Technical overview of IBM's Java initiatives

by K. D. Gottschalk

This paper gives an overview of some of IBM's major Java™ efforts and sets forth a structure that relates the individual Java efforts to one another. The paper describes IBM's overall approach toward Java and describes how IBM is exploiting Java to answer customer requirements in such areas as server platforms, reusable components, and tools. This paper will serve as an introduction to some of the other papers that are included in this issue of the* IBM Systems Journal *and that detail individual areas of IBM's focus on Java.

Over the last few years, IBM has seized upon the Java** programming language as a way to help its customers meet today's business challenges. In IBM we believe that the simplicity, portability, scalability, and security associated with Java will help our customers to meet their customers' needs. IBM's efforts with Java scale from the microchip to the enterprise and involve applications, infrastructure, and tools.

One of the main reasons why IBM's customers are excited about the potential of Java is that IBM's plans for exploiting Java technology allow users to easily access existing data, services, and applications residing on IBM mainframe systems such as S/390* (System/390*) and AS/400* (Application System/400*). It is estimated that 70 percent of the critical business data in the world reside on mainframe computers. Without Java, mainframe data and services can be reached and extended to the new interconnected world of the World Wide Web only by writing applications that deal with a myriad of proprietary application programming interfaces (APIs). Java, with its standardized APIs for invoking different system services and data, serves as the glue to bind existing data and applications to this new world. Thus, Java enables our customers to have the best of both worlds, combining the security and availability of systems that have been developed over many years with the new reach and immediacy of information available through the Internet and the World Wide Web.

This issue of the *IBM Systems Journal* describes several major efforts by IBM in such areas as Java for use in the enterprise, Java frameworks for application development, Java components, and Java tools. In this paper, we provide an overall picture of some of IBM's major Java efforts and point to some of the other papers in the issue for more detail.

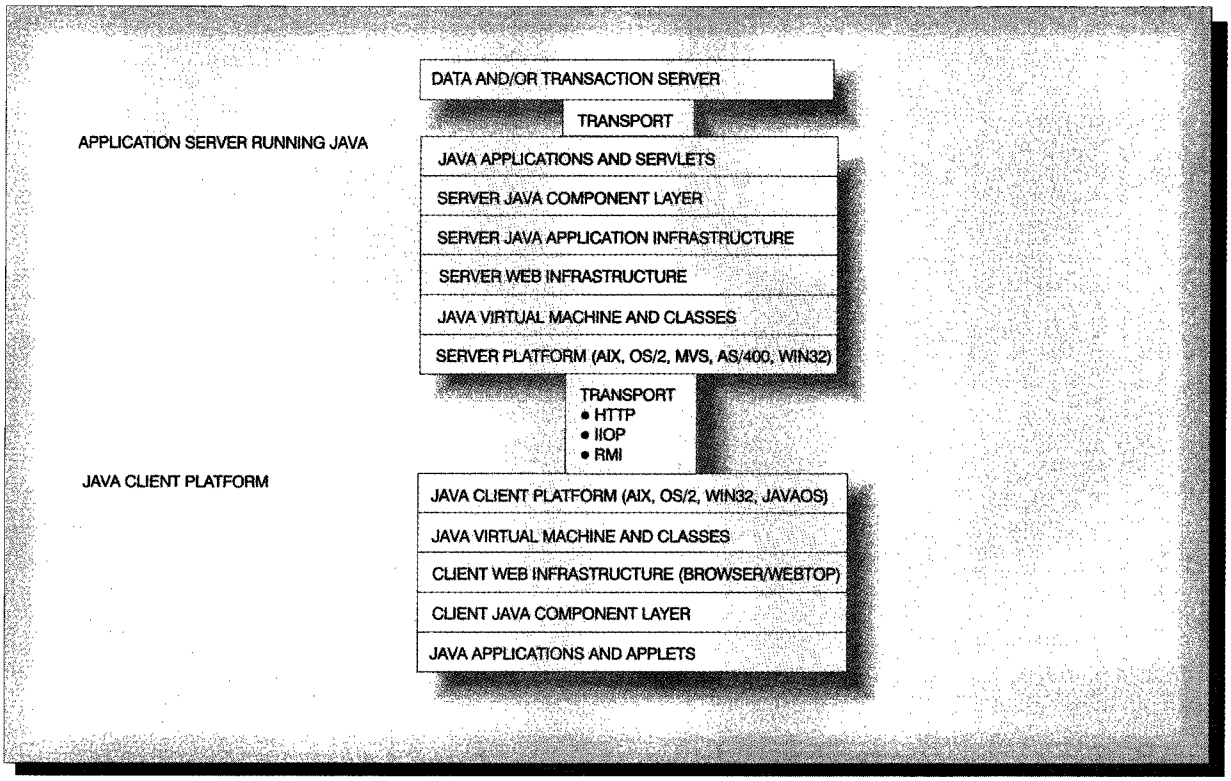
This paper assumes a basic familiarity with Java, the Java Development Kit (JDK**), the Java Virtual Machine (JVM), client/server computing, and network computing. There are many introductory books and presentations on Java; for a good technical introduction to Java see Flanagan.[1]

IBM Java architecture overview

Figure 1 sets forth the overall architectural structure of the IBM Java implementations. The architecture is divided into three tiers: client, application server, and data/transaction server. The application server and the client platform both execute programs written in Java and are connected via an Internet or intranet transport layer. The third tier consists of a data server or transaction server, or both, that typically

0018-8670/98/$5.00 © 1998 IBM

Figure 1 IBM Java architecture overview



provides non-Java services to Java applications residing on the application server.

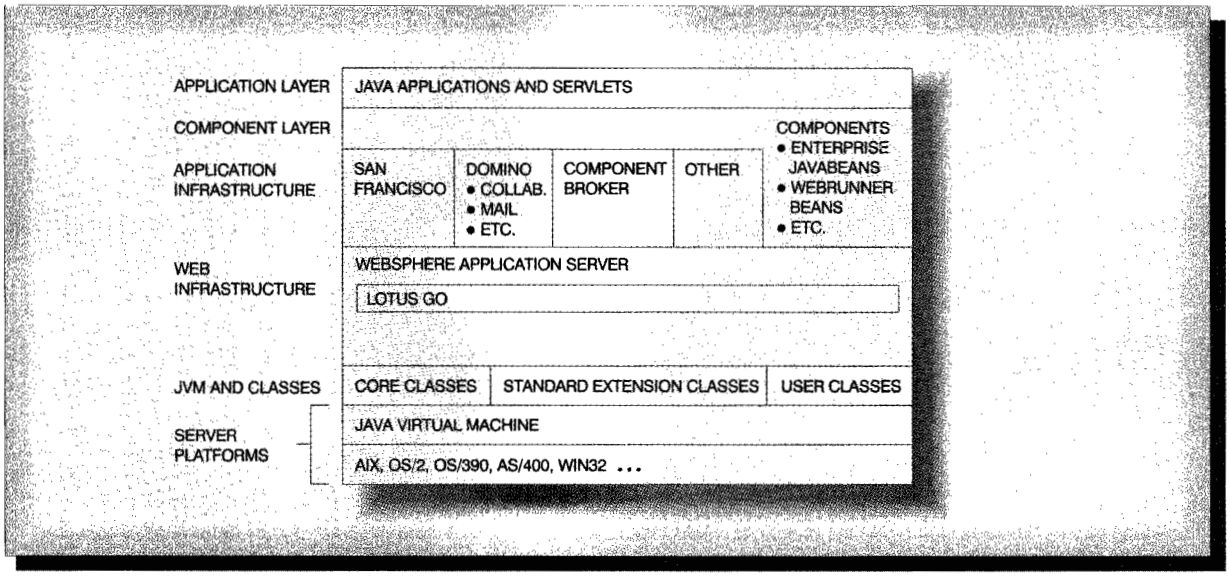
The multitier architecture illustrated in Figure 1, with connections between tiers, incorporates a logical, rather than a physical distinction. The three types of components may be located on different physical platforms, or they may all be located on the same physical platform. Although Figure 1 shows one application server and a single Java client, in reality each application server will communicate with many Java client platforms, and a single Java client platform may communicate with multiple application servers. Finally, multiple application servers may communicate with one another.

In order to more readily understand the components in the IBM view of Java, we have grouped them into layers, as indicated in Figure 1. Within the application server, the applications and servlets supporting Java make use of reusable software components called JavaBeans** that reside in the server Java component layer. Below the component layer is the server Java application infrastructure, an application-oriented set of services that provide the Java applications with needed common functionality; an example of such a service might be a framework oriented toward the development of integrated business applications.

Below the server Java application infrastructure is the server Web infrastructure, consisting of a set of services that provide general support for Java on the application server. An example of such functionality might be some classes to support the execution of servlets in a Java environment.

Below the server Web infrastructure is the Java Virtual Machine, or JVM, which supports Java components on the application server with basic Java functionality. The JVM is hosted on a server operating system, which might be a UNIX** platform such as AIX* (Advanced Interactive Executive), a Microsoft Windows** platform, OS/400* (Operating Sys-

Figure 2 Application server running Java



tem/400), OS/2* (Operating System/2*), OS/390* (Operating System/390), or VM/ESA* (Virtual Machine/Enterprise Systems Architecture*).

A transport layer connects the application server and Java client platforms, as well as the application server and data or transaction server platforms.

The Java client platform provides the interface to the end user. In the Java environment, the client platform might be AIX, OS/2, a Windows platform, a network computer (NC), or any other platform running the JVM and having the appropriate graphical user interface for communicating with the end user. In the client, the JVM and other client run-time infrastructure support Java client applications and applets that facilitate interaction with the end user via the client browser or webtop. As can be seen from Figure 1, layering of functionality on the Java client generally mirrors that on the application server.

We next look in more detail at IBM's particular infrastructure for the application server.

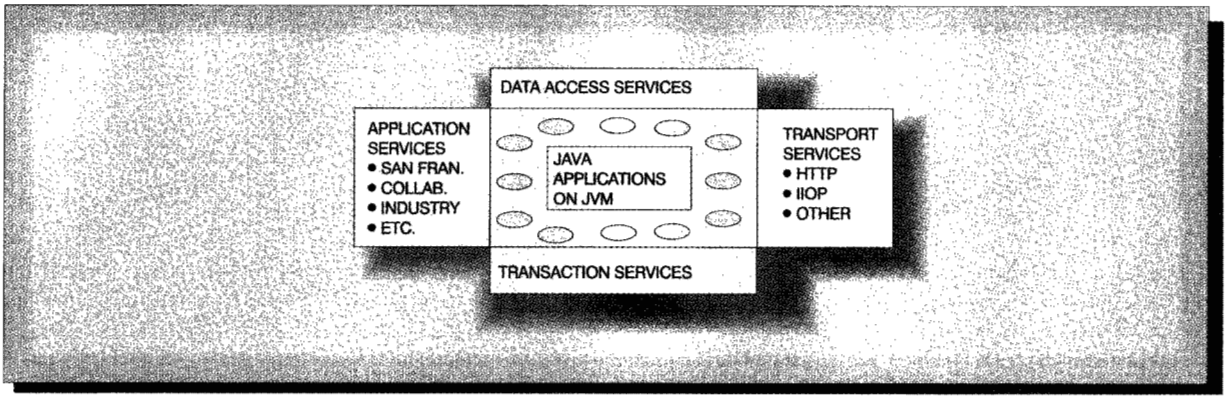
Application server infrastructure

IBM's implementation of the application server running Java is governed by a model called the IBM Network Computing Framework (NCF) for e-business. NCF is a Java-based model and set of services that ties together the IBM and Lotus Development Corporation server offerings and provides a Java-based set of server functionality that runs on all major Java platforms. NCF integrates the use of Java as the application programming environment with Component Object Request Broker Architecture (CORBA**) as the object-oriented infrastructure for distributed computing.

Figure 2 illustrates major IBM components associated with the application server running Java. The IBM VisualAge* tool makes it easier to use these components when building enterprise Java applications and servlets. The actual components in each layer will depend on the customer configuration; the richness of function in each layer will vary from customer to customer, depending on what products and services the customer installs. Throughout the rest of this paper, we discuss the components illustrated in Figure 2 in more detail.

At the top of the figure are Java applications and servlets that are used to bring enterprise information to a wide audience via the Internet. *Applications and servlets operating in the Java environment are constructed using reusable components called JavaBeans so as to make such applications easier to build.* We believe that the success of Java depends on IBM and

Figure 3 Java applications and JavaBeans



other Java suppliers providing JavaBeans components to facilitate the construction of Java applications. JavaBeans are a part of JDK Version 1.1; they provide a means of packaging functionality so that it may be easily reused by multiple applications and may easily communicate with other application functionality. JavaBeans lower the skill level required to assemble Java-enabled applications by allowing the user of a tool such as VisualAge to assemble applications from parts through visual manipulation, rather than having to do coding. The software-development world has been looking for a long time for a way to easily reuse components across multiple environments, and we believe that JavaBeans represent a major contribution toward solving this problem.

Figure 3 is a bean-centric view of Java applications, servlets, and applets. In Figure 3, Java programs running on the client or on the application server make use of many common services by means of JavaBeans. These can include:

- Beans to access existing relational databases
- Beans to talk with other applications on the client or server by means of HyperText Transfer Protocol (HTTP), Internet Inter-Orb Protocol (IIOP), or another transport protocol
- Beans to access applications and data in transaction monitors such as CICS* (Customer Information Control System)
- Beans to access application services provided by Lotus, San Francisco*, or other frameworks

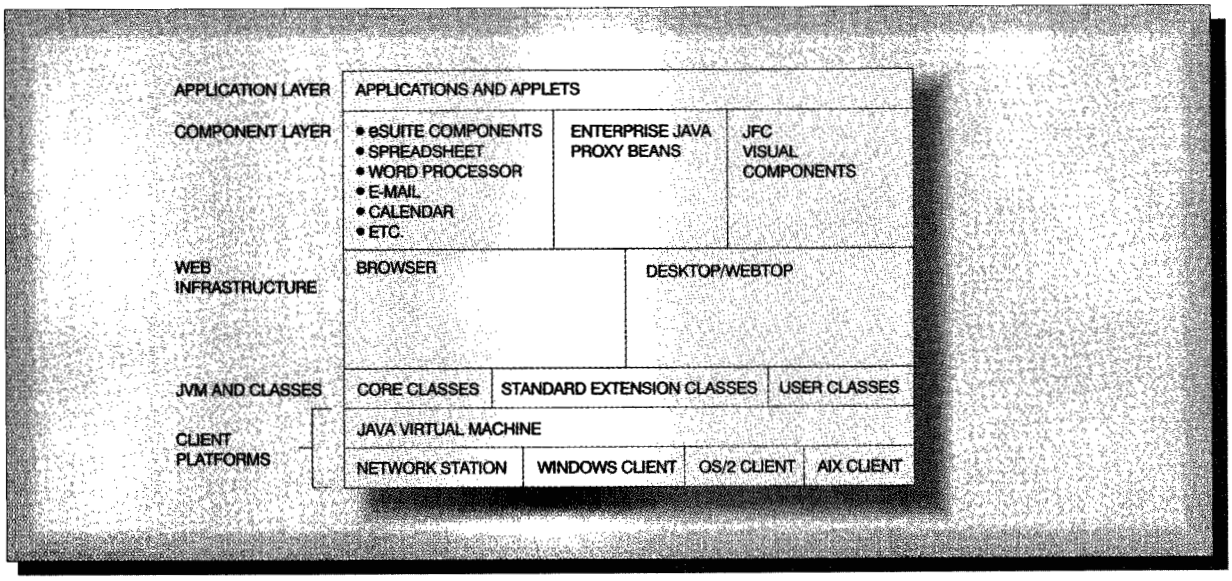
By providing application services via JavaBeans, we lower the skill level and experience required to write Java applications and applets. Since JavaBeans are reusable components, they can be assembled into applications and applets using visual assembly tools, rather than requiring programming to be the means to achieve the desired result.

IBM development groups and other companies producing Java applications have realized the potential of JavaBeans and are developing new functions as JavaBeans. Figure 2 lists some of the IBM-provided server-side JavaBeans components. They are described in more detail later in the section entitled "IBM's component approach and JavaBeans."

IBM groups are also developing many types of server application infrastructure components aimed at various types of applications. Figure 2 mentions the San Francisco frameworks to support commercial applications, Lotus Domino** services supporting various types of collaboration, and Component Broker services for object-oriented business applications. Some of these are implemented in Java and some are not, but all provide interfaces to Java applications and servlets running on the application server.

IBM has added value to the base Java components in the application server by providing a robust set of server functions oriented toward the development of business applications. At the Java application infrastructure layer, the San Francisco project consists of a set of Java server-side frameworks that provide the foundation for integrated business applications for small and medium-sized enterprises. San Francisco includes building blocks and prefabricated templates that will be extended by IBM Business Partners and independent software vendors (ISVs) to

Figure 4 Java client infrastructure



provide integrated application suites. San Francisco encourages the integration of "best-of-breed" applications from different vendors using the same IBM-supplied San Francisco frameworks.

San Francisco is very much ISV-oriented, with many of those who are Business Partners participating in advisory councils as the framework is developed. San Francisco broadens market opportunity for ISVs by providing cross-platform extendible frameworks, network enablement, global reach through national language support, multicurrency frameworks, and easy integration with other applications based on San Francisco. For more information on San Francisco, see the paper by Rubin et al. in this issue.[2]

Lotus Domino provides a variety of components for collaboration, mail, etc.; all of these are available to Java applications and servlets via Java APIs.

Component Broker is a server run-time environment and programming model that provides a rich, transactional, CORBA object-based development environment for distributed applications programming. Component Broker includes a run time and toolkit to enable rapid application development of reusable distributed object applications, and provides a robust programming model that allows developers to focus on business logic. For more information on Component Broker, see the IBM application development website at http://www.software.ibm.com/ad/.

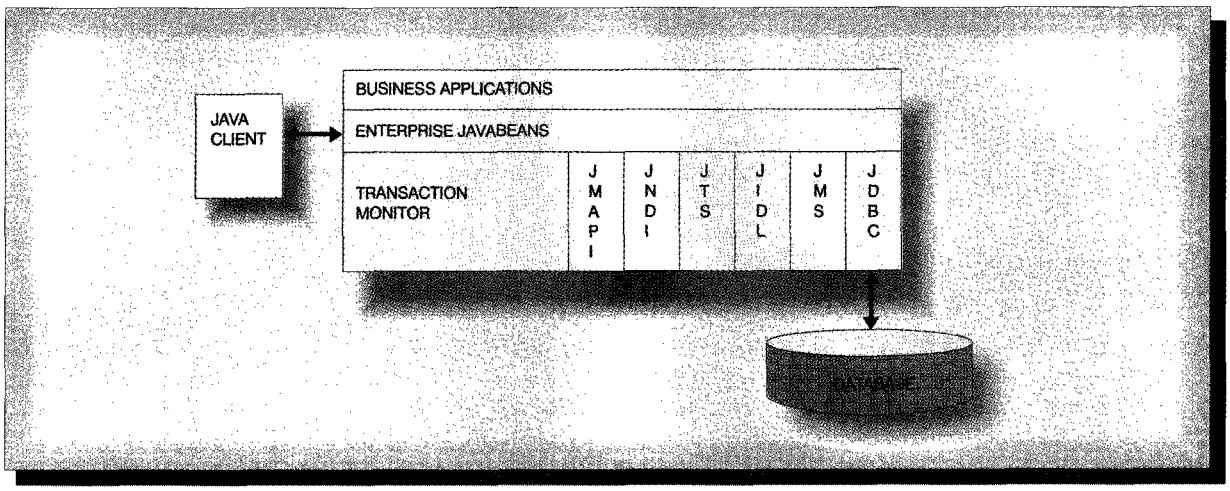
At the Web infrastructure layer, Lotus Domino Go** is IBM's Internet Web server platform and is the basis of the Lotus Domino functions in the application infrastructure layer. The IBM WebSphere Application Server* provides Java servlet support for the Lotus Go Web server as well as other leading industry Web servers. It includes a set of services providing rich functionality to Java servlets. For more information on the IBM WebSphere Application Server, see the paper by Bayeh in this issue.[3]

Client infrastructure

Figure 4 gives an overview of the Java client infrastructure. This infrastructure is similar to the application server infrastructure illustrated in Figure 2 but has components that are directed to a client rather than to a server platform.

On the client, 100 percent pure Java applications such as IBM's Host on Demand emulator products and those built out of Lotus eSuite** components coexist with native platform applications. As is the case with application server applications, the client Java applications run in a Java environment centered around the Java Virtual Machine and associated sup-

Figure 5 Java for the Enterprise



porting classes. On the Java client, the Java foundation classes for user interface components and widgets make available extensive user interface capability. Of particular interest as components on the Java client are the set of eSuite components from Lotus. eSuite provides a lightweight set of productivity components implemented as JavaBeans that facilitate the quick development of business applications. For a description of Lotus eSuite see the paper by Briggs in this issue.[4]

As is the case with the application server, the IBM Java client implementations are run on a variety of platforms, both IBM and non-IBM. In fact, because the implementations are based on Java, they will run on any client platform that implements the appropriate Java run-time environment.

Java for the Enterprise initiative and the Network Computing Framework

The IBM Java strategy for application servers is twofold:

1. To enable access to the broad range of IBM and Lotus server technology through the integrating model of NCF and Java APIs
2. To exploit the Java for the Enterprise initiative of JavaSoft (a division of Sun Microsystems, Inc.) and the concept of a richer set of server Java APIs that this implies.

In this section, we briefly describe the Java for the Enterprise initiative and describe the Network Computing Framework and how it fits into this architecture.

Java for the Enterprise initiative. The Java for the Enterprise initiative, announced by JavaSoft at the JavaOne Conference in April 1997, consists of a set of standardized APIs for enterprise-level distributed Java applications and a set of extensions to the JavaBeans architecture to allow enterprise applications to easily participate in important enterprise services. In this subsection, we describe the major components of this initiative at a high level.

Figure 5 illustrates the Java for the Enterprise platform, focusing on Java APIs on the server and JavaBeans.[5]

Java enterprise APIs. Working with its Java partners, JavaSoft has recently defined several APIs that allow client and server applications, applets, and servlets to access common server-side systems and subsystems. This approach facilitates the creation, deployment, and management of scalable Java business applications. This architecture allows application developers to focus on their business logic without having to worry about the plumbing that makes the server environment so complex today. These APIs are depicted in Figure 5.

In Figure 5, JMAPI refers to the Java Management API, which defines access to a set of services for managing Java resources. JNDI, the Java Naming and Directory Interface, is an API for accessing naming and directory services. JTS is the Java Transaction Service, an API for invoking transaction services. JIDL refers to Java interface definition language, an interface to the CORBA set of services for distributed computing. JMS, the Java Message Service, is an API for invoking asynchronous message delivery services. Finally, JDBC**, the Java Database Connectivity API, accesses data in existing databases through a common interface.

Figure 5 shows that business applications may participate in subsystem services via Enterprise JavaBeans**, a Java model for software components that encapsulate business logic and run on an application server. Enterprise JavaBeans allow people to assemble applications that run on the server and can take advantage of the Java enterprise services. Using a visual assembly tool such as VisualAge for Java, users will be able to utilize Enterprise JavaBeans to do such tasks as accessing existing data and defining transactions without having to write code, thereby realizing a major promise of the Java enterprise technology—giving companies the tools they need to directly define their company policies in terms of applications that are responsive to those policies, without having to acquire complex programming skills.

Figure 5 shows a transaction monitor intimately associated with the enterprise APIs and Enterprise JavaBeans. IBM products that provide a transaction-processing environment and plan to support Enterprise JavaBeans include CICS, Component Broker, and TXSeries*.

The IBM strategy with respect to the Java for the Enterprise initiative is simple: IBM will provide the necessary linkages between enterprise APIs and IBM subsystems that perform the appropriate services. IBM will supply Enterprise JavaBeans support in its major subsystems for these services. Finally, the IBM VisualAge for Java tool will include support allowing Java application writers to easily access appropriate functions and information using these APIs, including Enterprise JavaBeans support for these APIs.

The Enterprise Server for Java (ESJ) specification spells out how IBM is bringing the Java for the Enterprise initiative to IBM platforms. The ESJ specification prescribes specific APIs, services, and Enterprise JavaBeans support at the application-infrastructure and Web-infrastructure layers of the application server running Java that, when adhered to, transform such an application server into an Enterprise Server for Java. For more information on IBM's efforts with respect to Java for the Enterprise, see the article by Brackenbury et al. in this issue.[6]

The Network Computing Framework. The Network Computing Framework (NCF) for e-business, which was announced in April 1997, represents IBM's approach to providing Java-oriented services and capabilities to facilitate electronic business over the Web. Figure 6 illustrates a major idea behind NCF, which is to provide a coordinated set of IBM software servers centered around Java APIs and JavaBeans (including Enterprise JavaBeans), and provide Java applications with access to major services of the underlying server. With use of NCF technology and associated tools, Java applications can easily be built on the server to take full advantage of the powerful and robust system services that IBM has developed over the years.

NCF consists of the following components:

- An open, pluggable framework centered on Java
- A set of software servers that are accessed through standard protocols and Java interfaces
- A set of clients that exploit *just-in-time* downloadable components such as those in eSuite
- Tools that exploit the JavaBeans component standard
- Standard technologies to link components, such as HTTP and IIOP
- A set of built-in groupware solutions and a foundation for e-business applications
- A set of connectors to existing data and transactions on the third tier

Figure 7 illustrates NCF services. NCF provides for a Web server and Java Virtual Machine together with Java interfaces to many services. NCF provides application programming support in the form of development tools and support for JavaBeans, applets, and servlets. The Web server and Java-oriented application programming environment are base services of the NCF.

Additional optional components of NCF include software-server towers for community (mail and related services), collaboration, storing, and retrieving data located in relational databases and distributed files, transaction support, and delivery services. IBM's im-

Figure 6 NCF and JavaBeans

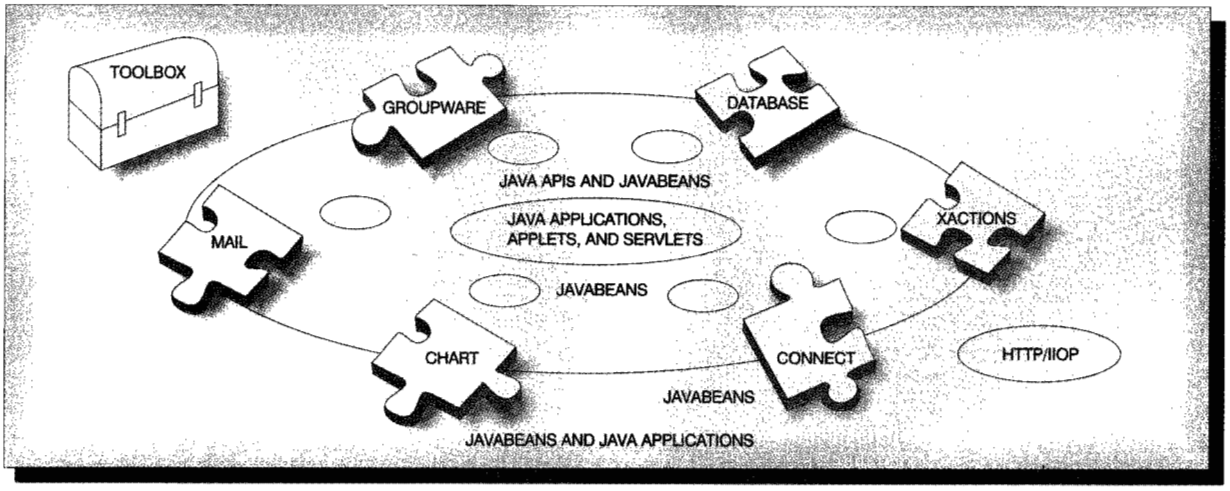
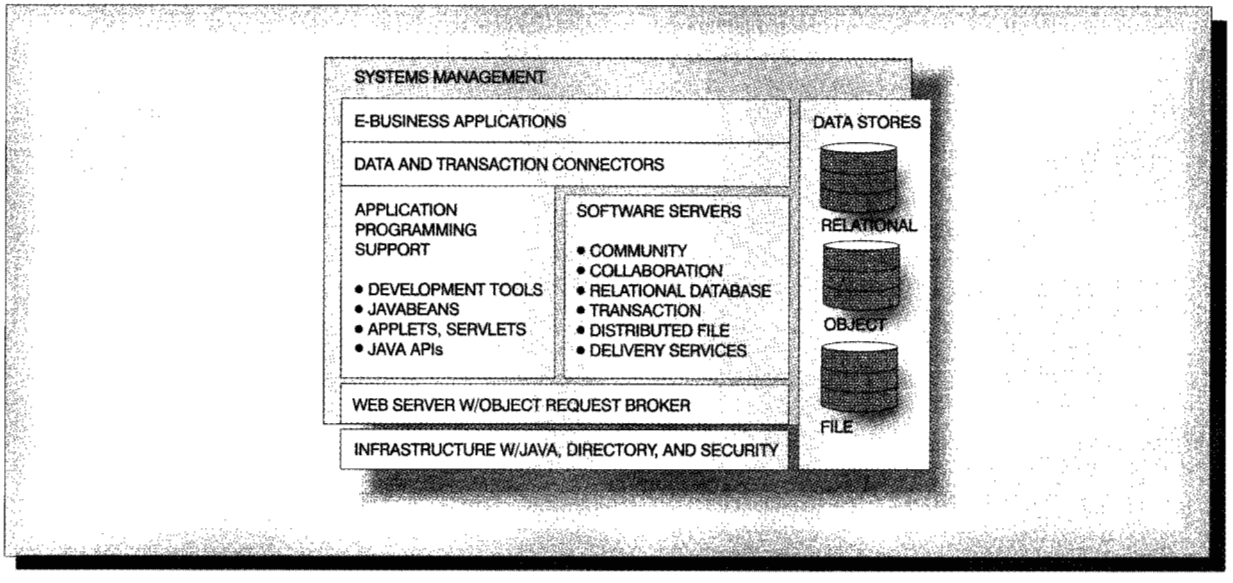


Figure 7 NCF services



plementations of these towers are based on appropriate IBM and Lotus technology, including the entire collection of Domino services and CICS; server towers access the appropriate underlying data stores. As has been stated previously, a major thrust of NCF is that these services are made available to Java applications by means of Java APIs and JavaBeans.

Finally, NCF includes optional systems management services that may be used to manage all of the above components.

NCF supports both e-business applications written in programming languages such as C and C++ and applications written in Java. Java applications access

NCF services by means of Java APIs and JavaBeans. Accessing NCF services in this way greatly reduces the complexity required to build applications in the enterprise server environment and reduces the likelihood of making mistakes, thereby enhancing reliability.

IBM has already made available many servers and components conforming to the NCF model. The Enterprise Server for Java[6] incorporates many of the ideas of NCF, whereas the IBM WebSphere Application Server[3] implements many server elements of the NCF model.

In this section we have looked at the architecture and some of the major components of NCF. For more information on NCF, see the IBM e-business website (http://www.software.ibm.com/ebusiness) and the paper by Bayeh in this issue.[3]

IBM's component approach and JavaBeans

IBM's customers have for many years been seeking productivity breakthroughs that they believed could be obtained through large-scale software reuse. They want ways to produce reusable components that can be used in arbitrary combinations to create user applications. The availability of such components and tools to manipulate them would lower the skill level and experience required to create applications that are responsive to the customers' needs. Instead of having programmers create applications, business policymakers could utilize visual development environments to "wire together" collections of reusable components to meet their exact needs.

Until the advent of JavaBeans, only modest progress had been made toward achieving this goal. IBM's VisualAge tool provides the capability to create and manipulate reusable components in various programming environments, such as C++ and SmallTalk, but the resulting objects and programs run only in certain limited environments. With JavaBeans, IBM has the opportunity to at last achieve its goal and give customers the benefits of visual computing. Applications composed wholly or in part of JavaBeans can be run on any Java-enabled platform.

JavaBeans takes Java's "write once, run anywhere" capability and extends it to include "reuse everywhere." JavaBeans are reusable components. Components are combined from disparate sources to create applications quickly and easily. To streamline the development process, programmers build small, reusable pieces called JavaBeans and then wire them together with the help of a visual builder program, such as VisualAge for Java or Lotus BeanMachine** (described later in this paper under "Tools overview").

Figure 8 illustrates the promise of JavaBeans. As shown in the figure, the customer can use beans obtained from a variety of sources to quickly create a customized application that meets his or her needs exactly. Beans range from small programs that perhaps encapsulate company-specific business rules, to large generic application components, such as word processors and spreadsheets. The customer would use a tool such as VisualAge or Lotus BeanMachine to assemble the application visually. The components themselves could be produced using a variety of IBM and non-IBM tools. Once assembled, the 100 percent pure Java application can be run on any platform that supports the Java Virtual Machine.

IBM JavaBeans development efforts. There are four types of IBM development efforts going on with respect to JavaBeans:

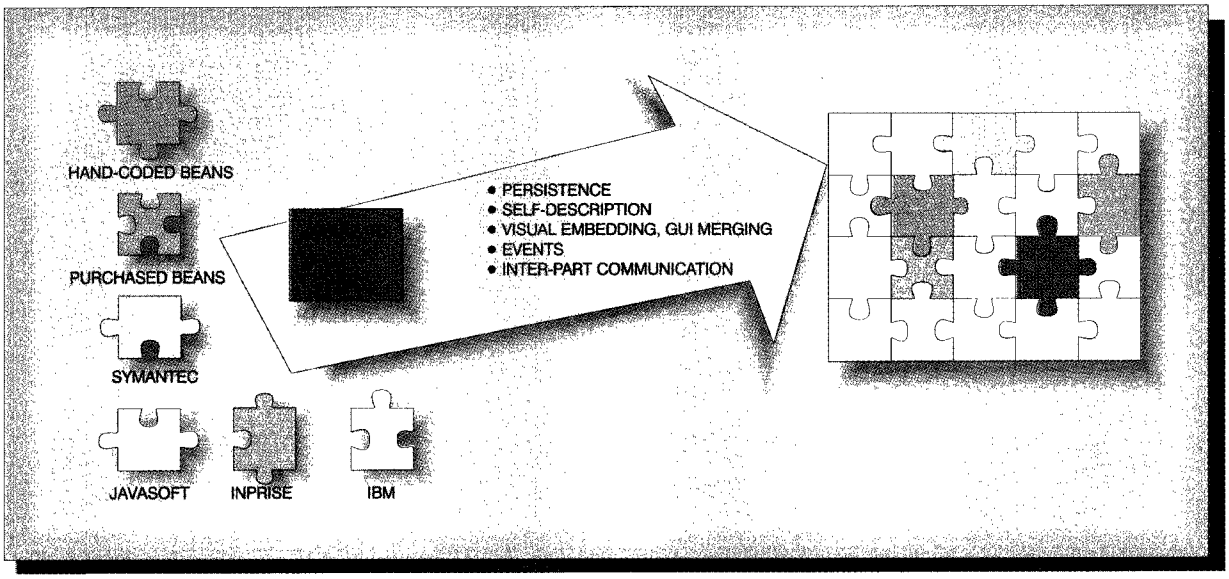
1. Tools for creating and consuming JavaBeans
2. Helper beans for helping the user get at system services and existing data
3. Beans providing client and server business functionality
4. Support for Enterprise JavaBeans

An overview of the IBM tools used for writing JavaBeans is presented later in this paper.

An example of helper beans is the bean extender technology that has been put into VisualAge for Java. Bean extenders provide capabilities that enable tools groups to provide better beans—they are JavaBeans that consume and produce beans in such operations as aggregation, collection, inspection, invocation, disassembly, and behavior modification. Another example is the Taligent Webrunner beans for accessing Internet files and sending and receiving Internet mail.

The JavaBeans that are being shipped as part of the eSuite product from Lotus are an example of beans providing business functionality. eSuite includes productivity components for e-mail, spreadsheets, word processing, charting, data access, presentation graphics, and project scheduling. They come with a communications infrastructure called InfoBus and an infrastructure for a common "look and feel" called

Figure 8 JavaBeans in IBM's strategy



InfoCenter. For more information on eSuite, see the paper by Briggs in this issue.[4]

IBM server platform approach

This section describes how the IBM server platforms are enabling themselves to work with Java.

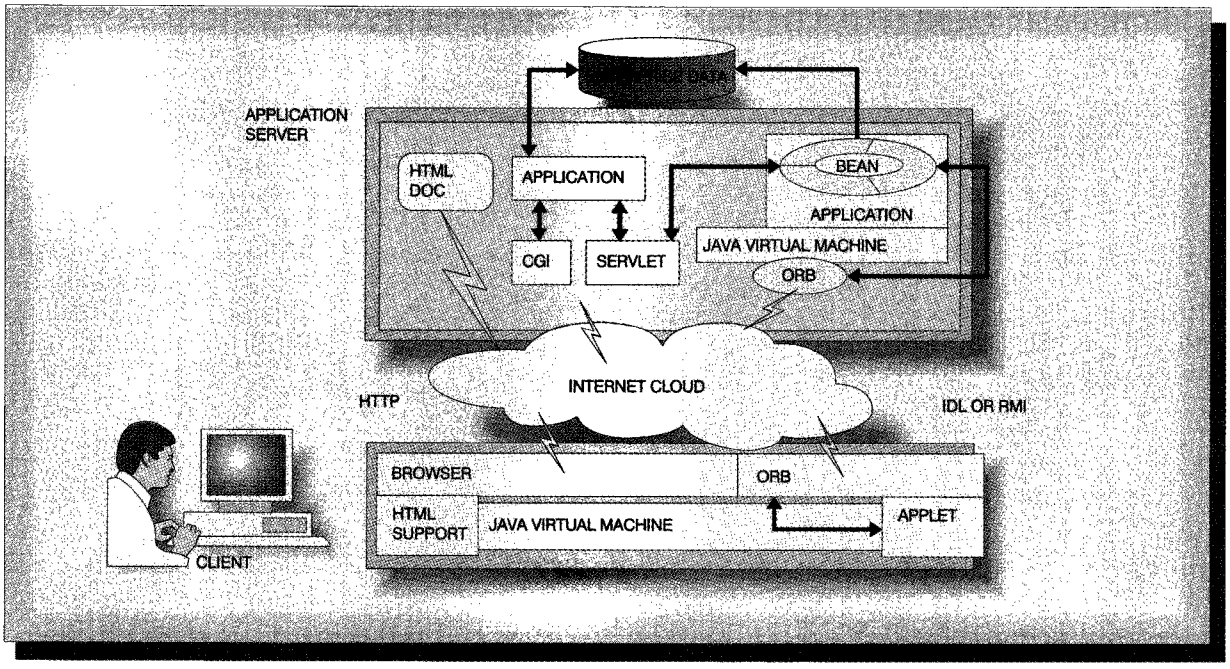
The IBM Centre for Java Technology Development, working in partnership with the appropriate platform development groups, has "ported" the Java reference implementation from JavaSoft to each of the major IBM server platforms (OS/390, AIX, OS/2, OS/400, VM). The centre is now focusing on improving the performance of the Java implementations running on these platforms, so that they will be "best of breed" in their competitive spaces.

In order to obtain best-of-breed performance from the Java Virtual Machine and associated class libraries, it is necessary for the platform organizations to modify certain pieces of the reference-implementation code so as to take advantage of features inherent in their individual platforms. To obtain maximum utilization from the reference implementation and to ensure that there is minimal deviation, each platform organization is working closely with the centre. The starting point for all modifications is the centre's porting of the Java reference implementation of any particular version of the JDK. The programmers responsible for adding Java to each platform review upcoming features of each new JDK version with the centre and decide under the guidance of the centre which pieces will need to be reworked for that platform. To ensure that it is a valid implementation of the JDK, *each platform implementation must pass the appropriate JavaSoft compliance test suite before it is made generally available to customers.*

In order for Java to be a pervasive platform technology in the industry, IBM is cooperating with JavaSoft and other major players in the Java arena to ensure that JavaSoft receives maximum benefit from IBM's experience in the server world and can improve the JDK and associated services based on the results of IBM's experience. Working with the centre, each platform group identifies technology that it has that might help the Java community in general, and IBM works with JavaSoft to make this technology generally available.

The centre has standard performance benchmarks that are run against the platform implementations, so that IBM may make interplatform comparisons and know better how to advise customers when it comes to platform choice.

Figure 9 Java client/server communications



IBM's approach to Java transport and distributed objects

We now present the IBM strategy with respect to client/server communications and distributed objects in the Java environment. We first describe the various ways of distributing information and services in the Java environment and then give some guidelines as to when to use which services.

Client/server communications overview. In this subsection we discuss the major ways in which Java applets or applications running on a client machine can communicate with Java or non-Java applications running on a server.

Figure 9 illustrates the various methods for client/server communications in a Java environment. In the figure, a person at a Java client platform is using the client to communicate with application code on an application server. Note that there are two major styles of communication. In one case, the person is using a Web browser to communicate with the server via the HyperText Transfer Protocol (HTTP). HTTP is the original transport protocol for the World Wide Web. The Web browser has HyperText Markup Language (HTML) support and communicates either via the Web server with an HTML document at the server, or via the Internet Common Gateway Interface (CGI) with an application running on the server. If the application is set up to do so, it may retrieve existing enterprise data from a mainframe computer or data server and return the data to the applet via the HTTP-CGI mechanism. Applications written to work with CGI are often written in a scripting language, such as the Practical Extraction and Reporting Language (PERL) or a UNIX shell script, but such applications may also be written in C or C++.

Figure 9 also illustrates a recent alternative to CGI called the Java servlet. Servlets run in a Java environment on the server and are somewhat analogous to applets running on a client. Like CGI applications, servlets can access and return data via HTTP, but servlets also benefit from the portability and security characteristics of the Java environment. As is illustrated in Figure 9, servlets can access JavaBeans, including Enterprise JavaBeans. The Java servlet development kit is part of JDK 1.2. For more information on servlets, see the paper in this issue by Bayeh.[3]

As the right side of Figure 9 illustrates, the person may also use a Java applet or application on the cli-

ent to provide an object-oriented style of communicating with the application server code. The applet in the client in Figure 9 contains a *proxy*, or *stub*, that is communicating with a remote JavaBeans function on the server via one of two object-oriented APIs: the interface definition language (IDL) of CORBA or the remote method invocation (RMI) of JavaSoft. CORBA IDL uses a protocol standard, the Internet Inter-Orb Protocol (IIOP), whereas RMI uses either the proprietary Java Remote Method Protocol (JRMP) or IIOP. IBM advocates the use of RMI over IIOP when doing distributed programming in Java, whether the target object is written in Java or in another language such as C++. These protocols, which are described in more detail in the next subsection, allow client applets and applications to invoke server functionality as if it were local through use of an object request broker (ORB). This object-oriented style of communication is used between servers, as well as between servers and clients.

In Figure 9, the application server itself might be a mainframe computer (an arrangement called two-tier client/server computing) or it might be communicating with a mainframe (three-tier client/server computing). In either case, the ability to access these data from any machine running the JVM greatly increases its usefulness.

Distributed objects overview. There are three major ways of communicating among distributed objects in a distributed environment:

- RMI API with JRMP or IIOP
- IDL-defined APIs with IIOP
- Distributed Component Object Model (DCOM**)

RMI is a set of Java services that was developed by JavaSoft to allow applications or applets running in a Java environment to invoke services (methods) that are provided by objects running remotely in Java applications or applets. RMI is provided as a service of JDK Version 1.1. RMI could be used for both client-server and server-server communications among Java applications and applets. RMI is a scheme for *pure Java* invocation of remote services.

The IDL API and IIOP are provided by the Common Object Request Broker Architecture, or CORBA. CORBA is a set of object interoperability standards that has been developed over many years by members of the Object Management Group (OMG). JavaSoft has incorporated IDL and IIOP into Version 1.2 of the JDK so as to make them available to Java users.

In addition to IIOP, CORBA has many other services associated with it that have been developed over the years. CORBA provides a relatively heavyweight set of services compared to RMI and is more complex to deal with. Yet, through the use of IDL for defining method interfaces provided by objects, CORBA is language-neutral, supporting distributed objects that are written in many languages, including C, C++, and Java. Thus, incorporation of IDL into Java allows Java applications to seamlessly invoke remote services that are written in a variety of languages, running on a variety of platforms. This complements the standard Java API for accessing local objects written in other languages, the Java Native Interface (JNI).

DCOM is Microsoft's proprietary protocol for facilitating remote invocation of COM objects. COM objects are Microsoft's component objects, analogous to JavaBeans. For more information on COM and DCOM, see Sessions.[7]

In addition to IDL support, JDK 1.2 includes an RMI API that works with an IIOP transport layer. This approach combines the best features of both the Java and the CORBA support for distributed-object computing and is a good model of how Java may work with other CORBA capabilities in the future. This approach is recommended by IBM for distributed Java computing.

The above is of necessity a very high-level discussion. To obtain a greater depth of the details of distributed computing using Java and CORBA, the author recommends the book by Orfali and Harkey as a good starting point.[8]

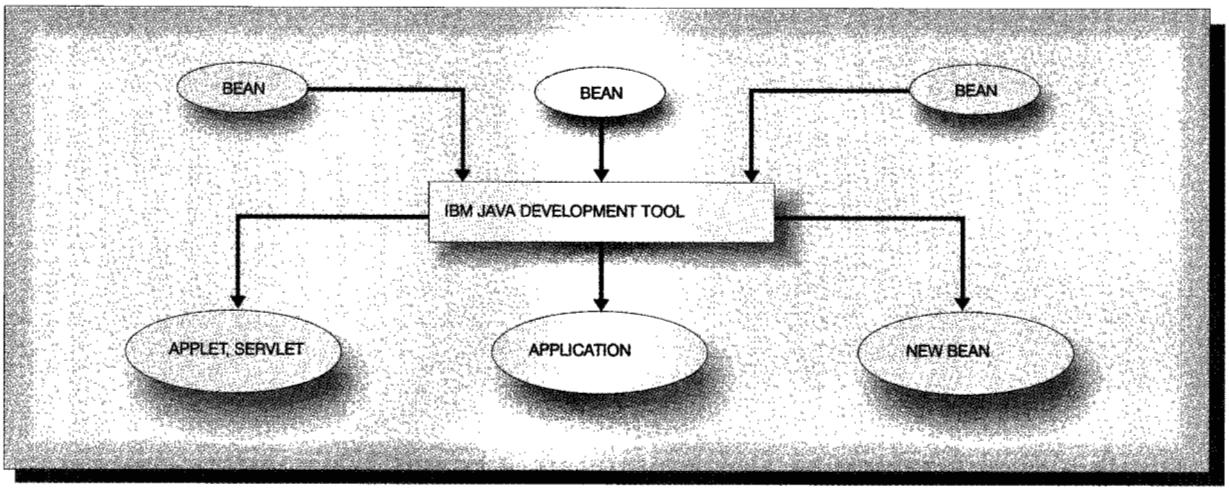
Tools overview

The IBM strategy for Java tools is centered around a new Java-oriented version of the VisualAge integrated development environment.

The overall IBM approach with respect to Java tools is twofold:

1. Provide a robust, enterprise-oriented integrated development environment that will facilitate the development of enterprise applications written in Java and rapid application development using JavaBeans
2. Provide lightweight tools for rapid application development using JavaBeans

Figure 10 JavaBeans consumer and producer



IBM tools enable the customer to use Java to quickly build enterprise solutions. They reduce the complexity of programming in the server environment through the exploitation of standard Java APIs and JavaBeans to access mainframe subsystems and data. They focus heavily on the use of JavaBeans for rapid application development.

IBM tools consume and create JavaBeans components, as illustrated in Figure 10. Here, an IBM development tool is taking as input (or consuming) JavaBeans that may come from a variety of sources—ISVs, customers, or IBM—and producing from these beans and other code Java applets and servlets, Java applications, and new JavaBeans. The tool is thus both a *producer* and a *consumer* of JavaBeans. The JavaBeans it produces can be reused—provided as input to the IBM tool or any other Java-capable rapid application development tool. In this way the full promise of reusable components and rapid application development can be realized.

IBM's VisualAge for Java is the centerpiece of IBM's Java tools strategy. VisualAge for Java consists of a set of integrated development tools for end-to-end development of Java applications and components. VisualAge for Java features a visual application builder that facilitates the visual construction of applications from JavaBeans, as well as an enterprise access builder that automates the connection of Java applications to enterprise data, transactions, and applications. VisualAge for Java also includes a highly integrated set of tools for rapid application development, including tools for managing projects, browsing classes, graphical debugging, and editing Java code. Also included in VisualAge for Java are an incremental Java compiler and a set of JavaBeans components to assist in building applications.

VisualAge for Java is packaged in three versions:

- *Professional*—contains the editor, debugger, browser, and visual application builder
- *Entry*—a free, scaled-down version of the professional version
- *Enterprise*—includes all of the components described in the paragraph immediately above, and facilitates connecting Java clients to existing server data, transactions, and applications

In addition to VisualAge for Java, IBM provides an integrated set of tools called IBM VisualAge e-business that integrates VisualAge for Java with a complete set of tools for building and maintaining Web sites. Besides VisualAge for Java, VisualAge e-business includes the following tools:

- NetObjects Fusion** for automatic website building, design, and publishing
- Lotus BeanMachine for creating Java applets from components without writing code
- Lotus Domino Go Webserver** for hosting robust websites
- DB2 Universal Database* Developer's Addition for

testing and debugging DB2* (DATABASE 2*) applications
- VisualAge Webrunner for Java productivity tools and JavaBeans
- Netscape Navigator** Web browser

VisualAge e-business provides Web developers with one place to go for a complete set of tools for building e-business applications.

For more information on VisualAge for Java Enterprise, see the paper by Chamberland et al. in this issue.[9] For more information on VisualAge in general and VisualAge e-business in particular, see the IBM application development website (http://www.software.ibm.com/ad).

Java security overview

Because Java applets were designed to be downloaded from a remote server and executed on a local client, security has always been a major focus for the architects of Java, and they have designed a high level of security into the Java environment. The Java run-time environment includes a ByteCode Verifier that ensures that compiled code is properly formatted and does not violate certain security restrictions, a ClassLoader that determines how and when Java code is loaded and prevents applets from replacing system level components dynamically, and a Security Manager that enforces restrictions on applet access to client system resources. Java also provides cryptographic and digital signature services, and with JDK 1.2, the earlier robust but somewhat inflexible scheme for controlling access to protected functions based on policy has evolved to a very flexible security mechanism for controlling access to protected functions based on user- or installation-administered security.

Although Java already contains robust security services, more functionality is needed to provide enterprise-level security to Java. More needs to be done in such areas as deploying Java applications securely and running multiple Java applications on a single JVM securely. For a detailed discussion of Java security features and additional security requirements, see the paper by Koved et al. in this issue.[10]

Conclusion

In this paper, we have described IBM's approach to providing best-of-breed Java capabilities for its customers. IBM is taking an organized approach to Java, providing basic Java facilities on all IBM platforms as well as frameworks, components, and tools that make it easier to build robust Java applications. IBM is also making it easier to hook new Java applications into customers' existing applications, subsystems, and data so as to provide the best possible leveraging of existing customer investment in the new world of e-business. The comprehensive set of Java capabilities being provided by IBM will allow its customers to combine the robust reliability, availability, and security characteristics of enterprise services that have been developed over many years with the exciting new Web-based capabilities of e-business, thus giving customers the best of both worlds.

To keep up to date on IBM's ongoing Java development and technical activities, visit the IBM Java website (http://www.ibm.com/java). This site includes sections on news, Java-based applications, developer tools, developer assistance, Java events (trade shows and conferences), the Java community (including user groups and commentary), and education (including white papers, courses, book reviews, and a glossary of Java terms).

Cited references

1. D. Flanagan, *Java in a Nutshell: A Desktop Quick Reference, 2nd Edition*, ISBN 1-56592-262-X, O'Reilly & Associates, Sebastopol, CA (1997).
2. B. S. Rubin, A. R. Christ, and K. A. Bohrer, "Java and the IBM San Francisco Project," *IBM Systems Journal* **37**, No. 3, 365–371 (1998, this issue).
3. E. Bayeh, "The WebSphere Application Server Architecture and Programming Model," *IBM Systems Journal* **37**, No. 3, 336–348 (1998, this issue).
4. B. Briggs, "Lotus eSuite," *IBM Systems Journal* **37**, No. 3, 372–385 (1998, this issue).
5. This figure is taken from an illustration on the Sun Microsystems website http://java.sun.com/marketing/enterprise/enterprise.html.
6. I. F. Brackenbury, D. F. Ferguson, K. D. Gottschalk, and R. A. Storey, "IBM's Enterprise Server for Java," *IBM Systems Journal* **37**, No. 3, 323–335 (1998, this issue).
7. R. Sessions, *COM and DCOM: Microsoft's Vision for Distributed Objects*, Second Edition, ISBN 0-471-24578-X, John Wiley & Sons, Inc., New York (1998).
8. R. Orfali and D. Harkey, *Client/Server Programming with Java and CORBA, 2nd Edition*, ISBN 0-471-24578-X, John Wiley & Sons, Inc., New York (1998).
9. L. A. Chamberland, S. F. Lymer, and A. G. Ryman, "IBM VisualAge for Java," *IBM Systems Journal* **37**, No. 3, 386–408 (1998, this issue).

10. L. Koved, A. J. Nadalin, D. Neal, and T. Lawson, "The Evolution of Java Security," *IBM Systems Journal* **37**, No. 3, 349–364 (1998, this issue).

Accepted for publication April 24, 1998.

Karl D. Gottschalk *IBM Network Computing Software Division, P.O. Box 12195, Research Triangle Park, North Carolina 27709 (electronic mail: karlgott@us.ibm.com).* Mr. Gottschalk is a senior software engineer focusing on IBM's technical strategy for Java. He has been heavily involved in the definition and use of IBM Java tools and components for building and running enterprise applications. Prior to working on Java, he worked for many years on IBM's systems and network management products; he was the chief designer for several releases of IBM's NetView product. Mr. Gottschalk joined IBM in 1968 and has held positions in the areas of program design, program development, program maintenance, and information development. He received a Master of Arts degree in English literature from the University of Mississippi in 1965, a Master of Science in computer science from the University of North Carolina at Chapel Hill in 1976, a Master of Business Administration from Duke University in 1983, and a Master of Arts in liberal studies from Duke University in 1988.

Reprint Order No. G321-5678.