertical structure of the hierarchy consists of a multiplicity of structures nested within the system. The horizontal structure of the hierarchy is characterized by mutual support, interaction, or interdependence among the units at each level. Achievement of higher-level objectives depends upon the functioning of the lower-level structure.

The partitioning of a system into hardware and software is a critical system engineering task because it has a serious impact on the cost and performance characteristics of the final CIM factory design. Partitioning decisions performed by a designer or by a computer-aided-design (CAD) tool must take into account the properties and cost of the resulting hardware and software building blocks. The tradeoffs in the partitioning problem are based on the architectural assumptions, partitioning objectives, and market constraints.

SYSTEM ARCHITECTURE AND DESIGN OF CIM

The system architecture gives the top-down description of the underlying structure and functions of the system. System architecture provides the basis for separating the hardware from the software associated with the architectural building blocks. It gives a combined view of the system operations and the interaction of the building blocks. An integrated system architecture is a set of interconnected subsystems and their characteristics. In terms of design details, there is a hierarchy of system architecture. The hierarchy of system architecture is determined from the "flow" of decisions, information, materials, and operations that must be made to accomplish the objective. The flow is based on the functions to be performed and the constraints of the environment. The sequence of the flow will determine an architecture of the system. There are five classes of flow that should be accounted for: (1) decision flow, (2) information flow (i.e., signal, data, and document), (3) material flow, (4) operations flow (i.e., events, processes, and tasks), and (5) flow of housekeeping tasks such as workgroup tools (groupware) (e.g., e-mail, bulletin boards, fax). In general, there is a multiplicity of possible architectures. The determination of the optimal architecture is the subject for further evaluation and tradeoffs (1).

At the top level, the system architecture is represented by a set of interacting subsystems. A logical step in producing a detailed system architecture is to decompose the subsystems into individual hardware and/or software configuration units (CU) that can be developed or acquired. System engineering principles are applied to the process of decomposing the subsystems into configuration units to ensure that the configuration units are identified in a consistent and rational manner. The application of these system engineering principles to the decomposition of subsystems:

1. establishes the initial functional boundaries and interfaces consistent with the requirements around the CUs.
2. allows the identification of CUs to be uniquely assigned to a specific organization for development or acquisition.
3. determines which CUs are to be implemented by hardware and which are to be implemented through software.
4. identifies which CUs are to be supplied via commercial-off-the-shelf (COTS) products and which are to be developed.

Applying the same principles to the decomposition of all subsystems will ensure that the resulting CUs are identified in a consistent manner.

Software architecture is concerned with the high-level partitioning of software into major clusters and the specification of those clusters. It also includes the selection of the environment for software development (e.g., languages, library structure, and standards). Software design includes all phases of software development that precede coding, and as such includes software architecture. It also includes the iterative design activities of decomposing software modules, leading to a level of detail that allows code to be written in the selected language. Software development and hardware design are closely coupled. Tight linkages between them are important especially in software designed to be embedded in hardware where changes in either one usually require changes in the other (8–10).

The next step in producing detailed CIM system architecture is to represent the interacting CUs by the function block diagram of the system. The function block diagram gives the interconnection and the functionality of the building blocks.

In manufacturing, computers are embedded at all levels of the factory. They are used in process control in the form of microcontroller, field-programmable-gate array (FPGA), and digital signal processors (DSP). Personal computers (PCs), workstations, databases, and mainframes are used within the factory to communicate and to command and control functions. Computers and the associated software play a critical role in the functioning and operation of CIM. Network computing is used in computer-aided engineering (CAE), computer-aided design (CAD), computer-aided manufacturing (CAM), computer-aided testing (CAT), communication, data processing, finance, simulation, and virtual prototyping. Virtual prototyping is an integration of data from various sources that define the total product and its environment. The

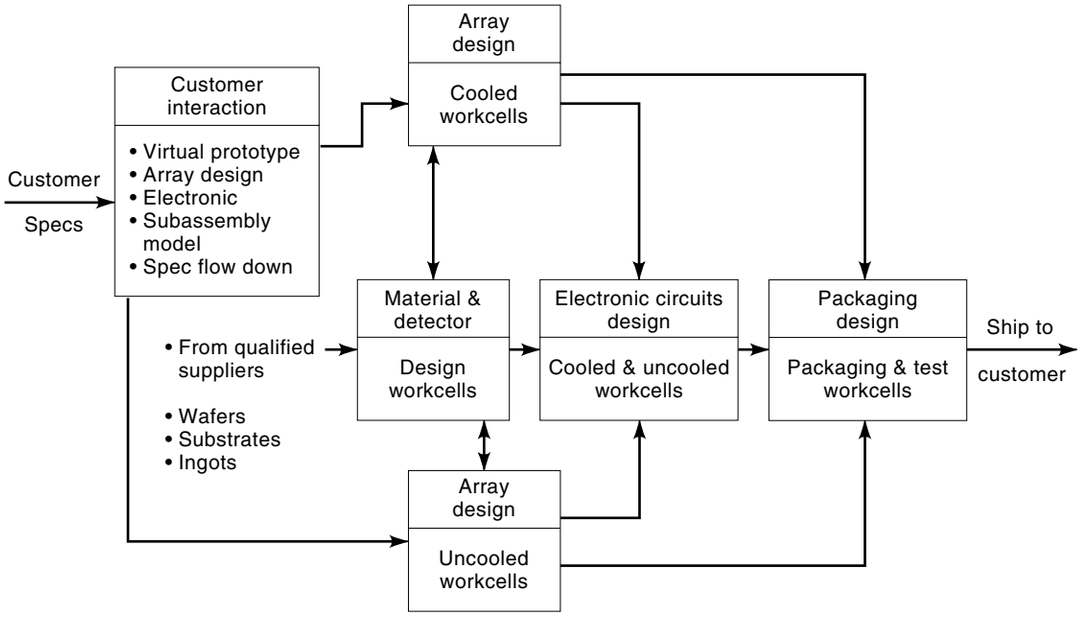

Figure 2. Configuration Units (CUs) of CIM factory to produce cooled and uncooled IRFPA. The factory architecture is derived from the flow of material and operations.

virtual prototype of a product is built electronically in three-dimensional solid digital models and evolves throughout the design process. In CIM virtual prototyping capability allows the customer to interact with the factory. The CIM factory receives customer specification through the internet, intranet, and/or the WWW. Through virtual prototype software the customer interacts with the factory management and/or engineering to generate a virtual product. The customer states a specification and the virtual prototype evaluates the specification and responds if the CIM factory can make the product, determines the cost of the product, and calculates when it can deliver the product.

The preceding principles are demonstrated in an example (i.e., the design of a CIM factory to produce infrared (IR) sensors). The following example is based on projects sponsored by DARPA to build a Flexible Manufacturing (FM) factory to make cooled and uncooled Infrared Focal Plane Arrays (IRFPA). A focal plane is a two-dimensional integrated circuit having infrared sensitive elements. The size of the array varies from 2×40 to 1024×1024 or larger. The IRFPA can operate at different wavelengths. Figure 2 shows the system architecture of a CIM factory. The factory is planned to produce cooled and uncooled infrared-focal-plane-array (IRFPA) sensors. The IRFPA factory configuration block diagram consists of six CUs. The diagram was derived from the flow of information, materials, and operations that must occur to make the products (1).

1. The Customer Interaction CU provides an evaluation of customer specification and the design of the IRFPA using virtual prototype software tools; it matches the customer specification to the factory capability.
2. The Material and Detector CU deals with all aspects of the detector material, such as preparation of substrates, deposition of thin films, photomasking, and mechanical handling (thinning, polishing, dicing).
3. The CU for Array Design of cooled IRFPA includes capabilities for array layout design, detector size, and interfaces with the electronics and the cryogenics.
4. The CU for Array Design of uncooled IRFPA deals with array layout design, detector size, thermal isolation, and detector interfaces with the electronics.
5. The Electronic Circuits CU deals with the design, integration, and manufacture of input circuits, control circuits, multiplexers, amplifiers, and interfaces with the detectors.
6. The Packaging and Testing CU deals with the final assembly of the focal plane array, design and manufacture of the dewar and cryogenic cooler, and the testing of the IRFPA (1).

In order to design the CIM factory, the factory software must account for the software needs within each CU and the connectivity with the rest of the factory. Software interfaces of the CIM factory provide very serious system engineering/integration challenges. Figure 3 gives an open system architecture block diagram of the IRFPA CIM factory. The diagram shows where the various applications software are used, such as computer-aided-design, computer-aided-manufacturing (CAM), computer-aided-testing (CAT), and just-in-time scheduling. The factory is connected through a local area network (LAN). The implementation of open system architecture is realized by using the client/server configuration. In CIM software interfaces are critical in achieving a seamless and smooth-running factory.

Figure 4 shows the factory client/server architecture, which uses a two-tier configuration. A client/server architecture is used because it provides: (1) factory flexibility at low

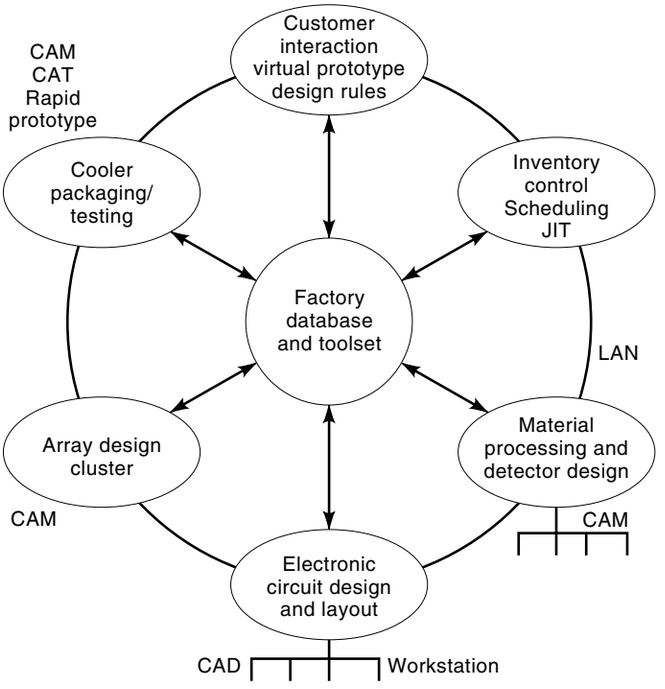

Figure 3. Block diagram of an open architecture of CIM of a typical IRFPA factory. The diagram illustrates the information and software connectivity of the CIM factory.

cost, (2) good end-user access to factory data, and (3) excellent scalability. The two-tier client/server configurations provide a simple division of labor. The client application, tier one, performs the user interface, application processing, and business logic. The server, tier two, performs the database management, usually by means of a Structured Query Language (SQL)-compliant Relational Database Management System (RDBMS). The client/server architecture is managed by database connectivity software such as the Digital Equipment Corporation (DEC) Framework Based Environment (FBE) middleware or the Microsoft Object-Linking-and-Embedding (OLE) middleware. In companies around the world, client/server (C/S) systems are helping people work together more efficiently, gain access to vital information, and improve their productivity. Client/server architecture requires software development tools to integrate the application within the system architecture. Visual Basic 4.0/5.0 for Windows or Visual C++ are popular 32-bit software development tools for wrapping in the client/server architecture. Software wrapping or encapsulation deals with the software interfaces that are necessary to make software applications transparent to the operating system. Visual Basic has a ready-made OLE middleware, and it can provide software wraps as remote automation objects on Windows server. Middlewares are programming tools that provide interoperability and portability within the various software applications. Middleware gives the system the ability to integrate separate hardware platforms, networking protocols, legacy applications, LAN environments, and desktop operating systems (1,11–14).

As manufacturers scale up the CIM architecture to enterprise-wide distributed systems, they are using more complex three-tier architectures. A three-tier client/server architecture provides a more comprehensive functionality spanning the enterprise and increases the system flexibility, but it is more complex. A three-tier architecture decouples the application logic from the user interface and the database. A three-tier architecture adds an intermediate layer of servers that supports application logic and distributed computing services. The intermediate servers facilitate scalability and reusability into the client/server environment. Instead of writing the same function or service in every application, application developers can write them once and place them on a server, which is accessible by all applications. Effective application

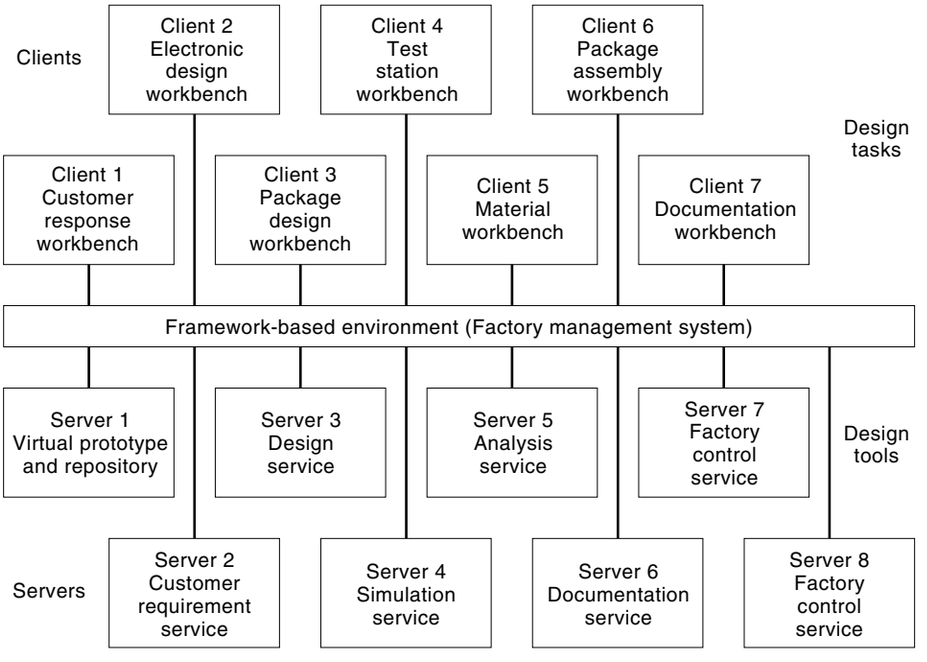

Figure 4. Two-tier client/server architecture using Framework Based Environment (FBE) in factory management. FBE is the middleware of the factory software, it provides the interface (hooks, protocols, wraps) between the various applications.

partitioning requires a solid understanding of the real-time requirements of the business and the technical issues of the application. Application designers and developers must establish where the application data will reside, be created, be read, or be updated. Also, it is necessary to determine: (1) how many users are expected at peak time, (2) the acceptable response time, and (3) how much data will be accessed by each transaction. A three-tier architecture splits the applications into three parts: (1) presentation part or client front ends, (2) applications functionality servers (process servers), and (3) database servers are shown in Fig. 5. The client front ends collect data for transactions and present the result to users. The clients terminals run on a variety of operating systems including UNIX/Motif by (Sun Micro Systems/DEC), Microsoft Windows by (Microsoft), and Apple Mac OS by (Apple Corp.) The client sends transactions to the application servers, which implement the business function. Application servers run on all major UNIX operating systems, open VMS (VAX Micro System), and Microsoft Windows. These servers communicate directly with the database servers, which can be Oracle, Informix, or Software AG's Adabas. The database servers usually run on UNIX, open VMS, or Windows. The following sections provide further details on software interfaces, open system environments, middleware, and distributed computing environments.

From a software point of view, after the architecture of the system is established and the subsystems and the configuration units are determined, the next step in the analysis is to group related software applications into clusters of modules (workcells). It follows that the system or enterprise software architecture is partitioned into clusters made up of software applications that will perform operations using the appropriate application: CAD, CAM, CAT, computer-aided software engineering (CASE), and CAE. The overall software system will be the interconnection and interaction of all the software clusters. Consequently, the software hierarchy is partitioned into six function and structure complexity levels: enterprise, cluster, module (domain), application, operation, and codeline, as shown in Fig. 6.

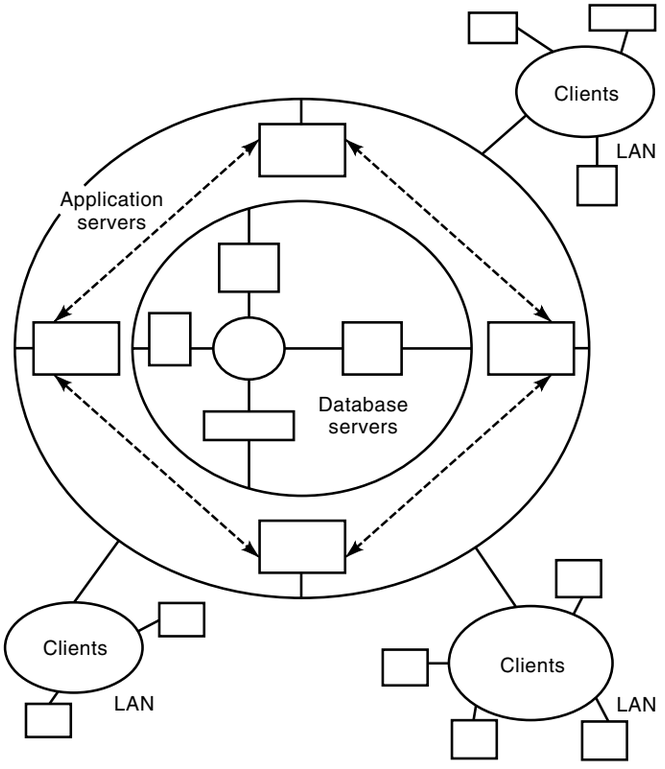

Figure 5. Three-tier client/server architecture. Client front-ends communicate with the application server via remote procedure protocol (RPC) and use data access middleware to communicate with the database server.

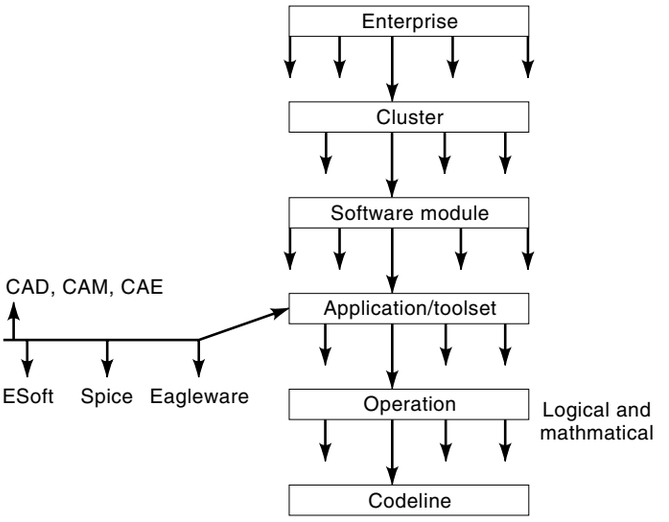

Figure 6. Partitioning levels of software hierarchy. There are six software complexity levels based on the complexity of functionality and structure: enterprise, cluster, module, application, operation, and codeline.

- The enterprise level represents the total software used within the organization, factory, or system. It includes the various software used in transaction processing, applications, design, services, networks, protocols, databases, and the like.
- The clusters level consists of groupings of software modules used in running mainframes, servers, desktop computers within a subsystem, or minifactories.
- The software modules (domains) level represents groupings of software applications used in running mainframes, servers, desktops, databases, or networks.
- The applications level consists of the various software tools, products, services such as word processing, transaction processing, spread sheets, toolsets, CAD, and CAM.
- The operations level deals with the variety of programming operations such as computing, storing, transferring, looping, formatting, and executing protocol.
- The codelines/source code level represents the codelines of software operations used to form the applications, services, or products.

The preceding methodology and concepts can be used to construct a broad spectrum of systems where software is embedded into the hardware (1).

SYSTEM INTEGRATION

System integration is a multidimensional process that is an important part of system engineering. At a given partitioning level, system integration is the process of blending hardware and software building blocks into a functioning whole. Vertical system integration is the process of uniting building blocks at different levels—system within a system, wheels within wheels. System integration requires complete understanding of the (1) input and output of the building blocks, (2) interfaces (interdependence) among the building blocks, and (3) operational environment as shown in Fig. 7. Interfaces are the keystones of system integration. Conceptually, interfaces are represented by the "intersections" of the sets of events associated with the building blocks. Mathematically, interfaces represent the interdependence between the functions of the building blocks. For example, an infrared (IR) sensor is made of several modules: detectors array, readout-integrated-circuit (ROIC), cyrogenic cooler, command and control electronics, front-end optics (telescope), and power supply. The manufacture of each module involves hundreds of operations that must be integrated horizontally. The IR sensor is made by vertically integrating the modules. As the modules are interconnected two, three, etc., at a time, more complex functionalities are obtained. Also, as the sensor is put together 2–4 modules at a time, the interfaces between the different modules grow rapidly. Interfaces are the glue of the system integration process and play a key role in the determination of the system performance drivers. For example, the interfaces between the telescope and the IRFPA are critical performance drivers in designing and building the IR sensor. The interfaces among the building blocks are:

1. Electrical: frequency response, bandwidth, power level, voltage, impedance, signal waveform, timing, current, data rate, transfer characteristics. Electrical interfaces are present when connecting or hooking up electrical electronic devices and modules.
2. Mechanical/Physical: pressure, torque, speed, gearing, flow rate, coupling alignment, bandwidth, size, weight, spectral response. Mechanical interfaces represent the coupling or connecting mechanical devices or equipment.
3. Software: logic, algorithm, protocol, data format, operating system, detection and correction codes, bus/LAN compatibility, timing. Software interfaces represent the software hooks, protocols, or wraps when integrating software modules/applications.
4. Social/Ergonomical: operator interface, safety, switchology, human factors, user friendly (1). Social interfaces are present whenever humans interact with machines.

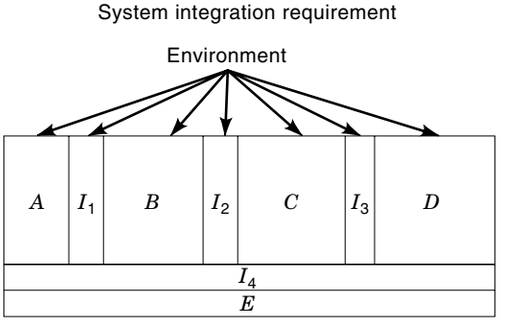

Figure 7. Building block interfaces, and environment. In the CIM factory A, B, C, D, E represent workstations, workcells, software module, or application. Interfaces (I) are the connecting events between the workcells or the software layers between applications.

In a CIM factory interfaces are at every operation, workstation, and workcell. Software interfaces are present every time applications interact with each other or the network. The software interfaces are addressed by the middlewares of the CIM factory. Electrical, mechanical, and human factor interfaces are present as the product goes through the factory. The interfaces between the hardware and software determine the structure, type, and size of memory, rate, size and type of data, operating system, programming language, and the type and structure of the middleware. Also, the study of interfaces between operations and processes provides insight about the cycle time of a given product.

The first step in the system integration process is understanding of the system requirement, the operational environment, and the functionality of each building block. The next step is determining the appropriate input(s)/output(s) of each building unit. The end result of system integration is driven by how well (1) the building blocks are connected (glued together) and interact with each other and (2) the functionality of the units are made available to provide the desired system output. The interfaces among the building blocks provide the required connectivity within the system. Interfaces are the means that make the functionality of the hardware and software of the building blocks available to each other to achieve the objective of the system. Mechanical, electrical, and human factor interfaces are very well understood except on occasion, when system integrators underestimate their importance resulting in a major redesign effort.

The integration of computers with communication networks and embedding computers into business and hardware systems pushed the industry into the distributed computing environment (DCE) [e.g., as developed by the Open Software Foundation (OSF)]. This created serious software interface challenges. The following discussion focuses on (1) embedded software in process equipment, (2) software interfaces between applications, and (3) networked interfaces.

The system developmental process consists of requirements flowing down a system hierarchy in the form of performance specification of the hardware and software with design specification as the final output. The effective integration of the system/enterprise requires the blending of the appropriate software with the hardware at each level. Consequently, the software is embedded into the hardware at each level as shown in Fig. 8. Examination of Fig. 8 reveals that the subsystem/cluster level represents the configuration unit level. It is at this level where the hardware and software are partitioned into hardware modules and software modules. The partitioning of the CUs into hardware and software requires detailed knowledge of the functions and operations that must be performed. Also, the figure shows that software has an impact on every level in the hierarchy. The diagram describes a snapshot in the system integration process, which

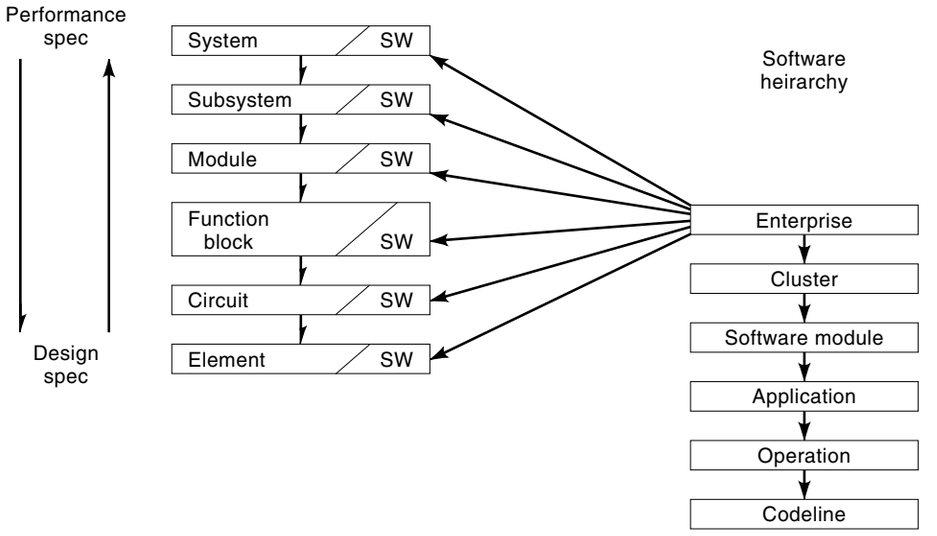

Figure 8. System Engineering design approach showing software embedded into the hardware. The figure shows a snapshot of the interaction of the embedded software throughout the CIM factory.

may require numerous iterations to finalize the system design specifications. Throughout the integration process, decisions must be made as to what should be automated (i.e., hardware and software) or what should be done manually by an operator. For example, the objective of CIM is to use computers and automation to increase the effectiveness of manufacturing; this requires examining the tradeoff of manual operator activities (touch labor) versus automated tasks that can be performed by machine. Automation is appropriate whenever it is cost effective and can show payoff considering performance, reliability, quality, cost, safety, and the like. The information flow of the factory must be analyzed to determine what should be implemented through the databases and software applications or in the client/server terminals. The analysis will provide insights into the tradeoffs between hardware, software, and manual operator activities. Today's products are differentiated and upgraded by changing hardware and more often changing software.

Embedded software is pervasive; it is built into CIM systems through the microprocessor or microcontroller, field-programmable gate array (FPGA), digital signal processor (DSP), and associated memory. Distinction is made between embedded systems design and design of self-contained computing systems. Self-contained computing systems are exemplified by computers. The term *embedded* means being part of a larger unit and providing a dedicated service to that unit. The computing elements in embedded systems are dedicated to performing signal and data processing or control and communication functions of larger systems. A personal computer can be made the embedded control system for manufacturing in an assembly line by providing dedicated software programs and appropriate interfaces to the assembly line environment. Similarly, a microprocessor can be dedicated to a control function in a computer (e.g., keyboard/mouse input control) and can be considered as an embedded controller. Embedded systems differ from traditional computers in the way in which they are programmed. Programming embedded systems has a serious impact on hardware organization and software compiler development.

The design of complex embedded systems in a CIM factory can be made more effective if system designers, developers, and users can participate and collaborate in the various stages of the development process. Collaboration in system design requires tools and applications that provide visualization of the system development process. Usually, complex systems are specified in a lengthy document that may contain fuzzy requirements. Collaboration and visualization will clarify and sharpen the understanding of the requirements.

The preceding system engineering principles are used to structure graphical simulations to provide meaningful visualization of system specification. The ability to see a visual representation of both the hardware and the software functions removes ambiguity from the specifications. Currently, graphical simulation and synthesis tools use a common visual design process called visual collaboration.

Advances in technology, the emergence of standards, electronic commerce (EC), electronic data interchange (EDI), metered software, global competition, and the availability of computer hardware and software are (1) providing the opportunity for automating the business processes and functions of an enterprise and (2) leading companies around the world to reengineer (restructure) their business practices. Through this reconstruction, enterprises are downsizing from mainframe-based systems to distributed and networked client/server implementation architecture (17). These advances in technology are increasing the number of CIM factories and improving the productivity of CIM, by lowering the development cost and reducing the cycle time of the product.

Distributed Computing Environment

Distributed computing is fundamental to the implementation of CIM. In manufacturing, distributed computing is consistent with the Distributed Computing Environment (DCE), as advocated by the Open System Foundation (OSF). In the traditional computing factory environment, there is no mechanism to share information with other applications. In other words, applications provide functions to the end user only and not to other applications. Each application maintains its own database and users must log into a time-sharing system to obtain simultaneous access to shared databases. In the CIM factory we find a collection of computers connected together to

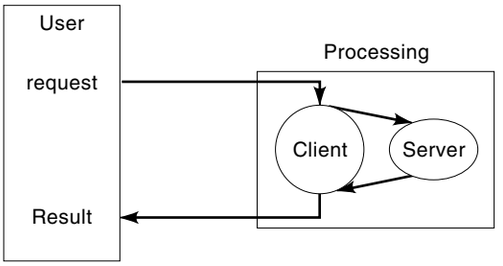

Figure 9. Distributed computing model using client/server architecture. The user interacts with the server/application through the client. Clients and servers are distributed throughout the CIM factory.

permit concurrent problem solving and sharing of information (databases). The architecture of distributed computing is based on an environment in which applications are divided into clients, which request services from a system or application, and servers, which provide services across the network on different processes. The DCE requires an overall enterprise management system that uses the client/server architecture running on a connecting network. A DCE using client/server architecture provides for the distribution of tasks that used to be done on a single machine. Complexities are added because the distribution of tasks can occur in a very large number of ways, since there can be any number of machines doing any number of tasks. Figure 9 shows distributed computing using the client/server model (18,19).

The DCE consists of six services: (1) remote procedure call (RPC) for communications, (2) cell directory service, (3) distributed file service, (4) distributed time service, (5) security service, and (6) threads package to create multitasking of software applications. Together these services act as middleware, the software that supports interactions between clients and servers.

RPCs are programming calls (messages) that process tasks executed by an application. Originally, RPCs were designed to connect two computers using synchronous connections. A protocol is a set of coded characters at the beginning and end of a message that enables one machine (as a computer) to communicate with another. It is a formal specification controlling the meaning of different messages passing between sending and receiving computers. There are two dominant nonproprietary protocols: (1) Open System Interconnection (OSI), (2) Transmission Control Protocol/Internet Protocol (TCP/IP). TCP/IP is the primary protocol used on the Internet. The DCE RPC allows developers to add or replace hardware, database, or the Graphical User Interface (GUI) as needed. It enables users and applications to communicate with other users and applications throughout the enterprise. The RPC mechanism provides protocol and network independence. The *cell directory service* is a database that contains the DCE cell resources, such as files, servers, disks, or print queues. The directory enables users to find these resources without knowing their locations. Currently, vendors are developing advanced directory and security services designed to make it easier for network managers to track (1) end users and (2) network resources across corporate Intranet and the Internet. Network managers are looking for a way to consolidate electronic mail (e-mail) authentication and security services, network operating system directories, and directories of Internet/Intranet users. This automation makes it easier for users and applications to share services, and it facilitates the management of the DCE. The *distributed file service* through the file transfer protocol (FTP) provides access to any file on any node in the CIM network. It makes a heterogeneous CIM network behave like a single system (18).

The *distributed time service* provides a mechanism for synchronizing the clocks on each computer in a DCE network. It provides time marking, which is used in many transactional applications. The *security service* provides authentication and authorization across a heterogeneous network. It validates that clients (users or programs) are what they say they are and determines whether they have access to the resources they request. Security permits applications to communicate with one another. There must be at least one security server for each DCE cell. Threads are program elements that enable multitasking. Threads allow servers to service RPCs and FTPs concurrently, which improves performance and memory use. In CIM, threads allow the industrial engineer to concurrently access process recipes, material inventory, status of products, and generate a cost profile of a product. Similarly, threads allow financial brokers to access account information, fund information, a portfolio list, and an equity transaction list from different servers concurrently. It follows that threads speed up the time it takes to create a customer profile.

Basically two components need to be distributed in a CIM factory—data and processing. Encapsulating these components as objects makes them portable, scalable, and sharable between applications. *Encapsulation* is a technique that allows the software developer to separate the implementation of an application from the abstract interface through software called wrappers. Users may develop these objects on proprietary platforms. However, they can aim at architecture that will permit them to port objects across a large number of standards-based platforms in the future. Keeping the logical description of the application separate from the physical description is very important, especially, when applications are moved from proprietary platforms, such as Windows, to standards-based platforms. Functions that previously ran in a proprietary environment are simply recompiled as objects in the industry-based environment. It follows that users save the time invested in developing the application and preserve the architecture of their applications.

Client/server computing is a combination of technologies that enables personal computers, Macintosh computers, laptops, workstations, mainframe, and other devices to interact with computers or software services to access the information and functionality of distributed applications. It involves networked remote services and applications that work with shared information across the CIM factory. In client/server architecture, processing is logically partitioned between independent, but cooperating, client/servers. Clients provide the interface into the application, and it is the client's function to present a coherent, usable, and appropriate interface into the application. The client also sends requests to the server and displays the results back to the user (application). Servers provide functionality by receiving and processing requests from clients and providing the results. Servers are distributed over the network to take advantage of existing hardware and software platforms across the network. The client/server architecture is becoming the design approach for future soft-

ware applications. The implementation of the client/server architecture is made practical by (1) the availability of low-cost powerful personal computers and fast networks to link them and (2) the need to access databases containing vast amounts of stored data. Many aspects of C/S application analysis and designs are similar to those of traditional architecture (20).

The partitioning of the client/server applications is user driven and is subject to tradeoffs between the capabilities of the client and server. For example, in a CIM factory a CAD application may require an extensive amount of data that reside on a separate database or mainframe. Therefore, the CAD application is partitioned accordingly. Embedded computers into manufacturing are used to perform many important functions, such as communication, supervisory control and data acquisition (SCADA), computation, storing data, design, and processing and distribution of signals, data, and information. The optimum integration, distribution, and partitioning of these functions within the CIM factory require a system engineering methodology to harmonize the operations of the factory. How, how much, and where to put the smarts of the factory are serious considerations in the design of a CIM factory. Should the factory smarts be distributed within the process equipment, data basis, application servers, or the mainframes? The answer to these questions is dependent on the manufacturing domain and is product-driven. The CIM factory architecture is very sensitive to the partitioning and distribution of embedded computers. There are several application-partitioning models. Each model has its strengths and weaknesses based on its suitability for different types of business and processing requirements. The models are distinguished by the division of the application components between clients and servers. The application components common to all models are (1) presentation, (2) business logic or function, and (3) data. The presentation services manage the user interface. The logic or function encodes the business rules that the application executes. The data consist of numerical information, text, images, or other types of information manipulated by the application. Typically, presentation services run on a client, data are kept on a server, and application logic runs on either the client or server or is divided between the two. Software developers must carefully allocate application components to minimize the impact of network latency and outages and to maximize the use of available computing resources. The basic issue is how well each architecture supports the applications given the constraints and requirements of the computing environments. The most common partitioning models are (1) the client centric, (2) the server centric, (3) the distributed function, and (4) the distributed services. A CIM factory will contain all four types of application partitioning models.

The client-centric model is shown in Fig. 10(a). In this model the presentation and logic components are placed on the client, and the data reside on the server. Client-centric applications allow users to generate SQL queries for transmission across a network to a database server that processes the request and returns a set of results. Most of the processing is made on the client, and the server just manages the data and the access to it by several clients. In CIM, the client-centric model is commonly used in process modeling and industrial engineering.

The server-centric computing model places the application logic and data on a server and distributes the presentation services to clients as shown in Fig. 10(b). The server-centric architecture duplicates the familiar centralized processing environment. This is important for companies running mission-critical applications that need to ensure data integrity and high availability. Collaborative applications such as document or image management work best with the server-centric architecture because they have heavy processing and storage requirements. The servers tier can be a single mainframe, minicomputer, or multiple LAN servers using operating systems such as Windows, UNIX, OS2 (by IBM), or Netware (by Novell). In any case, adequate bandwidth must be provided to accomplish the transfer of large documents or image files from servers to clients. The client in the server-centric model is referred to as the thin client or network computer. In CIM, thin clients are commonly used on the shop floor to perform process and product schedule, tracking, and monitoring.

The distributed-function model shown in Fig. 10(c) corrects many of the problems associated with the first two models. The distributed-function architecture allows developers to divide the application logic between the clients and servers and uses RPC messages to join the parts at run time. Using RPCs, the client side calls a procedure on a remote server and waits until it receives a response to continue processing. In a CIM factory many CAE, CAD, and virtual prototyping software tools give developers the ability to split application code between clients and servers using this model. This partitioning approach gives developers more flexibility to customize applications to satisfy specific business requirements.

The distributed-services architecture takes client/server computing to another level of complexity. The model breaks the hard-wired connections between clients and servers and adds support for more flexible forms of distributed computing. The architecture divides the presentation, function, and data into autonomous components that interact with each other across a network via standard interfaces, as shown in Fig. 10(d). The distributed-services model is also called a three-tier client/server architecture because the application components are frequently placed on separate tiers, as shown in Fig. 5. The three-tier C/S architecture have benefits for large-scale Intranet and Internet applications. The distributed-service model represents the CIM factory information system as discussed in the following section.

CIM Software

The CIM software is a distributed computing software of client/server, it is evolving into two groups—highly functional but proprietary environment and standards-based open system environment. The proprietary environment group has evolved out of the PC desktop world, whereas the standards-based group has grown from the microcomputer/workstation world. DCE is implemented using object-oriented technology. In general, CIM software is developed by using Object Oriented Technology (OOT). OOT includes OO tools, programming methods, and languages. Combining OO programming with distributed client/server architecture provides the capability of sharing and controlling CIM information throughout the factory. Objects are discrete program elements that contain data and the tasks to be performed on that data. An object is a concept, abstraction, or thing with distinct bound-

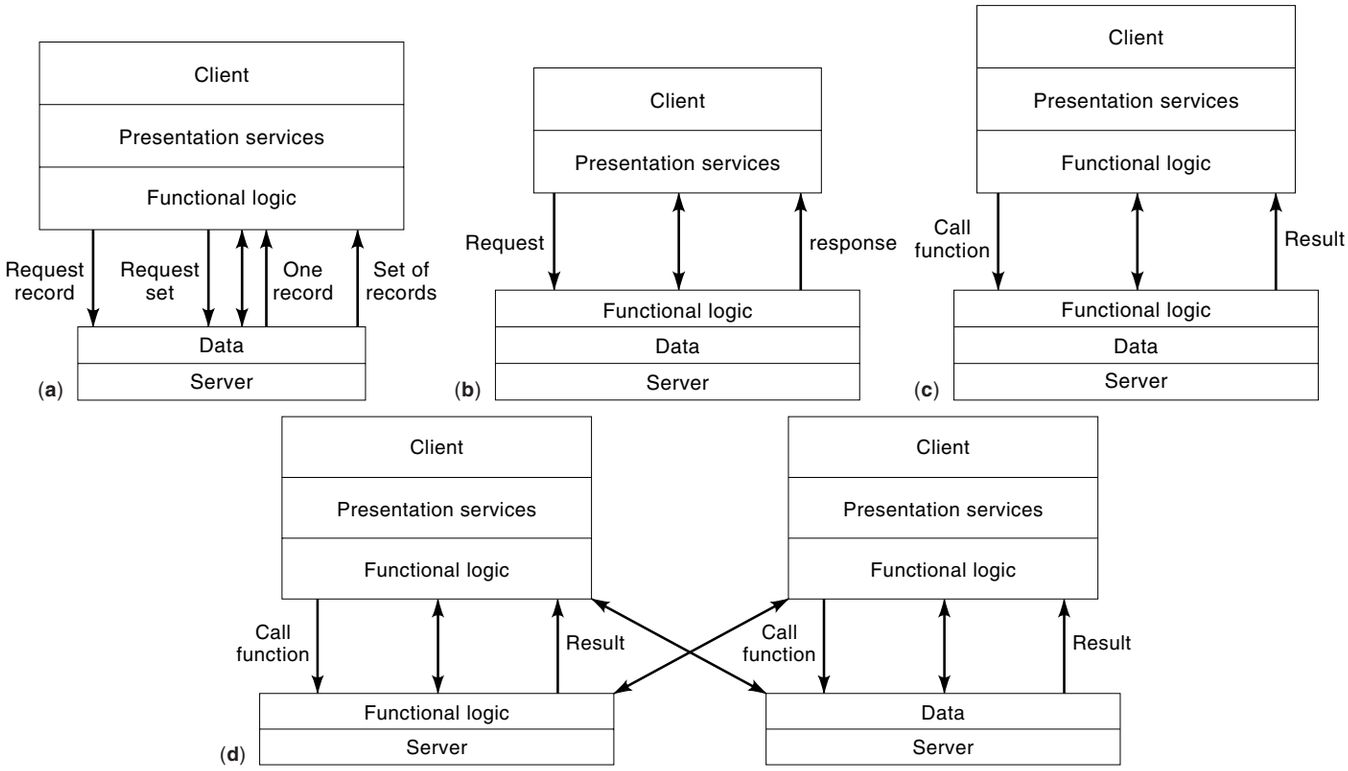

Figure 10. (a) Client-centric model: Presentation and functional logic run on client machines and data runs on the server. Client-centric model is used in industrial engineering to run CAD tools and process simulation. (b) Server-centric architecture: The client only handles presentation services, and the server mimics centralized computing (contains logic and data). The clients in this model are referred to thin clients or network computers. They are used on the shop floor to monitor and track factory processes. (c) Distributed-function model: divides the functional logic of an application into procedures residing on servers available to all clients. In CIM this model is used in CAE, Auto CAD applications, and simulation. (d) Distributed services model: Client/server computing components running on a server are deployed as shared services that can be used by a number of clients. In CIM this model is used in the implementation of the factory information system.

aries and meanings for the problem at hand. Objects describe elements that are common to a specific environment. For example, objects common to a manufacturing environment would include orders, schedules, and inventories. Objects containing common attributes are grouped into classes. Software developers working at the class level can influence the behavior of all objects within that class at once. Consequently, this approach improves productivity. The gains in productivity can be increased by storing individual objects and classes of objects in a central repository so that developers within the enterprise can share and reuse them. The software architecture of the CIM factory must have the flexibility to be useful in any manufacturing domain, and must be robust enough to support a variety of pluggable applications. The distributed OO client/server architecture is a unifying force that permeates every aspect of CIM, from its design and implementation to its appearance on the screen to the end user.

The most common object-oriented programming languages are C++, Next Step, Smalltalk, and Java. The Java programming language provides the interface that allows developers to create small applets that use the resources of the Intranet/Internet and particularly the World Wide Web (WWW), instead of the more expensive and limited resources of the personal computer. Applets are executable miniapplications. The Java environment provides a significant step in the process of making the Intranet/Internet a truly open and platform-independent network. True object-oriented products provide three basic functionality features that allow developers more flexibility. The first functionality feature is encapsulation, which refers to wrapping (bundling) data and processing tasks in a way that hides the individual procedure calls. The second feature is inheritance, which is a way for attributes or any changes made to an attribute within one class of objects to be automatically shared with subclasses created from the parent class. The third functionality feature is polymorphism, which supports the application of particular functions on different objects to achieve different results. Microsoft is advocating its ActiveX/Object-Linking-and-Embedding/Distributed Component Object Model (OLE/DCOM) architecture. The standard-based industry is backing the Common Object Request Broker Architecture (CORBA), as specified by the Object Management Group (OMG) and X/Open Co. Ltd. OLE is a Windows compound document protocol that allows one document to be embedded into or linked with another.

CORBA is built using Object Request Broker (ORB), which implements the mechanisms required to find an object specified in a client request and to communicate the data making up the request (21,22).

A three-tier C/S architecture provides the foundation to "distributed objects applications" where all application resources interact as peers. Manufacturing companies use three-tier architecture when they create a data warehouse. Production data on a host computer is periodically downloaded to a warehouse server optimized for user query and analysis. Network computing is a natural extension to C/S environment. Network computing deals with connecting the operators, accountants, and other employees to each other and the outside world. It is about connecting the enterprise to the customers and the business partners. The most common vehicle used to implement "network computing" is the Intranet/Internet.

Middleware/Software Interfaces

Software interfaces are the critical parts of any system integration effort. Interfaces ensure that proper software hooks, protocols, and wraps are used between the various clusters, workstations, and workcells of the CIM factory. There are six categories of software interfaces in CIM. (1) Protocols for the CIM system to gain information and control of operations, (2) protocols for CIM to process material on the machine, (3) interfaces with robots and protocols for moving material and executing transfers, (4) protocols for processing material and data file transfer (5) protocols for recording and accessing machine performance data, and (6) protocols for preventive and corrective maintenance scheduling and operations.

Today's multivendor realities add a subtle need for programming tools that will lead to truly successful distributed computing. In DCE, between the operating system and the application lies middleware. Middlewares are programming tools that provide interoperability and portability within the various applications. A simple definition of middleware is any software layer that resides between an application and its infrastructure environment, which isolates the application from the service component by providing an abstraction of them. It follows that middleware is a layer of software that sits between an operating system and network at one end and the applications at the other end. These services let developers write applications without having to master the details of the underlying services that will perform work on behalf of the application. The application program interface (API) is the software specification, and middleware is the software implementation of the API. There are three major reasons for using middleware: (1) providing basic transport from data source to a destination, (2) dealing with differences among platforms, operating systems, database servers, and protocols, and (3) protecting the software development team from the network complexity. Therefore, the middleware software program allows machines to communicate and interoperate by using a well-defined API. It provides the basic data transport from source to destination, giving users a single view into a heterogeneous environment. The ability of the enterprise to integrate different multiple hardware platforms, networking protocols, legacy applications, LAN environments, and desktop operating systems is realized by the middleware software. Middleware is an information broker that controls and manages the flow of information and processes between clients and servers. For example, the middleware layer uses its own communication code to set up the session with the target environment, initialize protocols, handle error recovery, and return a delivery confirmation to the workstation. Middleware ties the different components of a distributed system together into a single logical environment by providing consistent APIs that are independent of operating systems, networks, and databases. Using middleware enables application developers to write applications independent of operating system interfaces or specific network transport. Examples of middleware are the DEC Network Application Support (NAS) and Framework Based Environment, Hewlett Packard's Open View, and Microsoft OLE (23–26).

There are four types of middleware: (1) distributed database middleware, (2) communication or message-oriented middleware, (3) transaction-processing (TP) middleware, and (4) distributed object services middleware. A CIM factory software will have all four types of middlewares.

The distributed database or data access middleware makes databases within the enterprise available for applications. It provides services that are consistent across diverse databases (e.g., it facilitates communication between client applications and databases). The data access middleware provides a common high-level programming interface, such as SQL, that packages information so that it can be used by multiple applications. The data access middleware is used in the three-tier CIM to (1) bring data from databases for analysis and process control, (2) perform File Transfer Protocol (FTP) and integrate legacy and LAN-based data with web servers.

Communication middleware enables developers to partition applications and distribute them across the network. The two main forms of communication middleware are RPC and messaging. RPCs let application components on different machines converse synchronously across network. RPCs rely on a simple synchronous call return API. Messaging middleware enables application components to communicate asynchronously, meaning that users do not have to wait for a response. Consequently, an application that sends the message is not blocked and can do other tasks. RPCs are generally considered simpler than messaging, but messaging can be more efficient. The messaging middleware adds a layer of software that becomes a common denominator for all kinds of different technologies. Message-oriented middleware resides between applications and operating systems to facilitate interaction with other applications and operating systems. Message-oriented middleware stores the messages in a local or remote queue until the target is ready to receive it. The receiving target polls the queue until there is a message for it. The asynchronous communications ensure that an application is never forced to stop execution while it is waiting for a message. In CIM, the RPC middleware is used to establish channels of communication between clusters, workstations, and workcells. Message queuing is not appropriate for event-driven applications, such as GUIs or closely coupled distributed processing applications (14).

The TP middleware handles the detailed work of managing traffic among multiple application components and coordinates the execution of such tasks as job scheduling, resource location, data updating, and recovery from systems failure. It is designed to develop and manage online transaction pro-

cessing (OLTP) applications (27). The TP middleware is the heart of the virtual prototype of CIM.

The networking breakthroughs such as mobile and wireless networks, LAN switches, Integrated Services Digital Network (ISDN), and Asynchronous Transfer Mode (ATM) are making LAN and WAN indistinguishable. Global businessmen ventures created *virtual workgroups*—people working together on common projects that are distributed across the country and around the world. Switchable communication protocols such as TCP/IP, shuttled from LAN through Wide-Area-Network (WAN)-to-LAN by multiprotocol/router products, make remote network access transparent to the virtual workgroup. Businesses are investing many resources into electronic networks that link computers, databases, and other information technologies. The rationale for doing this is that they must have the capability to (1) track all the components and products, (2) synchronize deliveries, (3) keep engineers and marketers informed of each other's plans, (4) alert the research and development organization to the needs of the manufacturing side, and (5) give management a coherent picture of what is going on. Accomplishing these objectives keep the businesses competitive (28–31).

The development of the software for using DCE requires examining the layers of the software building blocks as they relate to the proprietary and standards-based client/server software. The software layers arise from the presence of distributed control networks within the DCE. The Open Systems Interconnection (OSI) of the International Standards Organization (ISO) provides the most robust standards and models for distributed data networking. The OSI model is a seven-layer reference model for computer networking, which is applicable to a CIM network in which the processing elements communicate over a common path. The seven layers are (1) application, (2) presentation, (3) session (operating system), (4) transport, (5) network, (6) data link, and (7) physical. Fig. 11 shows a seven-layer OSI reference model for computer networking and control (10).

The top layer is the application layer, where the user interacts with the CIM network application. Data are received as commands from the user. TCP/IP applications communicate in client/server pairs at this layer. Application protocols include FTP for file transfers, Telnet P (Telecommunications Network Protocol) for remote terminal sessions, SMTP (Simple Mail Transfer Protocol) for electronic mail and SNMP (Simple Network Management Protocol) for network management. At the presentation layer, middleware is used to determine how shared data are formatted for transmission across the network. At this layer, data format such as ASCII text and binary documents are broken into messages. At the session layer the DCE interfaces with the operating system of the machines, and the distributed computing facilities act as extensions to the operating system. Each message is serialized by adding session layer (operating system) header information. The serialized session layer elements are tagged with transport layer recovery information so that the communicating parties can request retransmissions of lost or garbled messages. The transport layer manages the data flow between two internetwork hosts. The transport layer messages are broken down into routable packets and datagrams whose network layer headers identify the sender and receiver by their corresponding node numbers. At the network layer, data are transferred around the internetwork. The Internet Protocol (IP) is the dominant protocol used for routing packets between hosts and across network links. ICMP (Internet Control Message Protocol) works with IP to carry control messages and status information between networked systems with IGMP (Internet Group Management Protocol), which routes data to multiple hosts for multicasting. Both ICMP and IGMP are carried with IP traffic. A set of methods called Internet routing is used to make sure that network traffic is

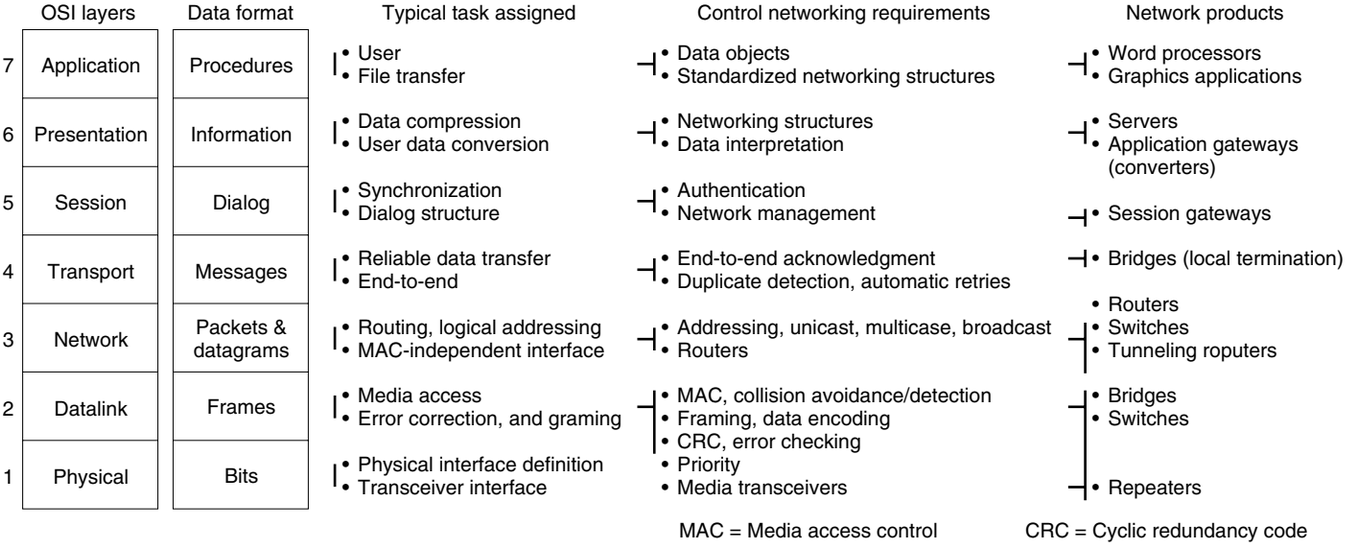

Figure 11. The International Standard Organization (ISO) seven-layer reference model for computer networking structured on the Open System Interconnection (OSI) standard. Software interfaces represent the transition from layer to layer.

passed between hosts and networks efficiently. Internet routing has its own set of protocols to ensure that the special systems that connect networks, called routers, are able to detect changes in internetwork routes and maintain data flowing across the Internet. The network layer is very well standardized and presents a consistent, transparent interface to the session layer. The technologies that constitute the network are understood and well established. The issues network designers must consider are (1) system protocols, (2) interoperability, (3) command status, (4) security, and (5) topology. The network protocol must be open, to ensure the availability of the network to everyone connected to it. The data link layer is also known as the network interface layer. Data are transmitted across a network at this layer. Because this layer deals with local network transmission exclusively, there is no universal TCP/IP protocol available. Two commonly used protocols called ARP (Address Resolution Protocol) and RARP (Reverse Address Resolution Protocol) are used on many networks such as Ethernet and Token Ring. Note that these two protocols are sometimes viewed as being part of the network layer. The physical layer transmits the signal along the various networks in the form of electrical pulses. Finally, these messages are transmitted through the data link and physical layer protocols of LAN and WAN cables. Presently, the Internet Protocol is going through several updates in order to meet the security and interoperability requirements. IP version 4 (IPv4) currently is widely deployed in routers and desktop software. The next generation of IP, IP version 6 (IPv6), includes an improved network addressing method plus support for moving PCs across subnets without changing their IP addresses. Also it includes optional security features such as authentication and encryption.

Managing the enterprise information is a mission-critical and challenging task. System engineering principles provide the basic tools for managing the information of the enterprise. The following section takes as an example a CIM factory and applies the preceding system engineering principle to the design of the factory information management system.

INTEGRATION OF FACTORY INFORMATION

Computer integrated manufacturing represents our best approach to staying competitive in a global economy. System engineering principles are used to determine the range of products to be made, the factory capacity, and the factory building blocks. The preceding guidelines are used to establish the architecture of the IRFPA CIM factory using commercial-off-the-shelf software applications. The factory is structured into interconnected clusters of workcells. An effective CIM factory provides on-demand, correct, and timely information to every participant in the integrated team including the manager, design engineer, process engineer, floor supervisor, operator, and accountant. A factory information management system (FIMS) is needed to tie the clusters together. Using a LAN and the FBE middleware, the FIMS connects the participants to (1) shop floor control, (2) process equipment control, (3) product data management, (4) process modeling, and (5) support tools.

Workstream from Consilium is a complete factory floor control system. It generates and maintains process instructions and handles recipes for automated machine control. Workstream provides the following functions: (1) Work-In-Process (WIP) tracking, (2) data collection from workcells, labor times, losses, rework, (3) directing data from design cells-to-processes-to-databases, (4) transferring data from databases for analysis, and (5) interacting with the process control software that controls equipment. Workstream runs on VAX using UNIX. It forms the backbone of the FIMS and serves as the source for the basic manufacturing data.

Cellworks is a COTS product from Fastech, Inc. It runs on DEC Alpha server using UNIX. Cellworks automatically generates data fields from Workstream. Workstream and Cellworks interact to document process recipes, product routing, and other information describing how the product moves through the factory floor. Cellworks maintains a consistent screen layout throughout the factory. Users connect to Cellworks through PCs running Microsoft Windows. When appropriate, Cellworks launches Windows-based views such as Microsoft Word or Excel or Borland Paradox to assist with data review and entry (1,18).

The rapid generation and retrieval of information by CAD/CAM/CAE and CIM strain the conventional configuration management system. The issue is further complicated by having the information stored in different format media and resident on different and dispersed computers. The CIM factory requires an engineering data management (EDM) system to automate the management of product design data. A COTS software called Product Data Management (PDM) from Computer Vision is used to automate the factory design data. PDM is a tool that helps engineers and others manage both data and the product development process. PDM keeps track of the masses of data and information required to design, manufacture and support a product. The PDM runs on a Sun workstation using UNIX.

The ability to model processes and factory capabilities leads to improved yields and better matching of product specifications to the factory. Models support the development of reliable design rules that enable factory flexibility. The building of models consists of identifying the variables in a factory database as inputs or outputs and programming the computer to handle the calculation automatically. Process Insights is a commercial software program from Pavilion Technologies that uses neural network and fuzzy logic technology to model complex nonlinear processes.

Sound and rapid management decisions require information from the entire factory. The manager, supervisor, and engineer need data on cost, schedule, technical specification, and resource data. The information exists as data and tools distributed throughout the factory. FIMS provides access and organization to draw resources from across the factory and to present the results in formats that help managers. Figure 12 illustrates the connectivity of the various software packages in the FIMS architecture.

Defects are a very serious manufacturing problem, and reducing defects will improve the yield and the competitiveness of the factory. An important payoff from CIM is the ability to perform on-line statistical quality control (SQC) to reduce defects in a timely manner. Figure 13 illustrates how CIM performs on-line SQC. For a given product, FIMS performs a series of tasks: retrieves control limits, locates files, and loads historical data. Cellworks reads current data from the process equipment, checks it against historical data, and sounds the alarm when the data exceeds control limits. A similar se-

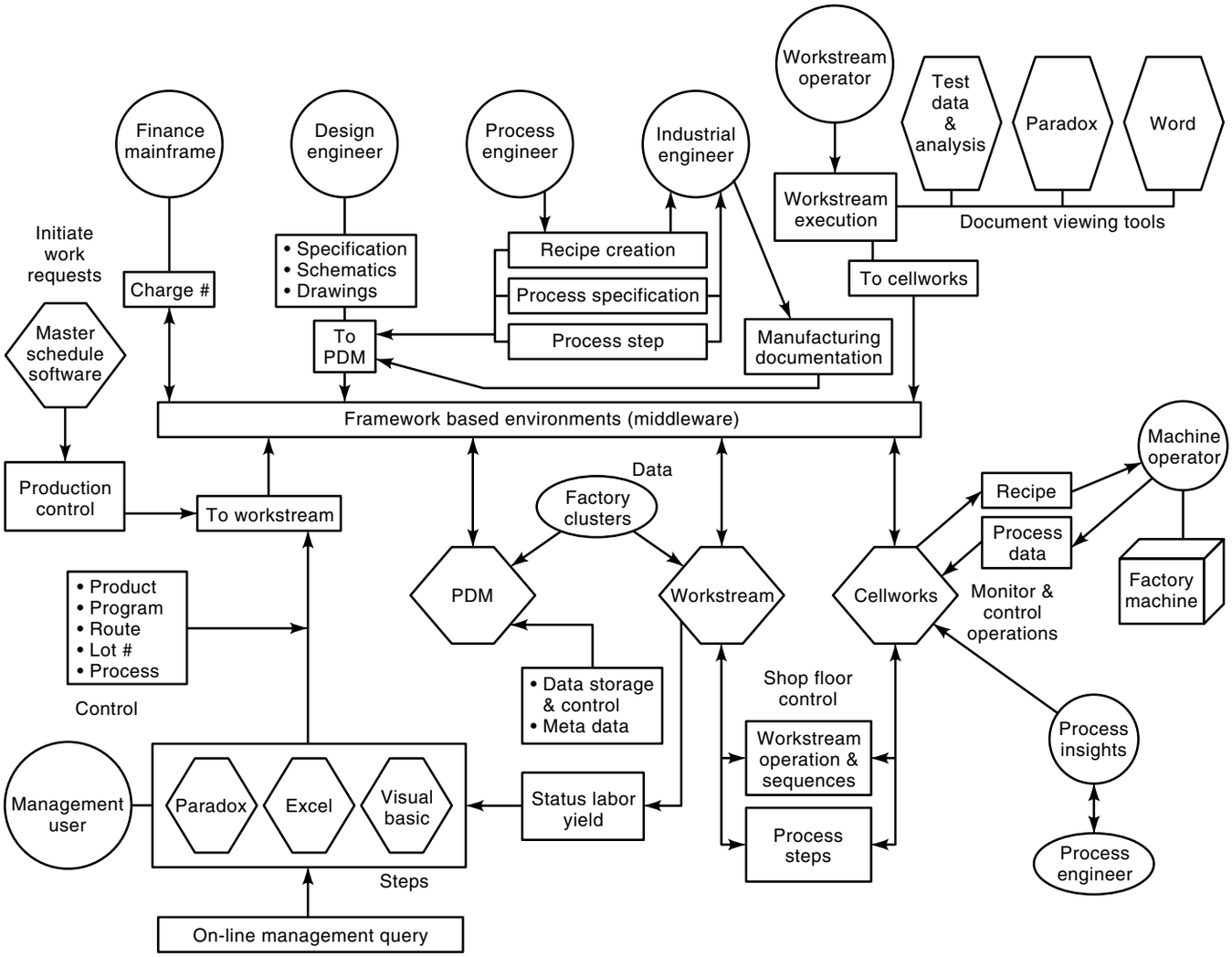

Figure 12. Factory Information Management System (FIMS) gives the architecture of the building blocks of CIM: clusters, workstations, workcells, and applications are distributed in the factory. Interfaces/middlewares are represented by arrows and junctions.

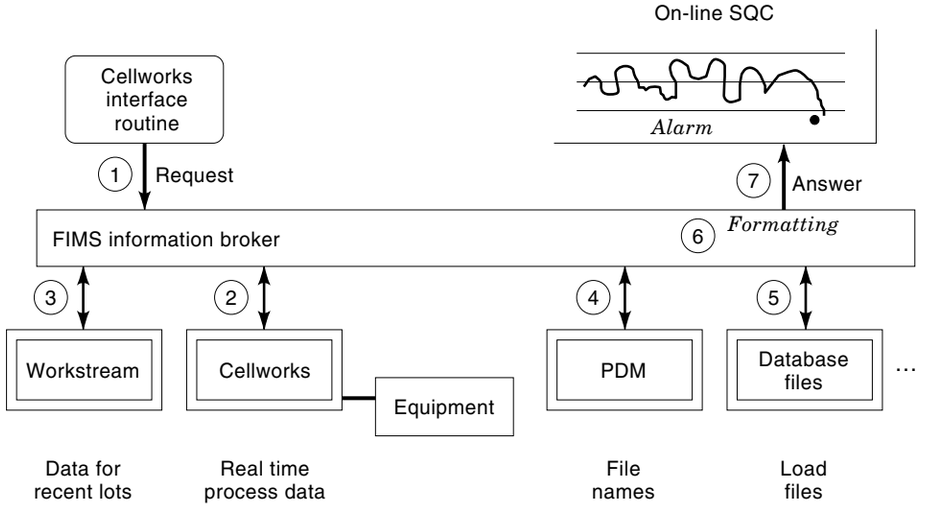

Figure 13. On-line statistical quality control: cellworks reads and checks data against process history.

quence of calls to applications and data files occurs for each element of the engineering analysis.

METRICS: MEASURES OF EFFECTIVENESS

Systems are evaluated in terms of their outputs or the outcomes they can provide. The building block model of systems provides the basis for determining the total factory outcome. The factory (system) sample space is a mathematical model that represents all the factory outcomes. The outcomes of a factory are based on all the possible interaction of the building blocks. Consider a factory consisting of n interacting building blocks. The number of possible system outcomes is the total number of combinations of the building blocks interactions taken 0, 1, 2, 3, . . ., n blocks at a time; this gives 2^n outcomes. Figure 14 gives the system space for $n = 2$, 3, and 4 using the events diagram, where the system space consists of $2^2 = 4$, $2^3 = 8$, and $2^4 = 16$ disjoint regions R_d (outcomes).

Networks are also used to represent the system space. In the network presentation, the terminal nodes represent the system outcomes as shown in Fig. 15 for $n = 2$, 3, 4. It follows that the network presentation of a system consisting of n building blocks contains $N_t = 2^n$ terminal nodes. In this formulation the factory is thought of consisting of n minifactories. Each minifactory consists of several modules of many workcells. Each workcell covers several operations, where each operation depends on the occurrence of many processes. The outcomes of the factory are driven by the states of the minifactories, modules, workcells, operations, and processes. Equation (1) gives the relationship between the number of disjoint regions R_d, the number of terminal nodes N_t, and the number of factory outcomes.

$$R_d = N_t = 2^n = \sum_{m=0}^{n} C_m^n \tag{1}$$

where, $C_m^n = n!/[m!(n - m)!]$ is the number of combinations of n blocks taken m blocks at a time. For a given integrated system made of (n) building blocks, the number of interfaces (I) between these parts is

$$I = 2^n - (n + 1) = \sum_{m=2}^{n} C_m^n \tag{1a}$$

Equation (1a) gives the number of combinations of (n) blocks taken (m) blocks at a time starting with $(m) = 2$. In some integrated systems, these interfaces are not distinct, because high-order interfaces subsume low-order interfaces.

System effectiveness is the term often used to describe the overall capability of a system to accomplish its mission. A measure of effectiveness (MOE) is any index that indicates how well a system is achieving its objectives. The effectiveness of the CIM factory is a function of the hardware and software design adequacy, reliability, and readiness. A computer software crash can result in the shutdown of the CIM factory. Consequently, the effectiveness of CIM is severely degraded by the software failure. A computer crash can be due to software design limitation, software unreliability, and/or software unreadiness. The design adequacy of CIM deals with the performance/capability of each piece of equipment and the software robustness of every software application in the factory. Design adequacy of software deals with the capability attributes of the software product. It is a measure of the quality of performance attributes of the application. The desired attributes of the software application are functionality, robustness, portability, reusability, and interoperability. Functionality of a software product is a measure of the quantity and quality of functions it performs. Robustness is a measure of the extent to which an application can continue to operate correctly despite the introduction of new or invalid inputs. The reliability factor of CIM accounts for the hardware and software reliability. The readiness of CIM is a measure of its availability and operability to produce the products. Hardware availability is a measure of the corrective and preventive maintenance of the equipment in the factory. Software maintenance includes performance upgrades and it is divided into three classes: corrective, adaptive (preventive), and perfective. Corrective maintenance is software repair; it concerns the fixing of faults (errors, bugs). Adaptive maintenance deals with changing the software application to accommodate changes in the environment, for example, altering the application so it runs on a new computer or operating system or network. Perfective maintenance deals with upgrading the functionality of the software. For example, improving the application to satisfy the user needs, rewriting documentation, improving efficiency, or the human interface.

A CIM factory is software intensive. Software reliability, robustness, and readiness contribute to the effectiveness of the factory. All software is subject to failure due to the inevitable presence of errors (faults, bugs, anomaly) in the soft-

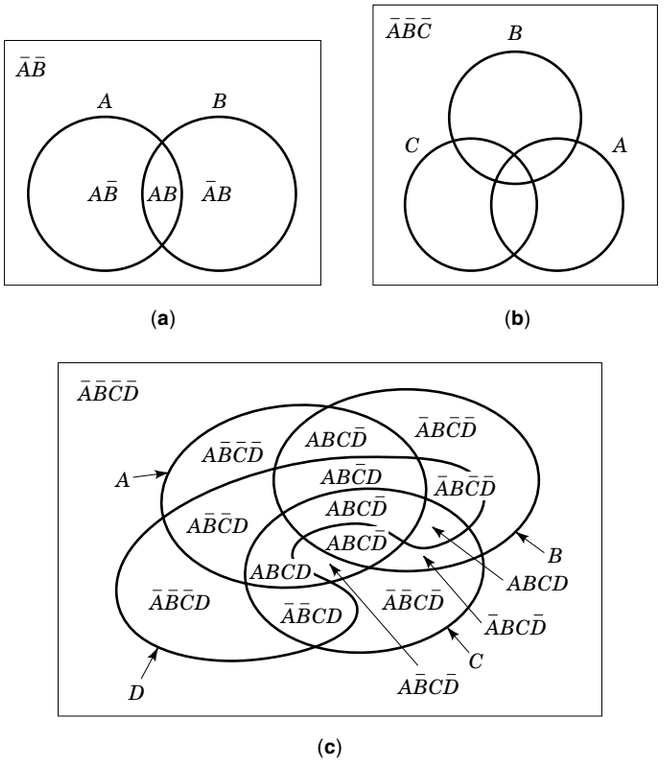

Figure 14. Events-diagrams: (a) system space diagram, $n = 2$, $R_d = 4$; (b) system space diagram, $n = 3$, $R_d = 8$; (c) system space diagram, $n = 4$, $R_d = 16$. In the events-diagram, the independent variable, e.g., time or factory floor space are implicitly shown.

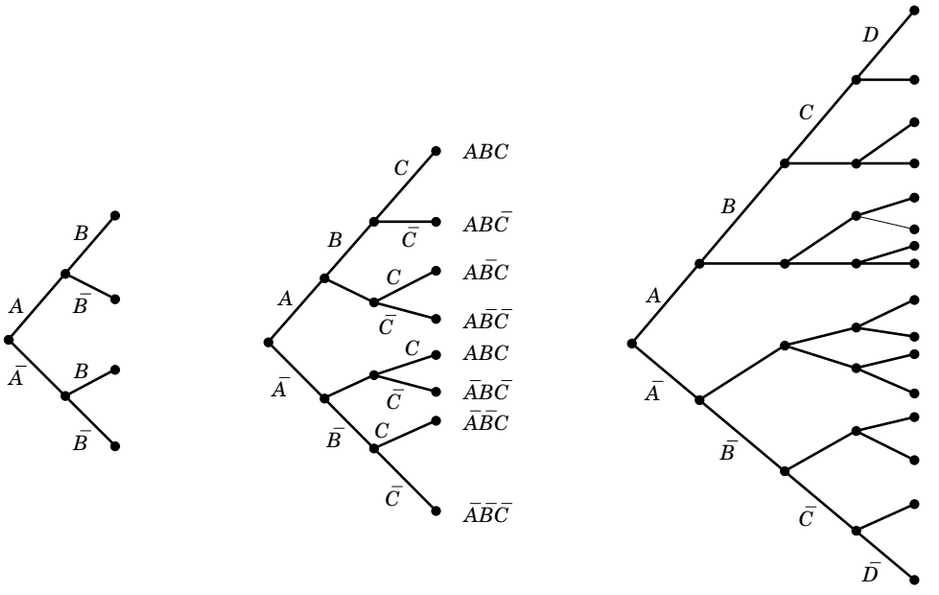

Figure 15. Network-diagrams: (a) $n = 2$, $N_t = 4$; (b) $n = 3$, $N_t = 8$; (c) $n = 4$, $N_t = 16$. Network diagrams of CIM show the independent variables explicitly as the network is constructed.

ware. Software reliability is the probability that the factory software will not cause the failure of the CIM factory, for an interval of time t, under specified conditions. Software reliability is a measure of the frequency of degradation in factory performance that is due to software failure. Software reliability is different from hardware reliability because software does not wear out like mechanical or electrical devices. In physical systems like aircraft, radar, motor, or furnace, reliability is related to wear and tear on the parts of the system. In software there is no wear or tear. A certain application may be used for years without encountering an error simply because an error exists in a rarely traveled path of the software. Fundamentally, software reliability is a measure of the quality of design and implementation of an application. If properly designed, tested, and correctly implemented an application should run without failure for all inputs and uses. Consequently, software reliability should increase with use because the probability of failure decreases with time. However, it is possible for software reliability to initially increase as the bugs are removed from an application, then decrease as a result of changes in the inputs to the application and/or changes in the functionality of the software. This is because these changes lead to new travel paths of the software where errors reside. In manufacturing, the MOEs of the CIM factory are grouped into two classes:

1. Attributes measures (discrete), such as number of defects or nonconformities in parts per million (ppm), cost/unit, rolled throughput yield (percent of defect-free products), and producibility measures (number rejects/number produced).
2. Variable measures (continuous) can be separated into two categories:
   (a) process variation parameters measured by using Shewhart charts R, X, σ.
   (b) process control capability measures: (1) C_p, C_{pk}; (2) cycle time, time from receiving order to time product is out of the factory; and (3) yield, percent of product meeting specification (32,33).

In manufacturing, the aggregation of these factors are grouped into two sets of measures: (1) measures at the factory level and (2) measures at the process (operation) or component level (1).

MOEs at the Factory Level

The objective of CIM is to provide quality products, improve yield of factory, reduce cycle time, increase the range of products, and reduce the unit-product-cost (UPC) by reducing the recurring and nonrecurring cost of producing products. Another attribute of CIM is enhancing the ability of the factory to produce at low cost at low volume. The CIM factory flexibility is based on the ability of the factory to produce low-cost products at low volume. The decoupling of UPC from volume is illustrated in Fig. 16. The cost-versus-volume curve consists of two regions: (1) low rate region characterized by the slope m and the knee of the curve and (2) high-production-rate region characterized by the ordinate of the asymptote. Figure 16 shows two measures of low cost at low volume: (1) the ratio of the knee of the curves (e.g., case 3 is more low cost at low volume than case 1 by a factor $V_1/V_3 = 5$) and (2) the ratio of the slopes m_3/m_1. These relationships can be determined by factory simulation of many runs. It is recognized that productivity of a new factory tends to increase in time. The concept of the learning curve has been around for some time and has been used as a measure of productivity. Equation (2) illustrates the concept:

$$C_n = kn^{-x} \tag{2}$$

where C_n is the unit cost of unit n; k is the cost of first unit produced that met the specification; x is the learning curve coefficient, characterized by the cost reduction obtained by doubling n. That is,

$$C_{2n}/C_n = (2n)^{-x}/n^{-x} = 2^{-x} = a \tag{3}$$

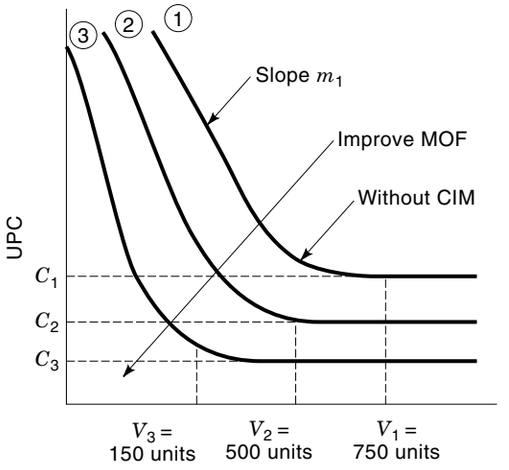

Figure 16. Cost versus volume curves illustrate low cost/low volume measure of effectiveness. Factory flexibility is demonstrated as the factory operates at a higher volume level by aggregating several low-volume products, as shown in curve #3.

The factor a is the fractional reduction in cost obtained every time the volume is doubled (3). For example, if $a = 0.85$, the learning curve is said to be 85%. Taking the natural logarithm of Eq. (3) gives the relationship of x to a as

$$x = -\ln(a)/\ln(2) \tag{4}$$

The manufacture of the Infrared Focal Plane Array (IRFPA) is a complex, highly technical, and labor-intensive task. A basic benefit of CIM is the ability of the factory to decouple the UPC from volume (i.e., produce products at low cost at low volume). Figure 16 shows that the state of decoupling occurs when the factory is operated beyond a threshold volume (i.e., beyond the knee of the curve). This principle is used as a guide in setting the capability of the CIM factory. The interest in applying the learning curve equation as a factory metric is to get a handle on the cost of producing IRFPAs. Through market research on a potential need for IRFPAs and the desired capability of the CIM factory, it was determined that a production rate of $n = 250$ IRFPA per month is feasible and will allow the factory to operate beyond the knee of the UPC-versus-volume curve. Another benefit of CIM is reducing the nonrecurring engineering (NRE) cost. Based on current experience, on the average the cost of the first IRFPA unit without CIM is $k = \$700{,}000$ and with a CIM is $k = \$500{,}000$. Through simulation of a CIM factory operating just below the knee of the curve, it was determined that the learning curve coefficient $x = 0.71$. Using Eq. (2) and substituting for $k = \$500{,}000$, $n = 250$, and $x = 0.71$ gives on the average the projected UPC = \$10,000. The fractional reduction or the learning curve rate $a = 0.61$. This example provides a good rationale and justification for computer integrated manufacturing.

Cycle time is a factory MOE. It is estimated by breaking down the work flow into steps and determining the time it takes to perform each step in the manufacturing process. The proper application of CIM and automation play a key role in reducing cycle time. Through simulation it was determined that the cycle time for a CIM factory is reduced from 9 months to 3 months.

Parts Per Million and Throughput Yield

Making complex devices such as IRFPAs involves handling material and integrating various components and requires several hundred parts and operations. If these parts and steps are not perfect, or do not meet very high standards, the products will have defects. The concept of parts per million is used to characterize defects of products and is based on the area under the curve of the standard normal probability density function. The standard deviation σ is used as the describing parameter.

Consider a product produced under the tolerance specification of $\pm 3\sigma$. Where the area under the curve is $A = 0.9973$ and $(1 - A) = 0.0027$; therefore, out of 1,000,000 units, 2700 units will be out of specification. Compare this to the case of $\pm 6\sigma$ specification, where $A = 0.999999998$, which gives 0.002 of one unit out of 1,000,000 will be out of specification. In the real world, there are normal shifts and drifts in the "mean," which take a toll on product quality. The shift in the mean by $\pm 1.5\sigma$ is the basis for the 6σ manufacturing method, which gives 3.4 ppm defects (32–35).

Throughput yield is a viable measure of quality and productivity. At the factory level, the question arises for a given number of parts per million, how many of a product could be expected to have zero, one, two, three, or four defects? The answer to this question defines the throughput yield of the factory. The Poisson distribution is used to calculate the yield.

Assume that the defects are randomly distributed throughout the units of a product. Let p be the occurrence probability of a defect, and let n be the sample size or the number of trials. The probability of getting r defects in the sample of size n is Y_r:

$$Y_r = \frac{(np)^r e^{-np}}{r!} \tag{5}$$

Expressing Y_r in terms of number of defects per unit (d/u) gives

$$Y_r = \frac{(\mathrm{d/u})^r e^{-(\mathrm{d/u})}}{r!} \tag{6}$$

The special case of getting zero defects $r = 0$, is called rolled yield,

$$Y_0 = e^{-(\mathrm{d/u})} \tag{7}$$

Rolled throughput yield is given in percent as

$$Y_0 \text{ in percent } (\%) = 100e^{-(\mathrm{d/u})} \tag{8}$$

Equation (8) shows that when d/u = 1, one defect per unit produced $Y_0 = 37\%$. This means that, on the average, 63% of the product will be rejected. The productivity in this case is very poor and is not affordable. Consider a product requiring 1200 parts and steps. Assume that the product is designed to accept a tolerance "twice" the normal variation of the pro-

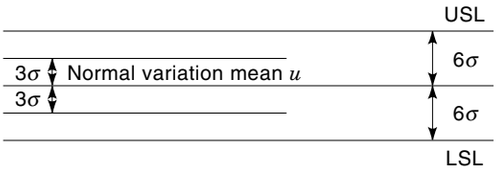

Figure 17. Capability index C is a function of the design specification width (tolerance) and the process variation width.

cesses, or $\pm 6\sigma$ with shift of $\pm 1.5\sigma$. This result in 3.4 ppm of parts or steps variation, and

$$\mathrm{d/u} = (1200)(0.0000034) = 0.0041 \text{ defects}$$

It follows that $Y_0 = 0.996$. This means that 996 units out of 1000 would go through the entire manufacturing process without a defect; this demonstrates a very effective manufacturing factory capable of producing affordable products.

MOEs at the Process Level

The previous example illustrates that at all design margins, product yield decreased with complexity (1,5,12,13,17). The complexity of a product is measured by the total number of parts and manufacturing steps required to make the product. To increase the yield of the factory, one must improve the quality of each component and reduce the process variation. Another way to improve yield is to increase the width of design specification. This influences the quality of the product as much as control of the process variation. Increasing the width of the design specifications requires robust designs. The throughput yield of the factory is dependent on the robustness of the design of the product and the process control of each manufacturing operation. It follows that there are two factors that control the throughput yield of the factory—process variation and width of the design specification. The two concepts are realized by making the robust design specification twice as wide as the process variation. Allowing for $\pm 1.5\sigma$ changes in the mean of process variation and using this concept, the manufacturing process is referred to as the six sigma (6σ) method.

The MOE of manufacturing is the capability index C_p, which is defined as (1,36,37)

$$C_\mathrm{p} = \frac{\text{Design specification width}}{\text{Process variation width}} \tag{9}$$

A process with normal variation is defined as a process with width of variation $\pm 3\sigma$ about the mean as shown in Fig. 17. Accordingly, the C_p of a product whose design specification width is twice the width of the normal process variation is

$$C_\mathrm{p} = \frac{|\mathrm{USL} - \mathrm{LSL}|}{6\sigma} = \frac{12\sigma}{6\sigma} = 2$$

where USL = Upper Specification Limit, LSL = Lower Specification Limit, and u = Mean.

When the process mean is shifted with respect to the design mean, the capability index is adjusted by a factor k, and it becomes C_pk where

$$C_\mathrm{pk} = C_\mathrm{p}(1 - k) \tag{10}$$

where

$$k = \frac{\text{Process shift}}{\frac{1}{2}(\text{Design specification width})} \tag{11}$$

The k factor for $\pm 6\sigma$ design specification width with a $\pm 1.5\sigma$ process shift is

$$k = \frac{2(1.5)\sigma}{12\sigma} = \frac{3}{12} = \frac{1}{4} \tag{12}$$

and from Eq. (9).

$$C_\mathrm{pk} = 2(1 - 0.25) = 1.5$$

The application of system engineering principles provided a tractable and comprehensive approach to the design of a CIM factory. System partitioning, integration, and architecture design are key steps in designing the factory. The approach considered the process of embedding software into hardware to establish the architecture of the factory. A client/server architecture was chosen because it provides the desired factory flexibility. The information and work flow of the factory are used to establish the architecture and select the factory software applications. The investigation illustrates the importance of software interfaces/middlewares in the system integration process. Two sets of measures—factory level metrics and process/component level metrics—are used to evaluate the effectiveness of the factory. Each set of measures deals with specific factory attributes. The investigation shows that the attributes of CIM in reducing the cycle time, recurring, and nonrecurring engineering costs are significant and will result in producing competitive products.

BIBLIOGRAPHY

1. A. R. Habayeb, System engineering of computer integrated manufacturing, *Naval Engineers J.,* **108** (6): 59–71, 1996.
2. R. Jaikuma, 200 years to CIM, *IEEE Spectrum,* **30** (9): 26–27, 1993.
3. D. Babcock, *Managing Engineering and Technology,* Englewood Cliffs, NJ: Prentice-Hall, 1991.
4. T. E. Bell, Bicycles on a personalized basis, *IEEE Spectrum,* **30** (9): 32–35, 1993.
5. A. R. Habayeb, *Systems Effectiveness,* London and New York: Pergamon Press, 1987.
6. A. R. Habayeb, System decomposition, partitioning and integration for microelectronics, *IEEE Trans. Syst. Sci. Cybern.,* 1968.
7. A. P. Sage (ed), *Systems Engineering Methodology and Applications,* New York: IEEE Press, 1977.
8. G. DeMicheli and R. Gupta, Hardware/software co-design, *Proc. IEEE,* **85**: 349–365, 1997.
9. S. Edwards et al., Design of embedded systems: Formal models, validation, and synthesis, *Proc. IEEE,* **85** (3): 366–390, 1997.
10. P. E. Green, *Computer Network Architectures and Protocols,* New York and London: Plenum Press, 1983.
11. W. Eckerson, Client/server architecture, *Network World,* 1995.

12. N. Engler, Riding the bleeding edge of distributed computing, *Open Computing,* 1995.
13. J. Mullich, The riddle in the middle, *Open Computing,* August 1995.
14. P. Bernstein, Middleware: A model for distributed system services, *Commun. ACM,* **39** (2): 86–98, 1996.
15. T. Williams, Graphical simulation tool for complex systems, *Electron. Design,* **44** (26): 54–55, December 1996.
16. T. G. Lewis, *CASE: Computer-Aided Software Engineering,* New York: VanNostrand Reinhold, 1991.
17. B. T. Harrison, Client/server development, Which way will we go?, *DEC Professional,* April 1994.
18. Open System Foundation DCE, *Administration Guide, Core Components,* Englewood Cliffs, NJ: Prentice-Hall, 1993.
19. *Application Portability Profile (APP), Open System Environment, Version 1.0,* NIST Publication 500-127, April 1991/update June 1994.
20. A. Cini, *Networking for Everyone,* Horsham, PA: DEC Professional Cardinal Business Media, 1994.
21. I. Jacobson, *Object Oriented Software Engineering,* Reading, MA: Addison-Wesley, 1992.
22. P. VandenHamer and K. LePoeter, Managing design data: The five dimensions of CAD frameworks, *Proc. IEEE,* **84**: 42–56, 1996.
23. D. T. Dewire, *Clearing up the Middleware Muddle,* Client Server Computing, Danvers, MA: Sentry Publishing Co., 1995.
24. A. Radding, Middleware works magic with apps, *Information Week,* June 17, 1996.
25. J. Rumbaugh, *Object Oriented Modeling and Design,* Englewood Cliffs, NJ: Prentice Hall, 1991.
26. S. Weissman, Your development tools, *Network World,* April 1995.
27. J. M. Willis, *TP Software Development for Open VMS,* Horsham, PA: Cardinal Business Media, Inc., 1994.
28. R. Whiting, *Bridging the CORBA OLE Gap,* Danvers, MA: Client/Server Computing Sentry Publishing Co., 1996.
29. R. S. Raji, Smart networks for control, *IEEE Spectrum,* **31** (6): 49–55, June 1994.
30. D. Settle, Scalable parallel processing, *Comput. Technol. Rev.,* Spring 1995.
31. D. J. Kaplan, *Processing Graph Method Specification Version 1.0,* Washington, DC: The Naval Research Laboratory, 1987.
32. A. A. Afifi and S. P. Azen, *Statistical Analysis: A Computer Oriented Approach,* 2nd ed., Reading, MA: Academic Press, 1979.
33. E. Grant and R. S. Leavenworth, *Statistical Quality Control,* 5th ed., New York: McGraw-Hill, 1980.
34. D. H. Evans, Statistical tolerancing: The state of the art, Part I: Background and Part II: Shifts and drifts, *J. Quality Technol.,* 1972.
35. B. Smith, Making war on defects, *IEEE Spectrum,* **30** (9): 43–43, September 1993.
36. V. E. Kane, Process capability indices, *J. Quality Technol.,* 1986.
37. A. Bowker and G. Liberman, *Engineering Statistics,* Englewood Cliffs, NJ: Prentice-Hall, 1972.

A. R. HABAYEB
Naval Air Systems Command

COMPUTER INTERFACE.

See CAMAC.