JAVA, JAVASCRIPT, AND HOT JAVA

Java, an object-oriented programming language that was introduced by Sun Microsystems in 1996, has quickly gained a unique and dominant status in software engineering. Examples of Java programs range from very small programs that fit on a credit card (known as Java card) to large-scale corporate applications that access multiple corporate databases and applications. In a very short time, the interest in Java has spawned an unprecedented bevy of new development tools and extensions of existing ones. This "Java family" consists of Java development aids (e.g., Java Development Kit, PowerJ, Visual Cafe), scripting languages (e.g., Javascript), and Java-enabled browsers (e.g., Hot Java, Microsoft Internet Explorer, Netscape Navigator). Java is playing a unique role in the World Wide Web (WWW) to support a very diverse array of Web-based applications, such as Web-based purchasing systems. This Java family is also becoming the foundation for distributed object applications across the Internet that support code mobility and portability. This article presents a technical overview of the key players of the Java family (i.e., Java, Javascript, and Hot Java) and attempts to answer the following questions:

- How does the WWW provide the environment in which Java operates (section entitled "World Wide Web—The Java Environment")?
- What are the key features of Java, what are Java applets, and how do they differ from Java applications (section entitled "Java and Java Applets")?
- What are Java programming features, what is a Java Virtual Machine (JVM), and what are the various Java development tools (section entitled "Java Programming and Development Environments")?
- What is Hot Java, how does it compare/contrast with other Java enabled browsers, and how can Java be supported uniformly across multiple browsers and platforms by using tools such as the Sun Java Activator (section entitled "Hot Java and Java-Enabled Browsers")?
- What is Javascript, how does it compare/contrast with Java, and how does it relate to other scripting languages (section entitled "JavaScript")?
- How can Java be used to develop distributed object applications with distributed object middleware such as CORBA (section entitled "Combining Java with Distributed Objects—Java and CORBA")?

WORLD WIDE WEB—THE JAVA ENVIRONMENT

Technically speaking, WWW is a collection of software that operates on top of TCP/IP (Transmission Control Protocol/Internet Protocol) networks (i.e., the Internet) as shown in Fig. 1. Java, as we will see, has become an integral component of WWW due to its interplays with the following core WWW technologies (see Fig. 2):

- Web servers
- Web browsers
- Uniform Resource Locator (URL)
- Hypertext Transfer Protocol (HTTP)
- Hypertext Markup Language (HTML)
- Web navigation and search tools
- Gateways to non-Web resources

Let us briefly review these components before discussing Java details.

Web sites provide the content that is accessed by Web users. Conceptually, a Web site is a catalog of information for each content provider over the Web. In reality, a Web site consists of three types of components: a Web server (a program), content files ("Web pages"), and/or gateways (programs that access non-Web content). A *Web server* is a program (technically a server process) that receives calls from Web clients and retrieves Web pages and/or receives information from gateways. Once again, a Web user views a Web site as a collection of files on a computer, usually a UNIX or Microsoft Windows NT machine. The large number of Web sites containing a wide range of information that can be navigated and searched transparently by Web users is the main strength of the WWW. Figure 2 shows two Web sites: one for a store (www.store.com) and the other for a computer science department for a university (cs.ud.edu).

Web browsers are the clients that typically use graphical user interfaces to wander through the Web sites. The first GUI browser, Mosaic, was developed at the National Center for Supercomputer Applications at the University of Illinois. At present, Web browsers are commercially available from Netscape, Microsoft, and many other software/freeware providers. These Web browsers provide an intuitive view of information where *hyperlinks* (links to other text information) appear as underlined items or highlighted text/images. If a user points and clicks on the highlighted text/images, then the Web browser uses HTTP to fetch the requested document from an appropriate Web site. Web browsers are designed to display information prepared in a markup language, known as HTML. We will discuss HTTP and HTML later. Three different browsers are shown in Fig. 2. Even though these are different browsers residing on different machines, they all use the same protocol (HTTP) to communicate with the Web servers (HTTP compliance is a basic requirement for Web browsers).

Browsers used to be relatively dumb (i.e., they just passed user requests to Web servers and displayed the results). However, this has changed because of *Java,* a programming language developed by Sun Microsystems. Java programs, known as *Java applets,* can run on Java-compatible browsers. This is creating many interesting possibilities where Java applets are downloaded to the Java-enabled browsers where they run producing graphs/charts, invoking multimedia applications, and accessing remote databases. We will discuss Java and Java applets in a later section.

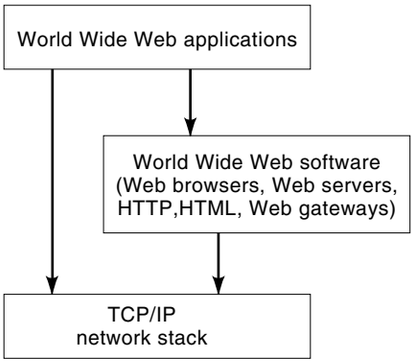

Figure 1. Technical view of World Wide Web.

Uniform Resource Locator (URL) is the basis for locating resources in WWW. A URL consists of a string of characters that uniquely identifies a resource. A user can connect to resources by typing the URL in a browser window or by clicking on a hyperlink that implicitly invokes a URL. Perhaps the best way to explain URLs is through an example. Let us look at the URL "http://cs.ud.edu/faculty.html" shown in Fig. 2. The "http" in the URL tells the server that an HTTP request is being initiated (if http is substituted with ftp, then an FTP session is initiated). The "cs.ud.edu" is the name of the machine running the Web server (this is actually the domain name used by the Internet to locate machines on the Internet). The "/faculty.html" is the name of a file on the machine cs.um.edu. The "html" suffix indicates that this is an HTML file. When this URL is clicked or typed, the browser initiates a connection to "cs.um.edu" machine and initiates a "get" request for the "faculty.html" file. Depending on the type of browser being used, these requests can be seen flying around in an appropriate window spot. Eventually, this document is fetched, transferred to and displayed at the Web browser. Information can be accessed through the Web by issuing a URL (directly or indirectly). As we will see later, the Web search tools basically return a list of URLs in response to a search query.

Hypertext Markup Language (HTML) is an easy-to-use language that tags the text files for display at Web browsers. HTML also helps in creation of *hypertext links,* usually called hyperlinks, which provide a path from one document to another. The hyperlinks contain URLs for the needed resources. The main purpose of HTML is to allow users to flip through Web documents in a manner similar to flipping through a book, magazine, or catalog. The Web site "cs.ud.edu" shown in Fig. 2 contains two HTML documents: "faculty.html" and "courses.html." HTML documents can embed text, images, audio, and video.

Hypertext Transfer Protocol (HTTP) is an application-level protocol designed for Web browsers. It is intended for exchange of hypermedia (or "rich") content between clients and servers. HTTP uses an extremely simple request/response model that establishes connection with the Web server specified in the URL, retrieves the needed document, and closes the connection. Once the document has been transferred to a Web browser, then the browser takes over. Typically, every time the user clicks on a hyperlink, an HTTP session is being initiated to transfer the needed information to the user's browser. The Web users shown in Fig. 2 access the information stored in the two servers by using the HTTP protocol.

Web navigation and search services are used to search and surf the vast resources available in cyberspace. The term *cyberspace,* as stated previously, was first introduced through a science fiction book by Gibson (1) but currently refers to the computer-mediated experiences for visualization, communication, and browsing. The general search paradigm used is that each search service contains an index of information available on Web sites. This index is almost always created and updated by *"spiders"* that crawl around the Web sites chasing hyperlinks for different pieces of information. Search engines support keyword and/or subject-oriented browsing through the index. Result of this browsing is a "hit list" of hyperlinks (URLs) that the user can click on to access the needed information. For example, the Web users in Fig. 2 can issue a keyword search, say by using a search service for the stores in Chicago. This will return a hit list of potential shoe stores that are Web content providers. The user then points and clicks till the store of choice is found. Many search services are currently available on the Web. Examples are Yahoo, Lycos, and Alta Vista. At present, many of these tools are being integrated with Web pages and Web browsers. For example, the Netscape browser automatically invokes the Netscape home page that displays search tools that can be invoked by just pointing and clicking. It is beyond the scope of this book to describe the various Web navigation and search tools. Many books about the Internet describe these search tools quite well.

Gateways to non-Web resources are used to bridge the gap between Web browsers and corporate applications and databases. Web gateways are used for accessing information from heterogeneous data sources (e.g., relational databases, indexed files, and legacy information sources) and can be used to handle almost anything that is not designed with an HTML interface. The basic issue is that the Web browsers can display HTML information. These gateways are used to access non-HTML information and convert it to HTML format for display at a Web browser. The gateway programs typically

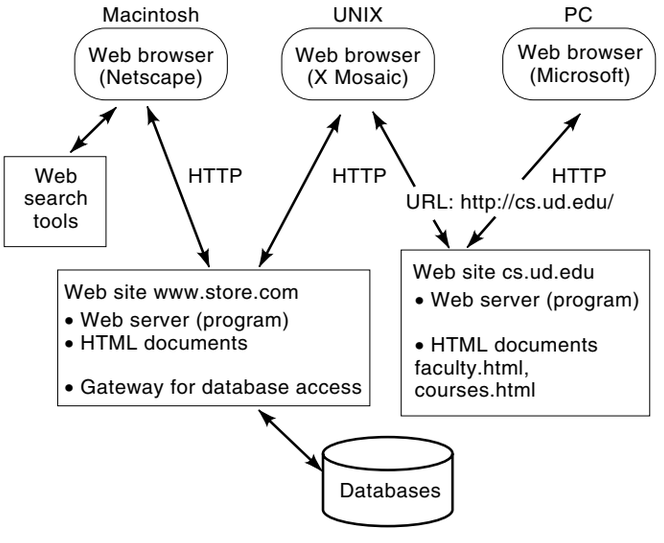

Figure 2. Conceptual view of World Wide Web.

run on Web sites and are invoked by the Web servers. At present, the common gateway interface (CGI) and its variants are used frequently.

JAVA AND JAVA APPLETS

Java Overview

Java is an object-oriented programming language that is playing a unique role in the WWW. The Java programming language and environment was introduced by Sun Microsystems initially to develop advanced software for consumer electronics. Initially, Sun intended to use C++ for these devices, which are small, reliable, portable, distributed, real-time embedded systems. It was found that the problems were best solved by introducing a new language that was similar to C++ but drew heavily from other object-oriented languages such as Eiffel and Smalltalk. The language, initially known as Oak, is now known as Java.

Why is Java so popular? The key is in supporting user interactions with Web pages that use Java. Simply stated, small Java programs, called *Java applets,* can be embedded in Web pages (these are called *Java-powered pages*). Java-powered Web pages can be downloaded to the Web client side and make the Web browsers a powerful user tool. Before Java, Web browsers were relatively dumb (i.e., most functionality resided in Web servers, not in Web browsers). Java changed all that because Java applets can run on Java-enabled browsers. When users access these pages, they along with the Java applets, are downloaded to the Web browser. The Java applets run on the Web client side, thus making the browser an intelligent processor. There are several implications to this:

- Java applets make Web applications truly client/server because the Java code can run business logic on the Web client site (i.e., the Web browser houses the first tier).
- Java applets exemplify "mobile code" that is developed at one site and is migrated to another site on demand. This introduces several security issues but also creates many interesting research opportunities.
- Back-end resources (databases and applications) can be accessed directly from the browser instead of invoking a gateway program that resides on the Web server site (security considerations may require a "proxy" server on the Web server site). The Java program can ask the user to issue a request and then send this request to back-end systems. A standard called Java Database Connectivity (JDBC) has been developed to allow Java programs to issue calls to relational databases.
- The Web screen content and layout can be changed dynamically based on the user type. A Java program can determine the user type and modify the screen layout. For example, different advertisements can be shown and highlighted to the user depending on the user characteristics (e.g., age, job type, education level, credit history, salary level).
- Graphs and charts can be produced dynamically at the user's browser instead of fetching predefined graphs and images from the Web server (transferring images takes a very long time over the Internet).
- The user can run animations, invoke business transactions, and run spreadsheets at the user's browser site. In essence, almost any application can be run on the user's Web browser that can interact with the user, display graphics, and interact with back-end databases and applications.

Java Applets versus Java Applications

What is the difference between a Java application and a Java applet? Basically, a Java application is a complete, stand-alone application that is written in the Java language. Java applets, on the other hand, are not stand-alone applications and they run as part of a Java-enabled browser. From a programming point of view, a Java application is Java code ("Java Class") that has the main ( ) method. The Java interpreter looks for main ( ) and executes it. Java applets do not execute main ( ). Instead, Java applets contain methods that are invoked by the Java-enabled browsers.

A Java applet contains methods (subroutines) to initialize itself, draw itself, respond to clicks, and so on. These methods are invoked by the Java-enabled browser. How does a browser know to download Java applets. It is quite simple. A Java-powered HTML page contains a tag (the ⟨`applet`⟩ tag) that indicates the location of a Java applet. When the browser encounters this tag, it downloads it and runs it. See the section entitled "Downloading and Running Java Applets."

Java applets are small enough so that they can be embedded in Web pages but large enough to do something useful. Java applets are transferred to the Web browser along with everything else embedded in the Web page (e.g., text, images, video clips). Once transferred to the Web client, they execute on the client side and thus do not suffer from the issues of network traffic between the Web client and Web server. Because these applets run on the client machine, the user sees the user a much more natural and efficient execution.

Due to the popularity of Java applets, many plug-and-play Java applets are already available. Once built, the Java applets can run on many different machines. The Java code is first compiled into byte codes (byte codes are machine instructions that are machine-independent). The byte code of the applet is loaded into the browser where it runs efficiently on different machines by using a runtime interpreter. Due to the appeal of Java applet style programming, other programming languages such as C++ and COBOL have started producing byte codes that can be invoked by Web browsers (the browsers do not know how the code was created).

Java applets have access to a wide range of libraries that allow Java applets to perform many operations such as graphics, image downloading, playing audio files, and user interface creation (i.e., buttons, scrollbars, windows, etc.). These libraries are included as part of the Java applet application program interface (API). This API is supported by all Java-compatible browsers. It is expected that these libraries will grow

with time, thus making Java applets even more powerful and diversified.

Key Java Features

Java has emerged as a very popular language for developing Web applications. According to Sun, "Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language." The following paragraphs discuss these features of Java. The following discussion is an abbreviated version of the Java white paper that can be obtained from Sun's web page (http://java.sun.com). Although Java is very popular at present, it is presenting some security concerns (see the section entitled "Java Security Concerns").

- *Simplicity.* Java was designed to be similar to C++ in order to make the system more comprehensible to current practitioners. Java omits many features of C++ such as operator overloading (although the Java language does have method overloading), multiple inheritance, and extensive automatic coercions. The auto garbage collection was added, thereby simplifying the task of Java programming but making the system somewhat more complicated. A good example of a common source of complexity in many C and C++ applications is storage management: the allocation and freeing of memory. By virtue of having automatic garbage collection the Java language makes the programming task easier and also cuts down on bugs. Java is designed so that it can run stand-alone in small machines. The size of the basic interpreter and class support is about 40 kbytes; adding the basic standard libraries and thread support adds an additional 175 K. [The Java Development Kit (JDK) system with all the documentation is getting larger—it is about 70 MB].
- *Object-Orientation.* The object-oriented facilities of Java are essentially those of C++, with extensions from Objective C for more dynamic method resolution.
- *Robust.* Java puts a lot of emphasis on early checking for possible problems, later dynamic (runtime) checking, and eliminating error prone situations. Java requires declarations and does not support C-style implicit declarations. The single biggest difference between Java and C/C++ is that Java does not allow pointer arithmetic. Java has arrays that allow subscript checking to be performed. In addition, Java does not allow an arbitrary integer to be converted into a pointer.
- *Distributed.* The main power of Java is that Java applications can open and access objects over the Internet via URLs in a manner similar to accessing a local file system. Java has an extensive library of routines for coping easily with TCP/IP protocols like HTTP and ftp.
- *Architecture Neutral.* Java was designed to support applications on networks. The Java compiler generates an architecture neutral object file format that is executable on many processors, given the presence of the Java runtime system. The Java compiler generates byte-code instructions that are independent of computer architecture. Byte codes are designed to be easy to interpret on any machine and can be easily translated into native machine code on the fly.
- *Portable.* Java specifies the sizes of the primitive data types and the behavior of arithmetic on them. For example, "int" always means a signed two's complement 32-bit integer, and "float" always means a 32-bit IEEE 754 floating point number. The libraries that are a part of the system define portable interfaces. The Java system itself is also portable. The compiler is written in Java, and the runtime is written in ANSI C with a clean portability boundary. The portability boundary is essentially POSIX.
- *Interpreted.* The Java interpreter can execute Java byte codes directly on any machine to which the interpreter has been ported. And since linking is a more incremental and lightweight process, the development process can be much more rapid and exploratory.
- *High Performance.* In some cases, the performance of interpreted byte codes is not adequate. Java byte codes can be translated on the fly (at runtime) into machine code for the particular CPU the application is running on. The byte-code format was designed with generating machine codes in mind, so the actual process of generating machine code is generally simple. According to Sun, the performance of byte codes converted to machine code will eventually become indistinguishable from native C or C++.
- *Multithreaded.* Multithreading is important for performance, but writing multithreaded programs is more difficult than writing in the conventional single-threaded programs. Java has a set of synchronization primitives that are based on the widely used monitor and condition variable paradigm.
- *Dynamic.* Java was designed to adapt to an evolving environment. It makes the interconnections between modules later. Java understands interfaces—a concept that is used heavily in distributed systems through Interface Definition Languages (IDLs). An interface is simply a specification of a set of methods that an object responds to. Interfaces make it possible to use objects in a dynamic distributed environment (we will talk about this when we discuss CORBA).

The best source for additional information about Java is the Sun home page (*http://Java.sun.com*). From this page, you can find a Java white paper that gives justification of Java, an 80-page technical document on Java and Hot Java, Java applets, and additional detailed documentation. The book *Hooked on Java* (2) gave one of the earliest introduction to Java. An interesting analysis of Java can be found in Ref. 3. At present, more than 100 books are available on different aspects of Java.

Downloading and Running Java Applets

The Java applets are downloaded and executed on the Web browser by using the following steps:

- User selects an HTML page.
- Browser locates the page and starts loading it.
- While loading, it starts to format text.
- It loads graphics if indicated by IMG tags in HTML.

- Java applets are indicated by an ⟨applet⟩ tag. For example, the tag indicates a Java applet called "myapplet.class" that is run in a window size of 110 by 150:

  ⟨APPLET CODE=myapplet.class WIDTH=110 HEIGHT=150⟩

  ⟨/APPLET⟩
- The applet code is assumed to be on the same site where the HTML page is.
- Browser loads the indicated class and other needed classes.
- Java-enabled browsers have a virtual machine and keep local classes that may be used by the applets.
- After the applet has been loaded, the browser asks it to initialize itself by invoking the init ( ) method and draw a display area that is used for input/output.

Distributed Applications with Java. A user who needs to write a Java application where a Java applet on the user's Web browser invokes another Java applet on another machine has the following choices:

- User-written low-level code (e.g., TCP sockets) to invoke the remote Java code.
- Using, if possible, distributed object middleware such as CORBA.

The first choice is not very attractive (users must write their own middleware). The second choice can be pursued through:

- CORBA calls
- DCOM calls
- Sun's Remote Method Invocation (RMI)

See the section entitled "Combining Java with Distributed Objects—Java and CORBA" for additional details.

Java Security Concerns

Several security flaws in Java are currently being discovered and addressed. The basic premise of the security concerns is that Java applets are essentially foreign applications that are brought into a local environment and executed on a local browser site. Such programs can contaminate the environment.

Java designers have taken reasonable precautions about Java security by introducing a Java verifier to make sure that the byte code was generated by a valid compiler before running it (Java compilers restrict pointers and typecodes to minimize security risks). However, several security flaws in Java are currently being discovered and addressed. Java applets are foreign applications that are brought into a local environment and executed on a local browser site. This opens the floodgate to unscrupulous code being brought in from other sites. A "social/administrative" remedy to this problem is to make sure that Java applets are downloaded from trusted sites only (e.g., corporate Web servers within the firewalls).

The examples of how Java programs can contaminate the environment abound. For example, David Hopwood at Oxford University found that Java applets can load malicious class files and libraries onto a user's system. Many "hostile applets," such as the following, have been documented:

- A noisy bear who refuses to be quiet
- A barking browser
- Popping up numerous unsolicited applet windows
- Forging e-mail
- Obtaining a user ID

A great deal of work is needed to resolve the Java security issues. A research group at Princeton University, headed by Edward Felten, is investigating Java security problems. An article by this group (4) lists a compendium of hostile actions that a Java applet can perform.

There are three different approaches to security for Java applets.

- Trusted servers
- Sandboxes
- Digital signatures

Trusting the server is a feasible "social/administrative" choice within the secure corporate intranet (files are downloaded regularly from corporate file servers). The corporate servers can be trusted not to deliver components that contain viruses or damage the system on which they are loaded and executed.

Sandboxing constrains the components themselves, making it impossible for them to execute unwanted functions. It is very much like putting a child in a sandbox—we put a sandbox around Java applets to that they do not hurt others/themselves. Sandboxing can guarantee security by dictating that the downloaded components are obligated to play only in their own sandbox. The disadvantage of this approach is that sandboxed components are prohibited from doing things that can sometimes be useful, like writing to a file on the client machine's local disk.

Digitally signing each downloaded component is an attractive approach. The digital signature can be checked by the browser that receives the component. If it is correct, the browser can be certain that the component was created by a specific trusted entity and that it has not been modified. These are potential problems with digital signatures also. For example, we do not know whether or not the signing person will attack our system. Basically digital signatures allow us to decide what the applet should be allowed to do.

JAVA PROGRAMMING AND DEVELOPMENT ENVIRONMENTS

Java Programming Details and Examples

Getting Started. An example of the classic "Hello World" application is shown in Fig. 3. The example is an application,

```
/**
* This class prints out the phrase "Hello, World!"
* @author The author's name
* @version 1.0 */
public class HelloWorld {
/* A simple Java application */
public static void main (String arg []) {
  System.out..println("Hello, World!");
// comment: System.out..println = output
    }
}
```

Figure 3. HelloWorld.java—The Hello World application in Java.

```
· Put the source statements in HelloWorld.java
· Compile by typing: javac HelloWorld.java
· This creates a file: HelloWorld.class (in byte-code)
· To run, type: java HelloWorld
```

Figure 4. Running the HelloWorld application with the JDK.

not an applet. It writes the "Hello, World!" string to the system console. This example shows many of the basic features of the language:

- The file name, without the ".java" extension, is the same as the public class name, including capitalization.
- Java has three forms of comments:

  /** Comments between slashes and asterisks, begun with a double asterisk. These comments may be read by the javadoc utility for automatic incorporation into documentation, including specific values of the form '@keyword'. */

  /* Comments between slashes and asterisks. */

  // Comments after a double slash, continuing to the end-of-line.
- All Java code is located within the class definition.
- The declaration of the "main" method, which is "public" and "static," or class level, which returns "void" (i.e., nothing) and which takes an array of string arguments in the parameter "args" is shown.
- The invocation of the "println" method of the "out" PrintStream attribute of the System class is used for printing.

The most basic way to run a Java application is to use the Java Development Kit (JDK), which is widely available for many platforms, usually from the platform vendor; for example, Sun Microsystems markets a JDK for Windows and Solaris platforms, Hewlett-Packard markets a JDK for HP platforms, and, similarly, IBM markets a JDK for IBM platforms. Microsoft markets a Java SDK that is similar to a JDK for Windows platforms. Figure 4 shows how to run the HelloWorld application using the JDK.

An Applet. Figure 5 shows the code for a simple Java applet that writes "Hello, World!" to the browser's Java console. It points out the basic differences between applications and applets.

- The java.applet libraries are imported. The HelloWorld application did not use any Java classes other than System, which is always available. In contrast, the HelloWorld applet uses the applet libraries. Practical examples usually import several libraries.
- An applet "extends Applet," meaning that it is a subclass of the Applet class, usually the version shipped as java.applet.Applet.
- An application has a "main" method; an applet does not.
- An applet overrides one or more of the methods of the Applet class, especially:

  init( )—how to set up.

  start( )—begins the operation of the applet.

  stop( )—ends or suspends the operation of the applet.

  paint( )—draws the visible representation of the applet.

A More Significant Applet. Figure 6 shows an applet with a paint ( ) method, which operates on the Graphics object that it takes in as a parameter.

Java applets are not stand-alone applications and they run as part of a Java-enabled browser. A Java applet may contain methods (subroutines) to initialize itself, draw itself, respond to clicks, and so on. These methods are invoked by the Java-enabled browser. A Java-powered HTML page contains a tag (the ⟨applet⟩ tag) that indicates the location of a Java applet. When the browser encounters this tag, it downloads it and runs it. Java applets are indicated by an APPLET tag. For example, the following tag indicates the Java applet called "LineApplet.class" that is run in a window size of 110 by 100:

Observe that the HTML code in Fig. 7 defines the size of the applet, and also observe that the applet paint ( ) code in Fig. 6 works with the space that it is given. The HTML controls the size of the applet display: if the applet tries to create a larger display than the HTML allocated, the display is truncated.

The applet code is assumed to be on the same site where the HTML page resides. The Java-enabled browser loads the indicated class and other needed classes (Java-enabled browsers also keep local classes that may be used by the applets). After the applet has been loaded, the browser asks it to initialize itself [the init ( ) method] and draw a display area [the paint ( ) method] that is used for input/output. Java applets have access to a wide range of libraries that allow Java applets to perform many operations such as graphics,

```
import java.applet.*; // include applet classes (i.e., imports necessary classes)
/**
* This applet prints out the phrase "Hello, World!"
* @author The author's name
* @version 1.0 */
public class HelloWorld extends Applet {
/* A simple Java applet */
public void start () {
  System.out.println ("Hello, World!");
// comment: System.out..println = output
  }
}
```

Figure 5. HelloWorld.java—Hello World as an applet.

```
import java.awt.*;     // include the java tool classes and
import java.applet.*; // include applet classes (i.e., imports necessary classes)
public class LineApplet extends Applet { // LineApplet is a subset of Applet
 public void paint (Graphics g) { // paint method is overridden
 Dimension r = size();  // find out how big the applet window is
 g.setColor (Color.green);  // set color to green
 g.drawLine(0,0, r.width, r.height);  // draw the line from corner to corner
 }
}
```

Figure 6. LineApplet.java: A simple Java applet that draws a line.

image downloading, playing audio files, and user interface creation (i.e., buttons, scrollbars, windows, etc.).

An Applet/Application Combination. The definitions of Applets and Applications are not exclusive: a single class definition can be both an applet and an application. To be both applet and application, a class is coded as an applet, but it is also provided with a main ( ) method that allocates a browserlike environment and then invokes the methods of the applet. This technique is sometimes used to create applets that can be unit-tested from the command line, without the browser. The LineApplet seen earlier is turned into an example applet/application combination in Fig. 8.

More Complex Applets. The applets discussed above have been simple and introductory. Considerably more complex applets with richer functionality are possible. An important way in which additional functionality is added is via database access.

By using Java applets, access to remote applications and databases can be invoked directly from the browser. The Java applet can ask the user to issue a query and then send this query to a remote application or database (Fig. 9). This is especially interesting for database gateways where the database gateway functionality runs on the client side. A standard called Java Database Connectivity (JDBC) has been developed to allow Java programs to issue calls to relational databases.

A user who wishes to write a Java application where a Java applet on the user's Web browser invokes another Java applet on another machine can use distributed object middleware such as CORBA. Sun has developed a special feature of Java that allows Java applets to talk to each other across machines. This feature, known as Remote Method Invocation (RMI) allows Java applets to communicate with each other over the Internet. In addition, Sun has added a capability that will allow Java applets to work across a firewall.

Java is quite popular at present, and its popularity keeps growing steadily. However, some security concerns have been raised mainly because Java is a downloadable application (i.e., it is downloaded from a Web server site). Java is not alone in this area. ActiveX also supports downloadable components. Different approaches to deal with the security of downloadable software such as Java and ActiveX Controls are being pursued at present.

```
<APPLET CODE=LineApplet.class WIDTH=110 HEIGHT=100>
<APPLET>
```

Figure 7. The HTML to display the LineApplet applet.

Handling Security of Downloadable Software: Java and ActiveX Controls Issues. Java applets are downloaded from the Web server and run at the Web browser site. This raises several concerns about Java. There are three different approaches to security for Java applets.

- Trusted servers
- Sandboxes
- Digital signatures

We have discussed these issues previously.

Java applets currently support the first two methods. Digital signatures on downloadable Java components is an area of active work (most Java enabled browsers support this feature at present).

A related issue is how the downloaded ActiveX controls can be made secure. ActiveX downloading currently supports only the first of these three methods, namely, downloading from a trusted server. This is because they are shipped to the client as binaries, and thus it is hard to sandbox them. Naturally, digital signatures is an interesting area of work for ActiveX controls. (Active controls can be digitally signed at present also.)

Digital signatures offer the most general solution to the problem, one that would work well with Java as well as ActiveX controls.

The Java Virtual Machine

The Java Virtual Machine (JVM) is an essential component of the Java environment. The JVM is a specification that defines whether and how Java byte-code class files should be executed (5). The specification makes no mention as to how this will be accomplished, and it may be done via interpretation, via compilation into binary code, or via hardware—the proposed "Java chip." The JVM specification provides many of Java's features, notably its portability and security.

Java byte code is run through a "JVM implementation," which is often loosely referred to as a "JVM." First the byte code is validated, to ensure that it does not try to do anything illegal.

If the byte code is valid, then it is executed. The specification of byte-code execution is machine-independent, so that the same behavior can be produced on any given machine by a JVM implementation that is specific to the machine. Although Java byte code is portable, a JVM implementation is not portable. A JVM implementation is machine-specific.

The JVM is not the first virtual machine. Other VMs have preceded it, and others continue to be developed. The JVM

```
import java.awt.*;
import java.applet.Applet;
public class TestableLineApplet extends Applet {
 public static void main (String args[]) {
  TestableLineApplet t=new TestableLineApplet();
  Frame f=new Frame("Testable LineApplet Test");
  f.resize(250,250);
  f.add(t);
  f.show();
  t.init();
  t.start();
 }
  public void paint (Graphics g) { // paint method is overridden
  Dimension r=size();  // find out how big the applet window is
  g.setColor(Color.green);  // set color to green
  g.drawLine(0,0, r.width, r.height);  // draw the line from corner to corner
  }
```

Figure 8. An Applet/Application combination.

Specification itself notes "the best-known virtual machine may be the P-Code machine of UCSD Pascal."

Java is not the only source of byte codes to feed the JVM. Compilers have been written to generate Java byte codes from other languages (e.g., Ada). However, Java source code provides a straightforward path to the generation of Java byte codes.

Differences among JVM Implementations. The JVM Specification specifies what a JVM implementation should do, not how it should do it. Thus, there is a level of flexibility in how a JVM goes about executing Java byte-codes. One of the degrees of freedom is whether to execute the byte codes via interpretation, via compilation into machine code and execution of the resulting machine code, or via hardware, by creating a machine whose instructions correspond to the byte-code operands. This area is a source of distinction among competing JVM implementations, as different vendors try to create implementations that execute byte codes quickly. Various techniques can be tried either separately or in combination. Among the popular techniques are:

- *Optimized Interpretation.* Executing the byte codes in an interpreter, but separately analyzing the byte codes for improvements that can be made—for example, in memory management or execution order.

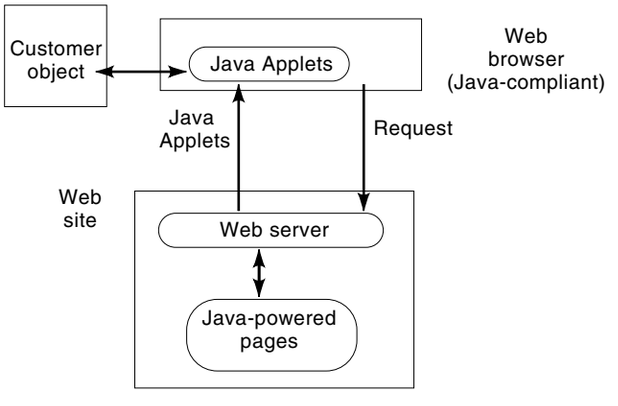

Figure 9. Java-based application.

- *Just-in-Time (JIT) Compilation.* Compiling the byte codes before and during execution. In contrast to compiling the entire applet or application before starting to execute them, this technique involves compiling and executing simultaneously, and the compiler is only a step ahead of the execution of the resulting machine code. The most obvious improvement caused by JIT compilation is when certain parts of the byte-code program are executed repeatedly. Because those sequences only have to be compiled once, the improvement in efficiency over interpretation can be significant.

Another degree of freedom provided by the specification is in what to optimize. While the most heavily publicized JVM implementations optimize for time, other implementations may optimize for other considerations; for example, a JVM implementation intended for use in embedded devices might optimize for efficient memory usage.

Java Development Tools and Environments

The JDK contains a compiler, a run-time environment, and some basic utilities. A Java developer needs much more, however, and that is where the Integrated Development Environment (IDE) comes in.

An IDE minimally combines the JDK with a source code editor and a debugger. Additionally, the IDE may provide such features as color syntax display, code formatting, templates, a graphical class library browser, components, and assistance in building the results.

The IDE is not an original feature of the Java programmer's world. Before Java was created, IDEs existed for other languages, notably C++. Examples of JAVA IDEs are Microsoft visual J^{++}, Symantec Visual Cafe, Powersoft's PowerJ, and Sun's Java Workshop.

HOT JAVA AND JAVA-ENABLED BROWSERS

Overview

Hot Java is a browser, introduced by Sun, that can run Java-powered pages; that is, it is Java-enabled. A special Java-enabled browser is needed that can run Java applets. Hot Java is such a browser. At the time of this writing, Hot Java is available for Windows 95, Windows NT, and Solaris 2.x platforms.

Due to the popularity of Java, many other browsers are also becoming Java-enabled. For example, Netscape Navigator as well as the Internet Explorer are Java-enabled. In reality, most commonly used browsers are Java-enabled at present.

If a Java-powered page is downloaded to a browser that is not Java-enabled, then nothing happens. Hot Jjava can be downloaded from http://Java.sun.com/products/hotJava.

Java-Enabled Web Browsers

Web browsers are the end-user interface to the Web servers. These browsers, also known as Web clients, typically reside on PCs, Macs, and UNIX workstations. From an end user's point of view, the browsers give a graphical user interface (GUI) and easy-to-use view of the Internet and provide pull down/pop up menus and buttons for accessing remote servers, scrolling through documents, printing results, downloading code, saving retrieved documents on a local disk, performing searches, and surfing the net. Many browsers have been introduced since 1990 and are currently in use. Examples are the Netscape Navigator, Microsoft, Internet Explorer, HotJava, NCSA X-Mosaic, NCSA Mosaic for Windows, Spyglass, Air Mosaic, and Win-Tapestry.

Web browsers are designed to display information in HTML format and communicate with the Web servers through HTTP. As a matter of fact, users can develop their own browser if they provide the following two capabilities:

- HTML compliance; that is, display information on the screen as specified by HTML tags.
- HTTP compliance; that is, generate HTTP commands to connect to the Web server, initiate needed operations whenever a user clicks on a hyperlink, and receive/interpret the responses.

Many popular browsers, such as the Netscape Navigator and the Internet Explorer, run on multiple platforms (PCs, Macs, UNIX). This is one of the many reasons for the popularity of WWW in the corporate world. While in the past a library system or a customer information system could have been developed by using a specially designed user interface, it seems much more natural for organizations today to use Web browsers for user interfaces. By using the Web browsers, users residing on different machines can use the same browser or a different browser to interact with the corporate systems. The same browser can also allow the users to use Web for document searches. Thus Web browsers have the potential of becoming the only user interface for all information. This makes WWW unique in that it makes hypermedia a key enabler of business as well as nonbusiness information that is becoming available through the Internet and Intranets.

Let us look at the Java-enabled Web browsers in some detail. As indicated previously, Java applets are indicated by an APPLET tag in HTML documents. For example, the following tag indicates a Java applet called "applet1.class," which is run in a window size of 100 by 150:

```
⟨APPLET CODE=myapplet.class WIDTH=100
  HEIGHT=150⟩
```

We have discussed how the applet classes are loaded and executed on the Web browser. Let us see what goes into an applet class. Basically, all applets are subclasses of java.applet.Applet. This Applet class inherits properties from several classes of the Java Advanced Window Toolkit (AWT) package. The Applet class inherits user interface capabilities such as displays and event handling from the AWT and adds a number of methods to interact with the Web browser. Examples of these methods are:

- init( )—the method where the applets initialize themselves
- start( )—the method called when the applet starts (i.e., applet page has been loaded or revisited)
- stop( )—the method called when the applet's page is no longer on the screen
- mouseDown( )—the method called to respond to when the mouse button is pressed down
- paint( )—the method called to paint and draw on the screen

Basically, a Java-enabled Web browser supports the libraries and methods needed by the Java applets (i.e., it supports a Java Virtual Machine). When writing Java applets, the user needs to invoke the init, start, mouseDown, paint, stop, and other such methods to interact with users through the Web browsers.

Different browsers (e.g., Netscape Navigator and Microsoft Internet Explorer) do not support the same features of Java because each browser supports its own default JVM that may differ from the others. This leads to compatibility problems for Java applets (i.e., some features are supported by one browser but not by the other). These compatibility issues between Web browsers cause significant problems for Web-based application developers (i.e., applets work for one browser but not for the other). The Sun Java Activator is designed to address this problem.

Sun's Java Project Activator (Also known as Java-Plug-In)

Project Java Activator, henceforth referred to as "Activator," is a Sun Microsystems software package that was announced in December 1997 to run the latest Java Runtime Environment (JRE) independent of the default browsers Java Virtual Machine (JVM).

As stated previously, different browsers do not support the same features of Java and lead to compatibility problems for Java applets. The latest features of Java, when announced by Sun, are not available to software developers unless the browser default JVM is "upgraded." Thus the developers have to wait for the default browser JVMs to be upgraded. In essence, the Activator allows the developers to override the browser default JVM with the Sun's JRE, thus supporting the same features across browsers. The Activator does not replace the browser permanently (for example, it does not remove Internet Explorer from a machine); instead it just plugs in the browser JVM when the user is operating in the "Activator Environment." The developers change their HTML pages to indicate Activator environment. When the Internet Explorer and Netscape Navigator browsers encounter these HTML tags, they invoke the Activator software. A converter is provided that converts the HTML pages to be "Activator aware."

```
<SCRIPT LANGUAGE="JAVASCRIPT">
<!--
document.write("This document was last updated at:"+ document.lastModified)
-->
</SCRIPT>
```

Figure 10. "Last Updated" script.

Main Features of the Project Java Activator. The Project Java Activator is a free software (the Activator software and the converter are free) provided by Sun Microsystems. At the time of this writing, the most current version of the Activator was the Early Access Release 3, (EA3), also known as beta3. The Activator provides the following features for the enterprises using Microsoft's Internet Explorer 3.02 (or later software) or Netscape's Navigator 3.0 (or later software):

- *Full JDK 1.1 Support.* Allows enterprise developers to develop and deploy Java applets, taking full advantage of all JDK 1.1 features and functionality (i.e., RMI, JNI, Java Beans, etc.).
- *Full Java Compatibility Kit (JCK) Test Suite Compliance.* The Project Java Activator software utilizes the latest release of Sun's JRE, which is fully compliant with the JCK test suite.
- *Future-Ready Architecture.* The Project Java Activator software makes it easy for Sun to bring new platform features and functionality, including those contained in JDK 1.2 and the new high-performance Project Hot Spot Java virtual machine, to web browsers quickly.

The Activator Working Model. To make the Java Activator work, the Web page authors make changes to HTML code on their intranet Web pages that host JDK 1.1-based applets. When the browsers (Internet Explorer or Netscape Navigator) are directed at a page with this modified HTML, they know to automatically invoke the Project Java Activator software. Sun has made available a free wizard (the Project Java Activator HTML Converter) that will automatically make the HTML code alterations on the specified Web pages. The operations performed by the browsers are as follows (conceptually, they operate in an identical manner): The first time a user's Web browser comes across a Web page that is enabled for the Project Java Activator product, it automatically downloads and installs the Project Java Activator software (and hence the latest implementation of the JRE) on the user's system. The next time, and from that point forward, the browser will automatically invoke the Project Java Activator software every time it comes across Web pages that support the technology. This is completely transparent to the end user.

JAVASCRIPT

Javascript is a widely used browser scripting language. Browser scripting is an important technique to create dynamic, responsive documents. Javascript is a scripting language with "Java-like" syntax. Many people contend that JavaScript belongs to a "scripting" family and not to "Java family."

Basic Scripts

Figure 10 shows the Javascript code for a common, simple script that mechanizes the "Last Updated" notice on a document, retrieving the update time from the file system so that no manual change is needed when the document is changed.

If placed within an HTML document, this script will generate a "Last Updated" notice.

- "SCRIPT" starts the script.
- "⟨!--" marks as a comment to any browsers that cannot support Javascript.
- "document.write" places the text in the document. Note that the text generated by this script will pass through the HTML interpreter, so it should generate HTML tags, if they are needed. This statement uses the lastModified property of the document object
- "--⟩" closes the comment.
- "/SCRIPT" ends the script.
- HTML is not case sensitive, and Javascript is. Thus, while the word "SCRIPT" could be entered in lowercase as "script," the property name "lastModified" must remain exactly so.

Slightly more complex scripts, especially within a form, may react to browser events, such as mouse movements, as in the example in Figure 11.

Frequently, scripts are written as functions, which can then be invoked. Because it loads first and because a function must be loaded before it is called, Javascript functions are usually placed within the HEAD section of the HTML document. Figure 12 shows an example of a function in a script.

HTML, Javascript and Java

In understanding the benefits of Javascript, it is helpful to review where it fits into the browser world:

```
<FORM NAME="ThisForm">
<INPUT TYPE="Text" NAME="Text1" VALUE="Initial Value of Text1"
onMouseOver="document.bgColor='blue'">

<INPUT TYPE="Text" NAME="Text2" VALUE="Initial Value of Text2"
onMouseOver="document.bgColor='red'">
</FORM>
```

Figure 11. "onMouseOver" Script.

```
<HEAD>
<SCRIPT LANGUAGE="JAVASCRIPT">
<!--
function show Alert() {
 alert("Warning: Don't type in this field unless you know what you are doing!")
}
</HEAD>
<FORM NAME="ThisForm">
<INPUT TYPE="Text" NAME="Text1" VALUE="Initial Value of Text1"
onMouseOver="showAlert()">
</FORM>
```

Figure 12. "onMouseOver" script with function.

- HTML is a primarily a document formatting language.
- Javascript has access to most of the features of the browser. It operates on the document as a single object, and it can receive events from the browser and can invoke methods on the browser.
- Java is a fully featured programming language. However, Java applets simply manage the display in a fixed-size rectangle on a web page and, perhaps, launch window frames outside of the browser. They have a very minimal direct connection to the browser, except for very few specific functions like requesting a new page be displayed in the document window. Browser-specific solutions like LiveConnect improve the connection to the browser substantially, but not fully, at the cost of browser-specificity.

Javascript and Jscript

Javascript is often used generically, although strictly speaking, it is associated with the Netscape Navigator (NN) Web browser. The Microsoft Internet Explorer (IE) web browser has implemented a similar language, referred to as "Jscript." Both scripting languages approximate the "ECMAscript" scripting language standard. Indeed, many simple scripts work in both browsers. As new features are added to one language or the other, however, the languages become less compatible. Scripts that take advantage of features only implemented in one variant can isolate a browser-specific code by examining the "navigator" object as in Fig. 13.

A similar script can check for the index of a specific release number, to verify the release level of the browser.

Other Browser Scripting Languages

Javascript and Jscript are not the only browser scripting languages.

```
<SCRIPT LANGUAGE="JavaScript">
<!--
 var ver=navigator.appVersion;
 if (ver.indexOf("MSIE")!=-1)
 {
 // perform IE-specific functions
 } else {
 // perform NN-specific functions
 }

// -->
</SCRIPT>
```

Figure 13. Isolating Javascript and Jscript code.

- VBscript is another scripting language available in the Microsoft IE browser. VBscript is similar in functionality to Jscript, but its syntax is more like Basic.
- Dynamic HTML is an evolution of HTML providing greater control over document display, on its own, and more so in combination with a regular browser scripting language.

Additional means to provide dynamic content are sure to emerge.

COMBINING JAVA WITH DISTRIBUTED OBJECTS—JAVA AND CORBA

Increasingly, Java applications are distributed across machines by using distributed object middleware such as CORBA. Let us work through some details.

Let us assume that a Java applet needs to invoke some Java code on another machine. In this case, the following choices are available:

- User-written low-level code (e.g., TCP sockets) to invoke the remote Java code.
- Use of an off-the-shelf middleware package such as ONC RPC (remote procedure call).
- Use of, if possible, distributed object middleware such as CORBA.

The first choice is not very attractive. The second choice does work but the user must translate the object model followed by Java to a procedural model. The third option is the most natural and consequently most popular. The prominent middleware options are as follows:

- CORBA calls
- DCOM calls
- Sun's Remote Method Invocation (RMI)

The first choice is most common at present. DCOM, introduced by Microsoft, is proprietary and available primarily on Windows platforms. RMI was introduced by Sun specifically for distributing Java applications. However, RMI is only restricted to distributed Java (C++ code cannot be distributed by using RMI). In addition, RMI has failed to gain market prominence at the time of this writing. CORBA, on the other hand, is open and offers a general-purpose distributed

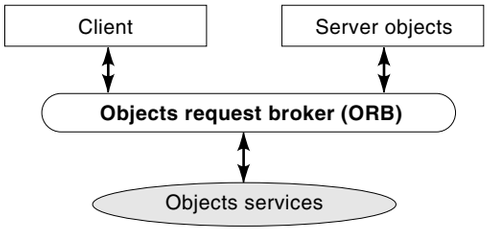

Figure 14. The basic distributed objects model.

object solution. Due to this reason, many distributed Java applications at present use CORBA. As a result, a brief overview of distributed objects and CORBA follows. Additional details about combining Java with CORBA can be found in Ref. 6.

Simply stated, *distributed objects* are objects that can be dispersed across the network and can be accessed by users/applications across the network. Figure 14 shows a conceptual view of a distributed object model:

- *Objects* are data surrounded by code with properties such as inheritance, polymorphism, encapsulation, and so on. Objects can be clients, servers, or both.
- *Object brokers* allow objects to dynamically find each other in a distributed environment and interact with each other over a network. Object brokers are the backbone of distributed object-oriented systems.
- *Object services* allow the users to create, name, move, copy, store, delete, restore, and manage objects.

Support of distributed object-based applications requires special-purpose middleware that allows remotely located objects to communicate with each other. Examples of middleware for distributed objects include Object Management Group's (OMG's) CORBA (Common Object Request Broker Architecture), Microsoft's ActiveX/DCOM, and Sun's RMI (Remote Method Invocation). These middleware packages use the distributed object model based on the object request broker (ORB) that receives an object invocation and delivers the message to an appropriate remote object (see Fig. 14).

CORBA was introduced in 1991 by OMG to specify the technology for interoperable distributed OO systems. CORBA specifications represent the ORB technology adopted by OMG and are published as OMG documents. The key concepts of CORBA are as follows (see Fig. 15):

- CORBA essentially specifies the middleware services that will be used by the application objects.
- Any object (application) can be a client, server, or both. For purpose of description, CORBA uses the client–server model where clients issue requests to objects (service providers).
- Any interaction between objects is through requests. The information associated with a request is an operation to be performed, a target object, zero or more parameters, and so on.
- CORBA supports *static* as well as *dynamic binding*. Static binding is used to identify objects at compile time, while dynamic binding between objects uses run-time identification of objects and parameters.
- An *interface* represents the services to be provided by the server applications. A typical interface definition shows the parameters being passed and a unique interface identifier. An *interface definition language* (*IDL*) has been defined specifically for CORBA. Program stubs and skeletons are produced as part of the IDL compiling.
- CORBA objects do not know the underlying implementation details: An *object adapter* maps generic model to implementation and is the primary way that an object implementation accesses services provided by the ORB.

To use CORBA from Java applets, the user can invoke CORBA directly from applets executing under the control of the Web browser. Current browsers support the CORBA calls directly. Thus, the Web browser sites behave as CORBA clients. The user can also use CORBA to interact between Java applications across machines (this may not have anything to do with Web).

In general, seamless integration of corporate information (e.g., relational databases, IMS databases, indexed files, Co bol subroutines, 3270 terminal sessions, or a combination thereof) through Web and Java by using distributed objects is a common practice at present. The distributed object middleware is used for translation of requests and data between host applications, synchronization of updates between the host applications, and support of intelligent features such as

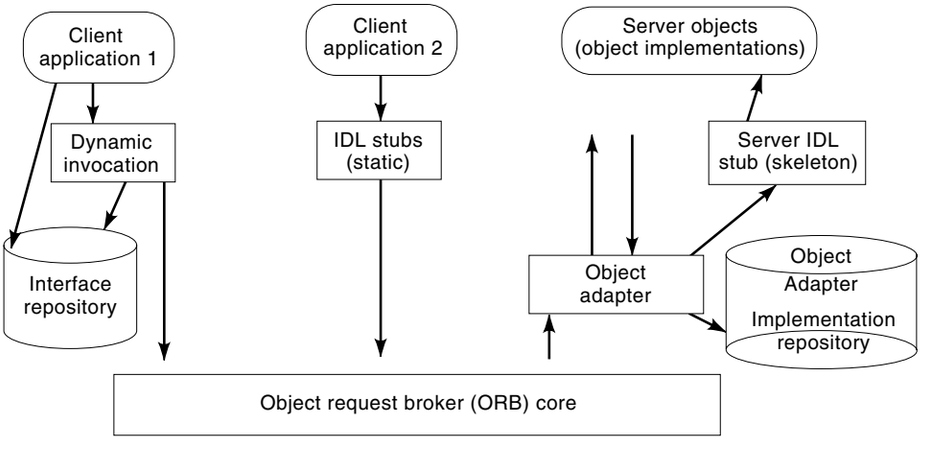

Figure 15. CORBA conceptual view.

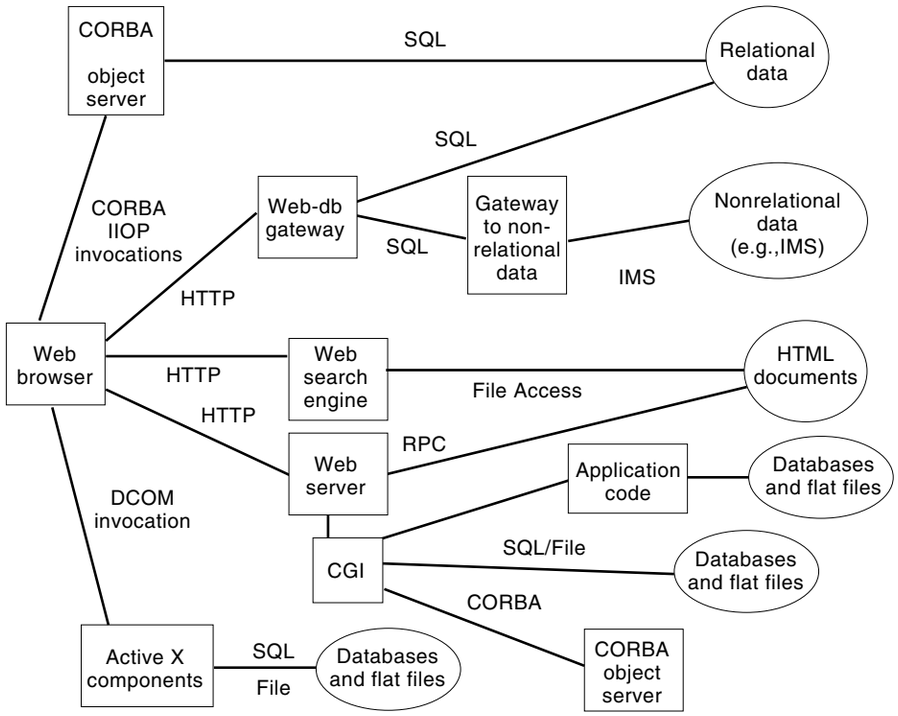

Figure 16. Object-oriented client/server Internet applications.

distributed query processing and distributed transaction processing. The solutions can also combine a wide array of technologies for a more general, albeit complicated, solution (see Fig. 16).

SUMMARY AND TRENDS

This article has briefly reviewed Java, Javascript, Java-enabled browsers (e.g., Hot Java, Microsoft Internet Explorer, Netscape Navigator), and other related technologies such as CORBA. The following trends are worth noting:

- The popularity of Java as a serious programming language is rising steadily.
- Many software development tools are becoming Java-based.
- Java and distributed objects, especially the Java–CORBA combinations, are becoming increasingly popular.
- Other technologies for increasingly flexible Web pages are evolving, including display technologies (like Dynamic HTML) and programming technologies (such as ActiveX).

BIBLIOGRAPHY

1. W. Gibson, *Neuromancer,* New York: Ace Books, 1984.
2. A. Hoff et al., *Hooked on Java; Creating Hot Web Sites with Java Applets,* Reading, MA: Addison-Wesley, 1996.
3. P. Philips, Brewing up applications with Java, *Internet Advisor,* **January**: 14–17, 1996 (premiere issue).
4. D. Dean, E. Felten, and D. Wallach, Java security: From HotJava to Netscape and beyond, *Proc. 1996 IEEE Symp. Security Privacy,* 1996.
5. T. Lindholm and F. Yellin, *The Java Virtual Machine Specification,* Reading, MA: Addison-Wesley, 1996.
6. R. Orfali and D. Harkey, *Client/Server Programming with Java and CORBA,* 2nd ed., New York: Wiley, 1998.

Reading List

B. Barron, J. Ellsworth, and K. Savetz (eds.), *Internet Unleashed,* 2nd ed., Indianapolis, IN: Sams Net, 1995.

H. Berghel, The client's side of the World Wide Web, *Commun. ACM,* **39**(1): 30–40, 1996.

T. Berners-Lee and R. Cailliau, World Wide Web, *Comput. High Energy Phys. 92,* Anney, France, 1992.

T. Berners-Lee and D. Connolly, Hypertext Markup Language 2.0, RFC 1866, *IETF,* **November**: 1995.

T. Berners-Lee et al., The World Wide Web initiative, *Proc. INET '93,* Internet Society, San Francisco, 1993.

R. Bicket, Building Intranet, *Internet World,* pp. 72–75.

M. Bjorn, A WWW gateway for interactive relational database management, Doctoral Program of Socioeconomic Planning, 1-1-1 Tennodai, Tsukuba, Ibaraki 305, Japan, 1995.

M. L. Brodie and M. Stonebroker, DARWIN: On the incremental migration of legacy information systems, Technical memor., Electronics Research Laboratory, College Engineering, Univ. California, Berkeley, 1993.

D. Chadwick, A method for collecting case study information via the Internet, *IEEE Netw.,* **10**(2): 36–38, 1996.

D. Chandler, *Running a Perfect Web,* Indianapolis, IN: Que Books, 1995.

D. Comer, *Internetworking with TCP/IP: Principles, Protocols, Architectures,* Englewood Cliffs, NJ: Prentice-Hall, 1988.

D. Comer, *Internetworking with TCP/IP,* , Englewood Cliffs, NJ: Prentice-Hall, 1991.

J. December and N. Randall, *The World Wide Web Unleased,* Indianapolis, IN: Sams Net, 2nd ed., 1995.

J. Gosling et al., *The Java Language Specification,* Reading, MA: Addison-Wesley, 1996.

I. Graham, *HTML Source Book,* 2nd ed., New York: Wiley, 1996.

H. Hahn, *Internet: Complete Reference,* 2nd ed., Berkeley: Osborne McGraw Hill 1996.

F. Halasz and M. Schwarz, The Dexter hypertext reference model, *Commun. ACM,* **37** (2): 30–39, 1994.

J. Kador, The ultimate middleware, *Byte Mag.,* **April**: 79–84, 1996.

P. Kent and J. Kent, *The Official Netscape JavaScript Book,* Research Triangle Park, NC: Ventana Communications Group, 1996.

L. Perrochon, W3 middleware: Notions and concepts, Institut fur Informationssysteme, ETH Zurich, Switzerland, 1995.

H. Schulzrinne, World Wide Web: Whence, whither, what next?, *IEEE Netw.,* **10**(2): 10–17, 1996.

E. Tittel and S. James, *HTML for Dummies,* Forest City, CA: IDG Books, 1995.

P. Varhol and V. McCarthy, Who wins the Web server shootout, *Datamation,* **April 1**: 48–53, 1996.

G. Wiederhold, Mediators in the architecture of future information systems, *IEEE Comput.,* **25**(3): 38–49, 1992.

http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html: T. Berners-Lee, HTTP: A protocol for networked information, CERN, IEFT Internet draft, 1994, original version 1991.

RICHARD WIKOFF
AMJAD UMAR
Bellcore