# IBM's Enterprise Server for Java

by I. F. Brackenbury
D. F. Ferguson
K. D. Gottschalk
R. A. Storey

***IBM is exploiting Enterprise JavaBeans™ in a family of compatible Java™ application servers conforming to IBM's Enterprise Server for Java (ESJ) specification. The ESJ provides a common set of dynamic, adaptive system services to meet today's (and tomorrow's) middleware requirements. ESJ will provide a standard programming model and set of services across major server platforms so that implementations of ESJ are differentiated not by function but by quality of service. Finally, ESJ increases productivity by enabling programmers to focus on business logic rather than on infrastructure details. This paper introduces the design of ESJ, including the attributes of the common execution environment, its interaction with other middleware, and its client/server capabilities. It provides an appreciation of the value of ESJ to application developers as a means of achieving cross-platform consistency, lower costs, and faster time to market. It also outlines the features that make ESJ the server technology base for wide-scale reuse through the "write once, run anywhere" promise of Java.***

The Enterprise Server for Java (ESJ) specification is IBM's roadmap for implementing JavaSoft's Java** for the Enterprise[1] initiative, which was announced at the JavaOne developers' conference in April 1997. This initiative consists of a set of standardized application programming interfaces (APIs) for accessing system services and a set of extensions to the JavaBeans** architecture, called Enterprise JavaBeans** (EJB) extensions, for quickly building applications from components that can make use of existing system services. IBM will make the IBM ESJ available across major hardware platforms, thereby bringing to IBM customers the advantages of Java for the Enterprise.

ESJ servers are realizations of IBM's Network Computing Framework (NCF) for e-business,[2] and they build on existing, proven middleware technologies to assure scalability and robustness. ESJ-compliant systems are logical second-tier Java application servers supporting execution of Enterprise JavaBeans, Java servlets, and Java applications. Compliance with the ESJ specification transforms the second-tier application server running Java, discussed in the paper by Gottschalk in this issue,[3] into an ESJ server.

ESJ consists of: a Java Virtual Machine (JVM) and core classes, including Java interface definition language (JIDL) and remote method invocation (RMI); support for Enterprise JavaBeans and a selection of other Java standard extensions; and a common set of IBM extensions called EJB connectors. These, in combination, provide the function required for a broad range of enterprise applications. The Java standard extensions in ESJ are: Java Naming and Directory Interface (JNDI), Java Database Connectivity (JDBC**) Version 2, Java Structured Query Language (SQLJ), and the Java Message Service (JMS).[4] The IBM EJB connectors are JavaBeans that connect applications in an ESJ with third-tier resources such as IBM's Customer Information Control System (CICS*), DATABASE 2* (DB2*), Information Management System (IMS*), MQSeries*, and Lotus Notes** products.




0018-8670/98/$5.00 © 1998 IBM

ESJ supports execution of Java programs written as Enterprise JavaBeans, servlets, or Java applications. Enterprise JavaBeans can also emulate servlets. Enterprise JavaBeans are run under control of an ESJ monitor that has the traditional strengths of transaction monitors in that large numbers of clients can be serviced concurrently through efficient sharing of server resources, and the client sessions and the objects they are accessing can be administered safely and efficiently.

The Common Object Request Broker Architecture (CORBA**) forms the basis of the architecture for distributed object communication for ESJ servers; such servers are also all Web-enabled. For example, every ESJ implementation supports client access from: Java clients using RMI over CORBA's Internet Inter-Orb Protocol (IIOP), C and C++ clients over IIOP, and Web browsers over HyperText Transfer Protocol (HTTP). Each ESJ includes a CORBA object request broker (ORB) and requires a companion Web server, relational database, security service, and naming service; these elements are pluggable to allow customer choice.[5]

In time, ESJ-compliant Web application servers will be hosted on most important operating systems and middleware environments, including transaction processing monitors and database stored procedures.

The primary goals in implementations of ESJ are fourfold:

1. The *cross-platform goal* is to reduce the cost of developing and using enterprise applications that require access to resources on multiple different hardware, operating system, or middleware platforms.
2. The *simplify application development goal* is to halve the effort in developing second-tier enterprise applications by absorbing the effort of managing transactions, security, resource-sharing, and administration into the infrastructure of ESJ.
3. The *widespread software component reuse goal* is to lay the foundations for wide reuse of software components—the Java "write once, run anywhere" promise—initially by providing the Java Virtual Machine environment together with simple ways to partition the portable and nonportable components of an application, reducing the porting costs.
4. The *integration goal* is to enable easy integration of enterprise Java applications into existing heterogeneous business middleware.

This high-level view of IBM's ESJ specification is intended for computing professionals who are familiar with Java technology and have an understanding of the requirements for creating and deploying server software for business use. For more information on the basics of Java, see Flanagan.[6]

In the remainder of this paper, we unpeel the ESJ specification in five diagrams, beginning with the abstract level of a generic three-tier system for programs written in Java, and finishing with a summary of all key elements in Release 1 of ESJ.

In the diagrams throughout the paper, the light red color (no shading) denotes components with industry-standard interfaces, whereas blue (cross-hatching) denotes components with IBM-defined interfaces, dark red (horizontal stripes) denotes components with enterprise APIs defined as part of JavaSoft's Java for the Enterprise initiative, and green (vertical stripes) denotes pluggable components.

## The three-tier model

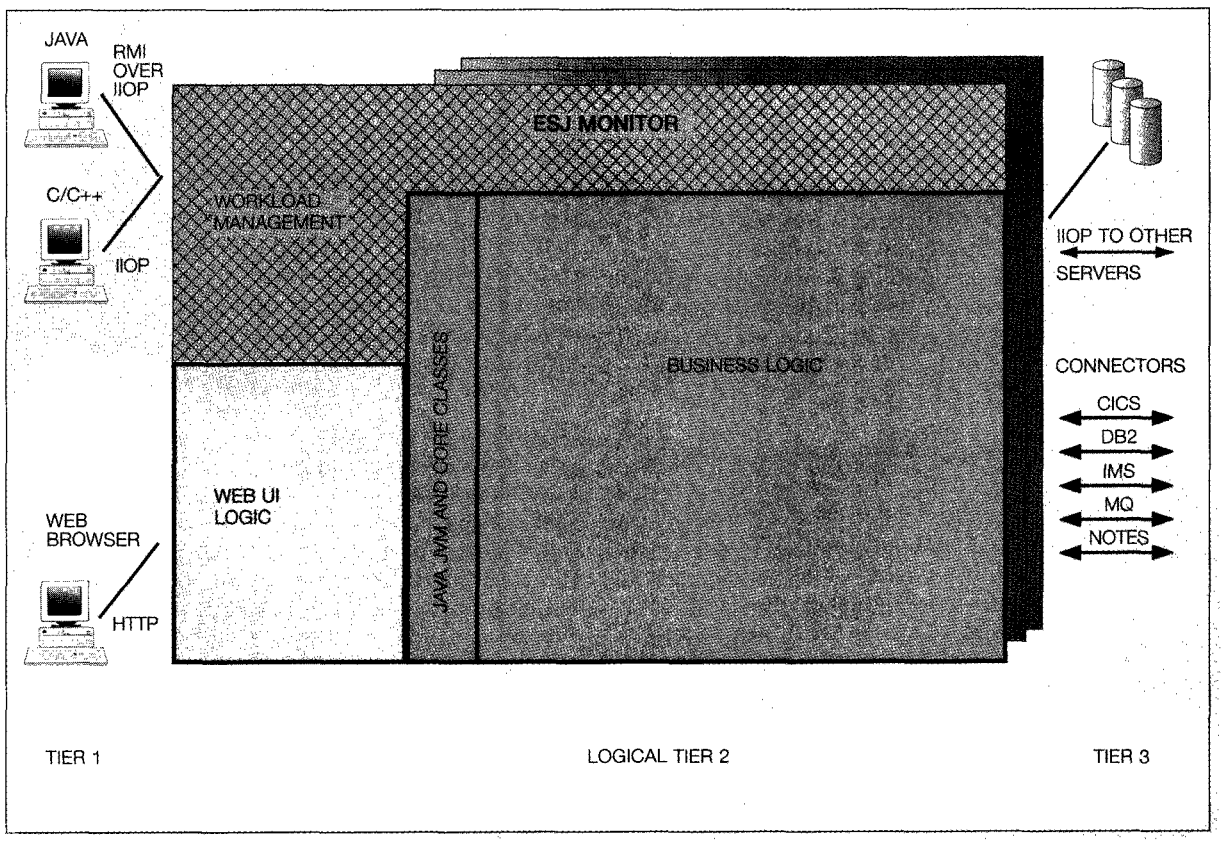
The ESJ specification defines a standard execution environment for the second tier of a three-tier enterprise application. It uses CORBA and Internet client technologies and deploys business logic written as Java programs.

ESJ exploits the JavaSoft Enterprise JavaBeans (EJB) APIs as the foundation for a transaction processing monitor that optimizes resources and provides the transactional, security, and administrative functions needed to manage large numbers of concurrent client accesses with limited server resources. Enterprise JavaBeans extend the JavaBeans component architecture to the enterprise level, allowing enterprise applications to be built from reusable JavaBeans components that can participate in robust system services, such as transaction services and database services, that are already available on most major server platforms. For more information, see Matena and Hapner.[7]

Figure 1 illustrates the three-tier model assumed by ESJ. This model includes a client tier (Tier 1), a Java application server tier (Tier 2), and a data and resource manager subsystem tier (Tier 3).

**Tier 1—The client.** All Enterprise JavaBeans servers support a minimum of three types of Tier-1 client. A client in this context is end-user equipment

**Figure 1 The three-tier model**




such as a workstation or personal computer; a network computer, laptop, or smaller networked device; or a program in another server, acting like a client. The three types of client are:

1. Clients that contain a Java environment. These clients can communicate with an ESJ server using any of the following three techniques:
   - By using GET/POST requests over HTTP to the Web server (HTTP daemon, or HTTPD) plugged into every ESJ
   - By issuing Java RMI calls over CORBA IIOP to the ORB. There is an ORB in every ESJ server. RMI over IIOP will be a standard part of every Java Development Kit (JDK**); it is based on the 1998 Java-to-IDL mapping recently standardized by the Object Management Group (OMG).[8] It includes the new "objects by value" IIOP function.
   - By creating standard CORBA method calls, using Java IDL to develop the appropriate stubs and skeletons, which will then communicate over IIOP
2. Clients supporting CORBA IIOP, including programs written in C and C++, using an IDL compiler and tools to generate communications that conform to the 1998 Java-to-IDL mapping mentioned in the third technique of 1 above
3. Clients that include an industry standard Web browser and can therefore use uniform resource locators (URLs) to communicate with the Web server embedded in every ESJ

With guaranteed support for these three types of client, the ESJ is well-suited for applications accessed from the Internet and from intranets and extranets; the IIOP support recognizes the growing importance

of CORBA as the distributed object infrastructure used by networked enterprises.

**Tier 2—The application server.** To become an ESJ, an application server must support three major modules:

1. A user-interface (UI) logic module that is the increasingly popular integration point for Internet UI technologies, combining information content, multimedia, and the results of business computation. This module contains a pluggable Web server, including an HTTPD and the facilities to create dynamic Web pages. It also serves up Java applets for downloading on demand and execution in the client.

2. An ESJ monitor where the emphasis is on high-performance dispatching of business logic components in response to verified client requests, invoking Enterprise JavaBeans methods, servlets, or Java applications. This module includes the ORB and distributes work to one or more concurrently executing Java Virtual Machines. It provides cluster management and supports multiprocessing where applicable.

   The ESJ monitor also contains the Java Virtual Machine (JVM) and its core classes. This JVM executes all Java bytecode programs in the ESJ server. The integration of ESJ with each operating system and hardware platform is largely a customization of this module. In addition, specific facilities to exploit symmetrical multiprocessing and system clustering, and to enhance scalability, are built into the appropriate JVMs.

3. A business logic module that provides the execution environments for Enterprise JavaBeans, servlets, and Java applications. The ESJ monitor provides resource sharing, transaction management, object life-cycle management, security, systems management, and systems administration for the business applications that reside here.

**Tier 3—Data and resource manager subsystems.** The industry has converged on the logical three-tier configuration for creating enterprise applications. The Internet has brought explosive growth in connectivity and much reduced prices for related software technologies. Customers now want to use industry and Internet standard componentry such as CORBA and Java; they also need flexibility in applying security and administration policy distributed across the three tiers; and they have to plan for significant acceleration in the pace of application development.

The third tier consists of the data stores and resource manager subsystems holding the durable business data that are the bedrock of enterprise computing. It is still subject to the most rigorous security and the most careful and deliberate evolution. ESJ recognizes that these strategic resources need to be treated differently. This huge existing corporate asset base is at the same time the most potent source of new application value and the least easily altered information system in the enterprise.

ESJ provides Enterprise JavaBeans connectors to interoperate with the third tier. Every ESJ implementation has a standard way of accessing the following IBM subsystems: CICS/ESA** (Customer Information Control System/Enterprise Systems Architecture), DB2 on OS/390* (Operating System/390), IMS, MQSeries, and Lotus Notes. Application developers are assured that all IBM second-tier servers provide access to corporate server data and applications. Enterprise JavaBeans using connectors are portable across ESJ implementations.

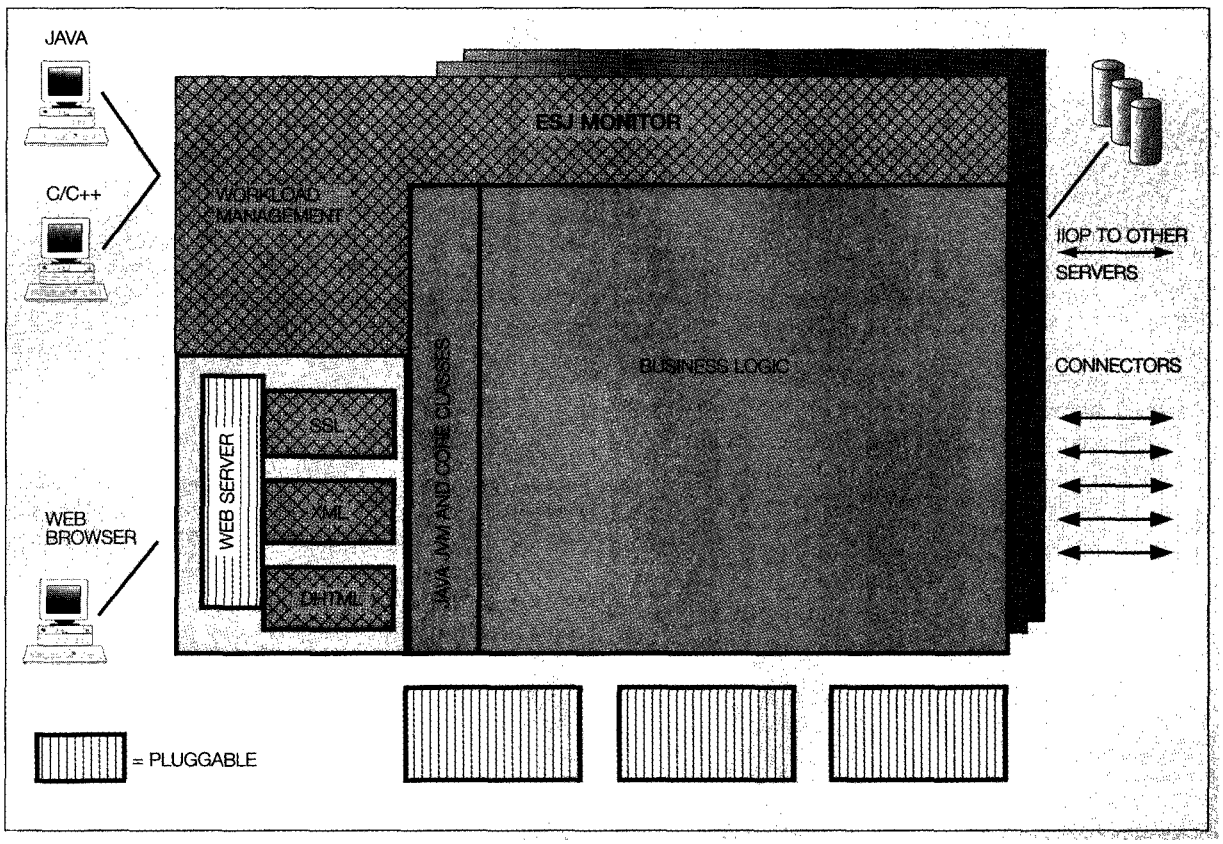
Connectors can be used to access resources on the same network node or on different nodes. ESJ is intended for both physical three-tier implementation and logical three-tier systems, where the second and third tiers reside on a single processor complex.

An ESJ application using connectors can coordinate transactions spanning multiple remote "back-end" resource managers. The transaction, security, and systems management services of the ESJ monitor use CORBA and proprietary interfaces to ensure proper interoperation.

As Java standards emerge for interoperation, these APIs are included in ESJ. For example, data access using JDBC and SQLJ provides even broader portability, and JMS is a Java standard that supports reliable messaging and queuing.

RMI over IIOP is the ESJ standard way of interworking with other instances of ESJ servers. Context for transactions, security, systems management, and remote debug automatically flows with these remote method calls between ESJ servers.

**Figure 2 The UI logic module**




In the next several sections of this paper, we consider the three major modules of the Java application server (Tier 2).

## The user interface logic module

The user interface (UI) logic module of ESJ exploits the rapidly evolving capabilities being added to industry Web servers, mostly in support of producing HyperText Markup Language (HTML) pages dynamically, through a combination of HTML static boilerplate text and multimedia with algorithmically determined data computed by server-side scripts or servlets. Figure 2 illustrates the UI logic module.

**The Web server.** Every ESJ implementation requires a Web server such as IBM's Domino Go**, Microsoft's Internet Information Server (IIS**), Netscape's SuiteSpot**, or the Apache Web server. The Web server must support HTTP 1.1 and is required to provide independent storage, administration, and serving of HTML pages, multimedia files, and Java applets.

Web servers that, like Domino Go, use ESJ interfaces for tight integration with the UI logic module can have systems administration integrated with the rest of the ESJ server. In addition, multiple HTTP requests and IIOP sessions can be carried on the same EJS-client connection. Otherwise, the ESJ specification is neutral on choice of Web server.

**Secure sockets layer.** The UI logic module provides an integrated secure sockets layer (SSL) service for both HTTP and IIOP connections. If the Web server also provides SSL, it is an installation choice whether to use the integrated service or the Web server. The integrated SSL service uses common registration and administration tools and is enabled for international use.

**Dynamic HTML and server-side scripting.** Dynamic HTML (DHTML) and server-side scripting using JavaServer** pages (JSP) are standard capabilities of ESJ servers. DHTML provides support for use of Server-side HyperText Markup Language (SHTML) pages that contain embedded programming fragments, including function definitions and event scripts. A choice of languages is provided: Java, JavaScript** (ECMA 262 standard), and NetRexx.

The SHTML pages are compiled when they are installed on the UI logic module. The resulting Java bytecode programs are executed upon demand, and give high-performance, flexible, server-side scripting.

Typically, the actual script embedded within an SHTML page is short and uncomplicated. Any significant processing is provided by a servlet or Enterprise JavaBeans invoked from the script.

JSP is a new technology supporting simple and powerful scripting that allows dynamic HTML generation on the server side. For more information on JSP, see Bayeh.[2]

**Extensible Markup Language support.** Extensible Markup Language (XML) is a standard, easy way to describe and exchange data on the World Wide Web.[9] ESJ provides full server-side XML support, including support for the W3C Document Object Model (DOM).

## The ESJ monitor

Among the marks of a high-performance, scalable server are efficient management of client sessions, avoiding session creation and termination overheads where possible, and reusing session resources quickly. Client/server interactions are typically bursty, with relatively large periods of inactivity interspersed by flurries of activity. The server resources pinned down during the idle time are effectively wasted, so the ESJ monitor acts to reduce these to the absolute minimum.

The ESJ monitor implements the Enterprise JavaBeans APIs. It supports the Enterprise JavaBeans life cycle for both session beans and entity beans, and manages the Enterprise JavaBeans containers. The ESJ monitor contains, or invokes the services of, a transaction manager and a security manager. Much of its function is support for optimizing use of resources against large and erratically varying numbers of client requests, and in providing for semiautomated administration of the EJB run time.

The Java Transaction Service (JTS) is a Java form of the CORBA Object Transaction Service (OTS). The JTS interface (or more likely a subset of it as defined by EJB) may be used by some ESJ applications to delimit transaction boundaries and specify transaction disposition. The actual transaction services that underpin this interface will be provided either by a 100 percent Java implementation, or by mapping the interface directly onto the existing transaction control services that will be present in the underlying middleware on which an ESJ is based.

Figure 3 illustrates the major components of the ESJ monitor.

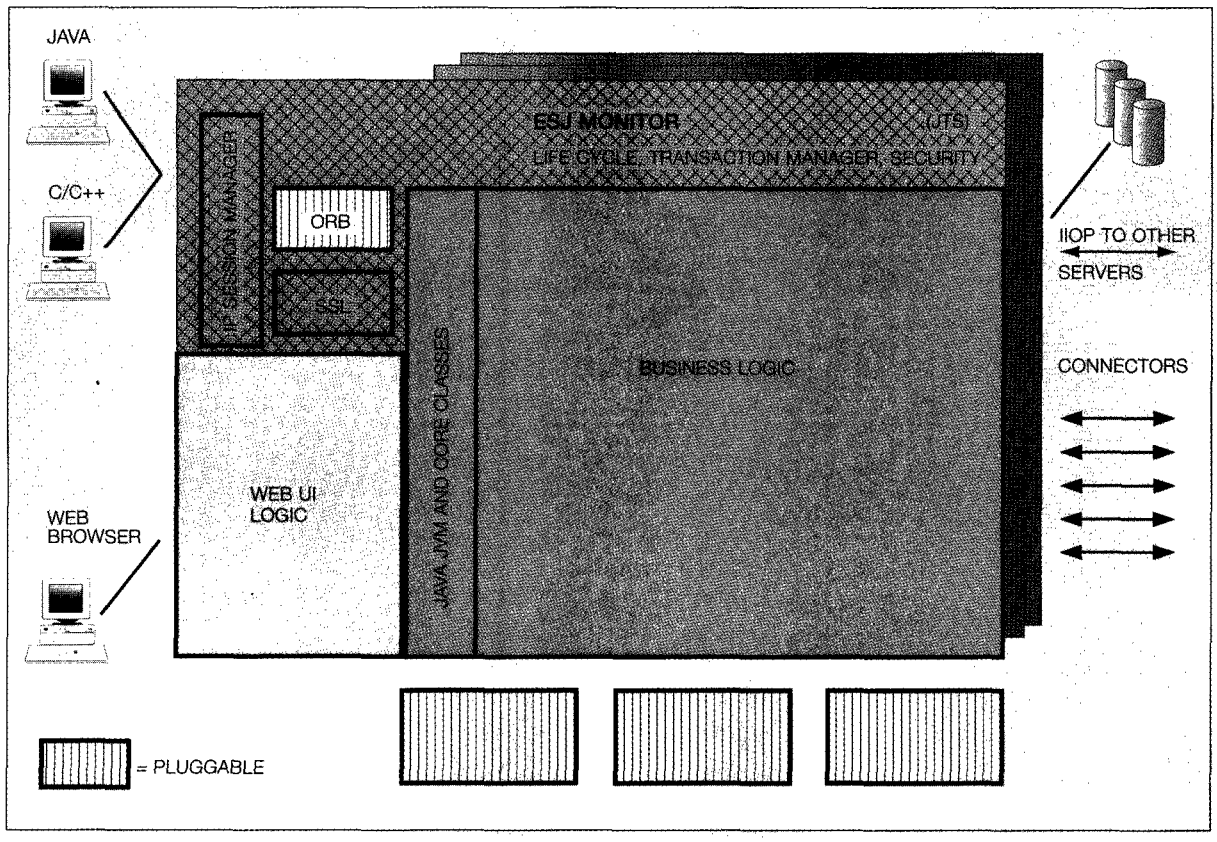
**The IP session manager.** The Internet Protocol (IP) session manager administers and optimizes reuse of client sessions over Transmission Control Protocol/Internet Protocol (TCP/IP) links. It contains instrumentation and calibration software for manual and automatic configuration for optimal performance; the administration tools are integrated with the rest of the ESJ.

**Workload balancing.** In most large server complexes the workload is managed at two levels. The outermost level selects a server or server cluster for each incoming client request. The inner level operates within an ESJ to optimize resource use and maintain responsiveness. The ESJ specification addresses the latter.

Workload balancing in the ESJ monitor provides cluster management for high availability and added performance. It constantly monitors active JVMs and independent units of work inside each JVM, so as to provide dynamic, adaptive system services. On high-end configurations, the workload balancing not only schedules new work to the most appropriate JVM, but it can also suspend and move work between JVMs.

**The JVM and core classes.** The ESJ specification requires JDK 1.2 security, RMI over IIOP, Java IDL, and other function first made available in JDK Release 1.2 in 1998. It benefits from, but does not require, JVM changes that provide for resource sharing in and among concurrently executing JVMs, and changes to garbage collection and object dispatch that make it more suited to server workloads. JVMs optimized for use with ESJ also include system management hooks

**Figure 3 The ESJ monitor**




for determining activity levels and resource consumption.

The JDK security required for ESJ includes a policy-based security model that grants code access to only the set of resources (e.g., files, hosts, ports) that the customer has specified in the appropriate security policy.

**The ORB.** ESJ implementations contain an ORB that supports use of CORBA IIOP, including implicit flows for transaction and security context. Support is mandatory for the IDL defined in the Java-to-IDL mapping; this is the only mapping that is assured for all ESJ implementations. The ORB accepts client calls from outside the ESJ and call-backs to clients from programs in the ESJ; it also arbitrates calls between Enterprise JavaBeans and other programs accessible using RMI over IIOP, such as servlets, Java applications, and Enterprise JavaBeans in other ESJ servers.

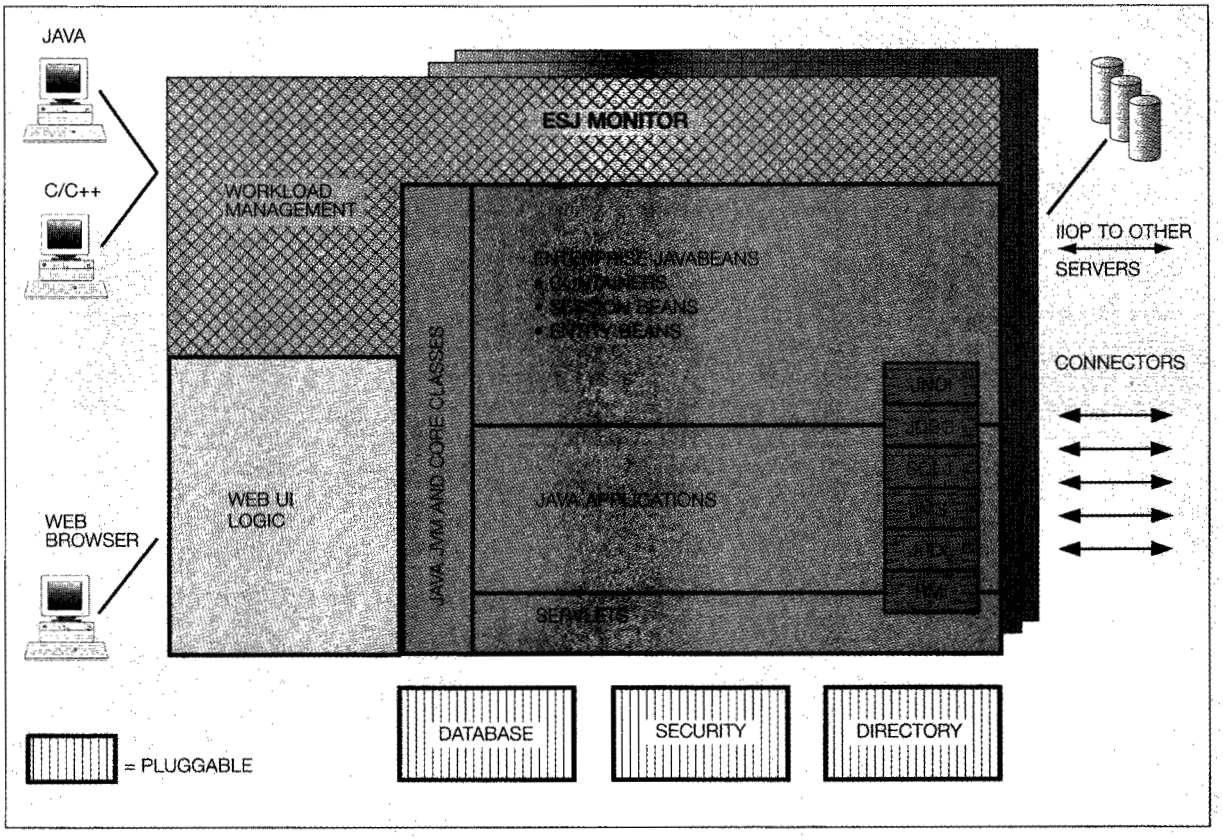
High-end, scalable ESJ implementations include distributed ORBs that are able to coordinate their activities across multiple active JVMs running on one or more processors and across members of a failover cluster.

## The business logic module

The business logic module manages the life cycle of programs in, and provides middleware services for, three distinct execution environments for Java bytecode programs. The business logic module, illustrated in Figure 4, provides the three environments for Java code running in the server: Enterprise JavaBeans, Java servlets, and Java EJB extensions for Java applications.

**ESJ environments for Java code.** The newest environment, Enterprise JavaBeans, is designed specifically for high-volume transactional execution of JavaBeans in the enterprise. Enterprise JavaBeans

**Figure 4 Business logic module**




are, in essence, JavaBeans extended for use in servers. The life cycle of Enterprise JavaBeans and the constraints placed on a bean's use of Java are designed to permit rigorous control of the run time.

The second environment is for servlets. Despite the rapid and recent development of Java technologies, there is already a legacy of Java servlet code that represents valuable investment. So, even though Enterprise JavaBeans provides a superior environment for executing Java scripts (the Java equivalents of CGI-BIN [Common Gateway Interface as stored in a UNIX** directory] scripts), the ESJ specification also supports the servlet environment in a first-class way.

Finally, the Java applications environment is for Java applications and applications that are combinations of Java and native C code, which can be used to provide additional system function. Such function would be disallowed under the stricter rules of Enterprise JavaBeans, or would not fit naturally into the "beans and containers" constructs of Enterprise JavaBeans. The Java programs that run in this environment are Java applications, with free access to all the capabilities of the standard Java APIs and virtual machine. (Of course, if the Java applications make use of non-Java code via such techniques as the Java Native Interface [JNI], they lose the "write once, run anywhere" capability. The programmer must decide whether this trade-off is worthwhile).

A single ESJ monitor manages the three execution environments: Enterprise JavaBeans, servlets, and Java applications, thus allowing efficient, integrated administration across the three environments. Even so, it cannot provide servlets with the scalability and robustness designed into Enterprise JavaBeans, so programmers are encouraged to write new servlet functions as Enterprise JavaBeans. The business

logic module includes helper classes to make this task easier.

**The extended development kit.** The extended development kit (EDK) is the set of JavaSoft standard extensions that are assured to be in every implementation of ESJ. The EDK APIs include the JVM and core class APIs. The EDK standard extensions are:

- JNDI: Java Naming and Directory Interface. This standard API is mapped to a naming or directory service such as the Lightweight Directory Access Protocol (LDAP) or CORBA Common Object Services (COS) naming service.
- JDBC Version 2: Java Database Connectivity. This is a way of accessing relational databases and issuing dynamic SQL (Structured Query Language) requests. In its basic form it is part of the core classes that come with the JVM. However, the ESJ servers can exploit the extended version, which includes XA interfaces, so that the databases accessed by Enterprise JavaBeans can be used automatically within a distributed transaction.
- SQLJ: Java SQL. A high-performance static SQL API, this API is preferred for serious high-performance access to existing relational databases such as IBM's DB2 Universal Database* (UDB).
- JMS: Java Message Service. This JavaSoft standard API provides access to a messaging and a reliable queue service. For ESJ the JMS will be mapped onto the API for IBM's MQSeries.

## The Enterprise JavaBeans run time

The Enterprise JavaBeans run time is the carefully controlled environment designed to support high numbers of concurrent object activations, securely and robustly. The robustness comes from the ESJ monitor, which catches misbehaving Enterprise JavaBeans, avoiding denial of service (for example, a bean that goes into a never-ending loop), and ensuring that the failure of one object does not impact the running of the entire server.

Enterprise JavaBeans are objects that come in two kinds: *session beans*, which are software representations of activities or tasks in a business process, and *entity beans*, which represent things such as employees, vehicles, pigs, diamonds, or hospital beds. A client uses session beans to obtain a service. The life cycle is: a private instance of the service is created, the service is used, and it is discarded. Contrast that with the life cycle of an entity bean. Here the bean is created, kept around for a long time, and, maybe, years later, destroyed.

Sometimes clients create entity beans, and sometimes they destroy them, but usually what the client does is to locate one of these beans, call some of its methods, and then let it go. Often those methods change the state of the bean—the data that make that bean unique. Entity beans are durable; once changed they stay changed, and next time a client accesses that entity bean, the changed state will be seen. EJB uses a transaction manager to ensure that the entity objects change in a predictable way, even if many clients want to access them at the same time. The data values that are the defining state of an entity bean are stored in a database or object store, and it is usually that database which is used to arbitrate the sharing among multiple clients, so entity objects do not receive haphazard partial updates.

Session beans, in contrast, execute on behalf of a single client. Session beans are private resources used only by the client running them and are usually relatively short-lived, with no guarantee of surviving system crashes and restarts (nonpersistent).

The EJB run time consists of EJB *containers*. A container is the principal organizing factor for Enterprise JavaBeans. All beans live inside containers. The user chooses to create different containers in order to group beans in a way convenient to the application; and the user can nest containers so that all the assets of an application, even if spread across multiple containers, can be linked back to a super container that effectively holds the entire application.

The container hierarchy within an ESJ server comprises containers that contain other containers, and containers that contain beans. Beans only live in the leaf nodes of the container tree. Containers have bean *factories* and *finders* that are used to create beans, and to find beans that were "prepared earlier."

New beans are created from a *prototype*. Each different type of bean has its own prototype that is associated with all containers permitted to create instances of that bean type. Because, in the real world, most interesting beans will be "beanified" versions of rows of data that already exist in a relational database, many beans are created as part of creating their container—the container comes full of entity beans.

We offer one final note on entity object persistence. EJB provides two ways of storing the defining state for an entity bean. Either the bean contains Java code to store itself and reinstate itself when needed, or that is done by the ESJ monitor through what is called "container-managed persistence."

A more complete description of Enterprise JavaBeans would describe the full development, deployment, and execution life of a bean. Extensions to the development tools for building JavaBeans will assist in the creation of Enterprise JavaBeans, and special tools are used to deploy new prototypes—for importing a new bean type into the server. Here perhaps the most important characteristic of Enterprise JavaBeans is that about half the effort of producing Java enterprise applications, and a very similar proportion of C and C++ programs, is in the nitty-gritty development of the code to make the application behave transactionally, and to integrate it with the security and system management services on each platform. Enterprise JavaBeans performs most of that chore. The required transactional and security attributes are declared as deployment descriptors, not programmed into the application. That saves work, and also makes the individual beans more likely to be reusable.

**The Java application run-time environment supports execution of Java applications.**

The EJB run time is implemented by an underlying server subsystem such as IBM's Component Broker, CICS/ESA, TXSeries*, and WebSphere products. The underlying subsystem provides services for monitoring, security, reliability and serviceability, etc. EJB run-time implementations by various subsystems will provide the same set of services but will be differentiated on the basis of cost and class of service.

Most transactional attributes of Enterprise JavaBeans are described using declarative markers, stating, for example, whether all methods in a bean need to be executed under control of a transaction, or whether each time a method is called on a bean, a new transaction should be started. There are some methods for describing where transactions begin and end procedurally; these can be used when a bean is acting as a nontransactional client obtaining services from other beans. All procedural transaction control methods in ESJ are copied precisely from a subset of the JTS API definitions.

### The Java application run time (EJB extensions)

The Java application run-time environment supports execution of Java applications. Unlike the Enterprise JavaBeans environment, no constraints are placed on what features can be used from the basic JVM and core classes. This freedom, which is necessary when the Java code is extending the middleware function of the ESJ, perhaps using JNI access to C code, comes with a downside. The ESJ monitor knows very little about what these Java applications are doing and, hence, is less able to detect misbehavior or to protect the server from harm. This is particularly true of programs that call C code; in this case, the ESJ monitor has very little likelihood of knowing what is happening outside the Java environments.

Helper Java classes are provided to make it easy to have Java application programs appear in the Enterprise JavaBeans environment as if they were Enterprise JavaBeans. In this way the Enterprise JavaBeans programs can be completely portable, dependent only on functionally equivalent EJB extensions underpinning the same Enterprise JavaBeans interfaces on each target ESJ server.

IBM supplies some EJB extension programs that are part of the ESJ standard. These programs are the Enterprise JavaBeans connectors for accessing the following:

- CICS/ESA and distributed CICS applications and data
- DB2 UDB systems, where IBM's proprietary APIs are the appropriate way to access the database. The JDBC and SQLJ APIs are recommended for broader portability.
- IMS transactions
- MQSeries servers, using the IBM message queuing proprietary interfaces. The JMS APIs are recommended for broader portability.
- Lotus Notes servers

Other Java application run-time programs can be installed on ESJ systems, for example, a loan application providing server-side administration and sup-

port for network computers, or a DirectTalk* telephony server.

### The servlet run time

The servlet run time provides the JavaSoft APIs for the Java servlet environment, including the servlet life cycle: *init*, *service*, *destroy*. Servlets can be preloaded, so that when a client request comes in, a servlet is loaded and waiting to act on it.[10]

Servlets send and receive most of their data through output and input streams. These streams are supplied each time a servlet is invoked using the *service* callback. A popular specialization of servlets provides function designed to make it easy to read parameters from a URL and send HTTP output in response to that URL request.

Enterprise JavaBeans can emulate the function of most servlets using classes provided by extensions to the Enterprise JavaBeans environment. So, if the servlet logic can stay within the constraints imposed by the Enterprise JavaBeans environment, servlets can, in fact, be built as Enterprise JavaBeans. The advantage is that the servlet programs then benefit from the scalability, robustness, and administrability of Enterprise JavaBeans.

### The pluggable corequisite servers

An ESJ server has dependencies on five pluggable services or servers:

1. A Web server; the default here is to use IBM's Domino Go.
2. A database; the default here is IBM's DB2 UDB.
3. A security service; usually the platform has a security service, and the Java security APIs have been mapped to that, but ESJ does not legislate a specific security service. It does require an implementation of Java security and SSL. The permanent repository of security identity can be on a remote server.
4. A naming and directory service; this service could be on a remote server. The JNDI APIs are used by ESJ and by application code; this service is the implementation underpinning the JNDI APIs. Implementations include, but are not limited to, LDAP and COS naming.
5. CORBA ORB; the ESJ service will supply a default ORB that matches the scalability of the platform used.

### Bringing it all together: a summation

Figure 5 illustrates all of the modules that turn a Tier-2 Java application server into an IBM ESJ.

In Figure 5, the UI logic module comprises a pluggable Web server; server-side scripting support, and XML support for production of dynamic HTML.

The ESJ monitor consists of the IP session manager, Java JVM and core classes, an ORB, and the ESJ monitor functionality required to support the three execution environments of the business logic module.

The business logic module comprises the JavaSoft standard extensions selected for Enterprise JDK Release 1, and three execution environments: Enterprise JavaBeans (Enterprise JavaBeans in their containers), Java applications (EJB extensions), and servlet run time.

The extended development kit consists of the following APIs:

- Access to the Java JVM and its core classes at level 1.2 or higher; the JVM and core classes are part of the ESJ monitor.
- Java IDL and RMI over IIOP with JNDI Release 1, for naming services
- JDBC, including XA enhancements for transactional JDBC
- SQLJ, static SQL
- JMS for messaging and reliable queues
- EJB, both mandatory (session) and optional (entity) objects

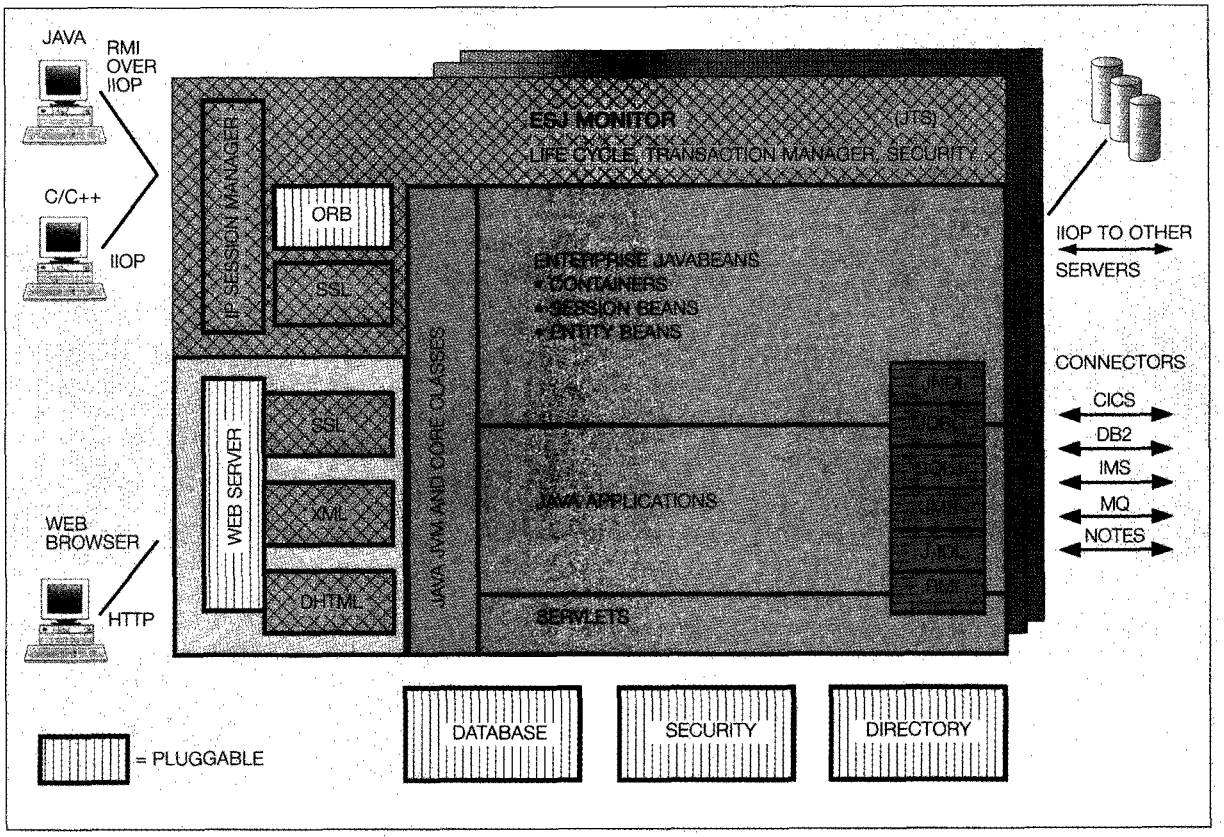
In addition to the ORB, which is a standard but pluggable component of every ESJ, another four pluggable corequisite servers are required by ESJ, and these are not themselves included in the ESJ specification:

- A Web server, such as Domino Go
- A relational database, such as DB2 UDB
- A security service, either local or accessible remotely by the ESJ
- A naming and directory service, either on a local server or remotely accessible

Clients can connect to ESJ servers if they use any of the following:

- RMI over IIOP, for Java clients
- IDL language-neutral access over IIOP, including C and C++ client access

**Figure 5 The complete picture**




- HTTP, for accessing HTML, SHTML, applets, servlets, and Enterprise JavaBeans emulating servlets

Connections to Tier-3 services are via Enterprise JavaBeans connectors, including connectors for:

- CICS/ESA (External CICS Interface), distributed CICS (External Call Interface and External Presentation Interface)
- DB2 and UDB (Enterprise JavaBeans implementations of JDBC and SQLJ access)
- IMS
- MQSeries (Enterprise JavaBeans implementation of JMS reliable queues)
- Lotus Notes

**Candidate IBM implementations of ESJ.** To put the ESJ IBM standard in perspective, ESJ implementations are being considered for the following platforms and IBM enterprise middleware:

- AIX* (Advanced Interactive Executive*), OS/400*, OS/390, Windows NT**, and selected UNIX operating systems
- CICS/ESA, TXSeries/AIX, TXSeries/NT, and TXSeries/UNIX
- Component Broker
- DB2 UDB stored procedures
- MQSeries
- IMS
- WebSphere Application Server

Implementations will range from entry level to advanced. By providing this assured set of APIs and functions on a wide range of host operating systems, hardware, and middleware platforms, IBM addresses the customer and developer need for:

- Solid cross-platform interoperability and portability of code and skills

- Dramatic reduction in time and effort to develop new enterprise applications
- Foundation for future widespread reuse of component enterprise software
- Scalability to meet the broadest possible range of business requirements

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Sun Microsystems, Inc., Lotus Development Corporation, Object Management Group, Microsoft Corporation, Netscape Communications Corporation, or The Open Group.

## References and notes


1. A brief description of JavaSoft's Java for the Enterprise is given in Reference 3 and is also available on the Java for the Enterprise website at http://java.sun.com/enterprise.
2. E. Bayeh, "The WebSphere Application Server Architecture and Programming Model," *IBM Systems Journal* **37**, No. 3, 336–348 (1998, this issue).
3. K. D. Gottschalk, "Technical Overview of IBM's Java Initiatives," *IBM Systems Journal* **37**, No. 3, 308–322 (1998, this issue).
4. For a brief description of these APIs and of JavaSoft's Java for the Enterprise initiative, see Reference 3. See also the Java for the Enterprise website at http://java.sun.com/enterprise.
5. CORBA is a set of distributed computing specifications put out by the Object Management Group (OMG). For a good discussion of CORBA and its relationship to Java, see R. Orfali and D. Harkey, *Client/Server Programming with Java and CORBA*, ISBN 0-471-16351-1, John Wiley & Sons, Inc., New York (1998).
6. D. Flanagan, *Java in a Nutshell: A Desktop Quick Reference, 2nd Edition,* ISBN 1-56592-262-X, O'Reilly & Associates, Sebastopol, CA (1997).
7. V. Matena and M. Hapner, *Enterprise JavaBeans*, Specification Version 1.0, Sun Microsystems, Inc., Palo Alto, CA (1997). Available from the JavaSoft website at http://java.sun.com/products/ejb/docs.html.
8. See the OMG Java to IDL Request for Proposal, OMG Document ORBOS/97-03-08, and submissions to this RFP, available from the OMG website at http://www.omg.org/library.
9. T. Bray, J. Paoli, and C. M. Sperberg-McQueen, *Extensible Markup Language (XML) 1.0* (1998). Available as a Recommendation from the World Wide Web Consortium website at http://www.w3.org.
10. For a good discussion of Java servlets, see the white paper titled *The Java Servlet API*, available from the Sun Java website at http://java.sun.com/marketing/collateral/servlets.html. See also Reference 2.


*Accepted for publication April 27, 1998.*

**Ian F. Brackenbury** *IBM United Kingdom Laboratories Ltd., Hursley Park, Winchester, Hants SO21 2JN, United Kingdom (electronic mail: Ian.Brackenbury@acm.org).* Mr. Brackenbury is Chief Technologist at the IBM Hursley Development Laboratory in England and an IBM Distinguished Engineer. He is a member of the British Computer Society, a member of IEEE and ACM, and a member of the IBM Academy. He has a B.Sc. in experimental psychology and computer science from the Victoria University of Wellington, New Zealand. He joined IBM at Hursley in 1974, working on advanced computer languages, image systems, workflow, and high-function graphics. In 1983 he was assigned to the staff of the IBM Corporate Technical Committee in Armonk, New York, returning to the United Kingdom a year later to run the IBM United Kingdom Scientific Centre. Mr. Brackenbury has been responsible for a wide range of software advanced development including prototypes for: Object Rexx; desktop hypertext and multimedia; person-to-person conferencing; and OMG's transactional service, OTS. Since its inception in 1995, he has been leading the IBM Centre for Java Technology, responsible for porting Java to over a dozen platforms.

**Donald F. Ferguson** *IBM Software Group, Thomas J. Watson Research Center, 30 Saw Mill River Road, Hawthorne, New York 10532 (electronic mail: dff@us.ibm.com).* Dr. Ferguson joined IBM as a research staff member in 1987 and is currently a Distinguished Engineer in the IBM Software Group. He is also Chief Architect and Technical Leader for IBM's Component Broker product and Enterprise JavaBeans implementations. During his career in IBM, he has contributed to IBM cache management, operating system and transaction processing workload management, multimedia content server, system management, and distributed object-oriented products. He is an author of seven current or pending patents and over two dozen technical publications. Dr. Ferguson has received two IBM Outstanding Innovation Awards, four Research Division Technical Awards, two IBM Invention Plateau Awards, an IEEE best paper award, and was elected to the IBM Academy of Technology in 1997.

**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.

**R. A. (Tony) Storey** *IBM United Kingdom Laboratories Ltd., Hursley Park, Winchester, Hants SO21 2JN, United Kingdom (electronic mail: Tony_Storey@uk.ibm.com).* Dr. Storey is an IBM Distinguished Engineer focusing on IBM's transaction processing products and direction, working in the Transaction Systems organization in the IBM UK Laboratory. He joined the UK laboratory in 1980, working on transaction processing systems and has been heavily involved in the direction of one of IBM's premier transaction processing products, CICS, over a number of years. He was also very much involved with the original concept and definition of IBM's MQSeries. More recently his focus has been on transaction processing and its relationship with Java, and object technologies generally. He joined IBM at the United Kingdom Scientific Centre in 1974, working in the area of relational database research. He received a Ph.D. from the University of Durham, England, in 1967.

Reprint Order No. G321-5679.