What Is the Java™ 2 Platform, Enterprise Edition?

This document provides an introduction to the features and benefits of the Java 2 platform, Enterprise Edition.

Overview

Enterprises today need to extend their reach, reduce their costs, and lower their response times by providing easy-to-access services to their customers, partners, employees, and suppliers.

Typically, applications that provide these services must combine existing enterprise information systems (EIS) with new business functions that deliver services to a broad range of users. These services need to be:

- *Highly available*, to meet the needs of today's global business environment.
- *Secure*, to protect the privacy of users and the integrity of enterprise data.
- *Reliable and scalable*, to insure that business transactions are accurately and promptly processed.

For a variety of reasons, these services are generally architected as distributed applications consisting of several tiers, including clients on the front end, data resources on the back end, and one or more middle tiers between them where the majority of the application development work is done. The middle tier implements the new services that integrate existing EISs with the business functions and data of the new service. The middle tier shields the client tier from the complexity of the enterprise and takes advantage of rapidly maturing Internet technologies to minimize user administration and training.

The Java 2 platform, Enterprise Edition reduces the cost and complexity of developing these multi-tier services, resulting in services that can be rapidly deployed and easily enhanced as the enterprise responds to competitive pressures.

The Java 2 platform, Enterprise Edition (J2EE) achieves these benefits by defining a standard architecture that is delivered as the following elements:

- **J2EE Platform** - A standard platform for hosting J2EE applications, specified as a set of required APIs and policies.
- **J2EE Compatibility Test Suite** - A suite of compatibility tests for verifying that a J2EE platform product is compatible with the J2EE platform standard.
- **J2EE Reference Implementation** - A reference implementation for demonstrating the capabilities of J2EE and for providing an operational definition of the J2EE platform.
- **Sun Blueprints**TM **Design Guidelines for J2EE** - Guidelines that describe a standard programming model for developing multi-tier, thin-client applications.

The following sections of this document describe each of these elements in greater detail.

Origins of J2EE

During the early `90s, traditional enterprise information system providers began responding to customer needs by shifting from the two-tier, client-server application model to more flexible three-tier and multi-tier application models. The new models separated business logic from both system services and the user

interface, placing it in a middle tier between the two. The evolution of new middleware services - transaction monitors, message-oriented middleware, object request brokers, and others - gave additional impetus to this new architecture. At much the same time, the growing use of the internet and intranets for enterprise applications contributed to a greater emphasis on lightweight, easy to deploy clients.

Multi-tier design dramatically simplifies developing, deploying, and maintaining enterprise applications. It enables developers to focus on the specifics of programming their business logic, relying on various backend services to provide the infrastructure, and client-side applications (both standalone and within web browsers) to provide the user interaction. Once developed, business logic can be deployed on servers appropriate to existing needs of an organization. However, despite these clear benefits, the model limits developers' ability to build applications from standardized components, to deploy a single application on a wide variety of platforms, or to readily scale applications to meet changing business conditions.

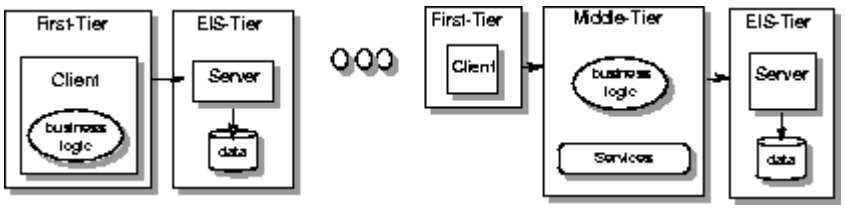
Within Sun Microsystems, several development efforts pointed us toward what would become J2EE technology. First, the Java servlets technology showed that developers were eager to create CGI like behaviors that could run on any web server that supported the Java platform. Second, the JDBC technology gave us a model for marrying the "Write Once, Run Anywhere" features of the Java programming language to existing database management systems. Finally, the success of the Enterprise JavaBeans component architecture demonstrated the usefulness of encapsulating complete sets of behavior into easily configurable, readily reusable components. The convergence of these three concepts -- server-side behaviors written in the Java language, connectors to enable access to existing enterprise systems, and modular, easy to deploy components -- led us and our industry partners to the J2EE standard.

J2EE Application Model

J2EE is designed to support applications that implement enterprise services for customers, employees, suppliers, partners, and others who make demands on or contributions to the enterprise. Such applications are inherently complex, potentially accessing data from a variety of sources and distributing applications to a variety of clients.

To better control and manage these applications, the business functions to support these various users are conducted in the middle tier. The middle tier represents an environment that is closely controlled by an enterprise's information technology department. The middle tier is typically run on dedicated server hardware and has access to the full services of the enterprise.

J2EE applications often rely on the EIS-Tier to store the enterprise's business-critical data. This data and the systems that manage it are at the inner-core of the enterprise.



Two-Tier vs. Multi-Tier Application Models

Originally, the two-tier, client-server application model promised improved scalability and functionality. Unfortunately, the complexity of delivering EIS services directly to every user and the administrative problems caused by installing and maintaining business logic on every user machine have proved to be

major limitations.

These two-tier limitations are avoided by implementing enterprise services as multi-tier applications. Multi-tier applications provide the increased accessibility that is now demanded by all elements of an enterprise. This shift is driving major investments in the development of middle-tier software.

Developing multi-tier services has been complicated by the need to develop both the service's business function and the more complex infrastructure code required to access databases and other system resources. Because each multi-tier server product had its own application model, it was difficult to hire and train an experienced development staff. In addition, as service volume increased it was often necessary to change the whole multi-tier infrastructure, resulting in major porting costs and delays.

The J2EE application model defines an architecture for implementing services as multi-tier applications that avoid these problems and deliver the scalability, accessibility, and manageability that is needed.

The J2EE application model partitions the work needed to implement a multi-tier service into two parts: the business and presentation logic to be implemented by the developer, and the standard system services provided by the J2EE platform. The developer can rely on the platform to provide the solutions for the hard systems-level problems of developing a middle-tier service.

The J2EE application model provides the benefits of Write Once, Run Anywhere portability and scalability for multi-tier applications. This standard model minimizes the cost of developer training while providing the enterprise with a broad choice of J2EE servers and development tools.

The J2EE application model is a major step forward in simplifying and expediting application development, by minimizing the complexity of building multi-tier applications.

The J2EE application model is being provided as part of the Sun BluePrints™ best practices program. The *Sun BluePrints Design Guidelines for J2EE* will be presented through a website (http://java.sun.com/j2ee/blueprints) and in a forthcoming Addison-Wesley Java Series book, *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition*. The design guidelines will include a sample application demonstrating the model, and discussions of best practices for using various services provided the J2EE platform.

Java Technology Foundation

The J2EE application model begins with the Java programming language and the Java virtual machine. The proven portability, security, and developer productivity they provide forms the basis of the application model.

The application model also includes the JavaBeans component model. JavaBeans components make it easy to componentize the Java technology-based code for common functions, then customize and combine these components visually with JavaBeans development tools.

Platform-Independent Security

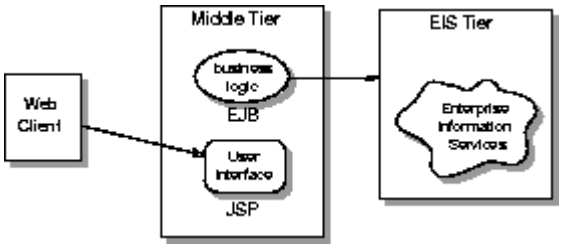
While other enterprise application models require platform-specific security measures in each application, the J2EE platform's security environment enables security constraints to be defined at deployment time. By shielding applications from the complexity of implementing security, the J2EE platform makes them portable to a wide variety of security implementations.

The J2EE platform defines standard declarative access control rules to be defined by the application programmer/assembler and interpreted when the application is deployed on the enterprise platform. J2EE also requires platform vendors to supply standard login mechanisms so applications do not have to incorporate these mechanisms into their logic. The same program works in a variety of different security environments without change to the source code.

As an example, a J2EE application developer can specify several levels of security (say user, super-user, and administrator), then write code to check the current user's permission level when accessing secure operations. At deployment time, the Application Deployer assigns groups of users to the appropriate security levels, enabling the application to easily verify permission level before performing the restricted operations.

The Middle Tier

The major benefit of the J2EE application model is in the middle tiers of multi-tier applications. In the J2EE platform, middle-tier business functions are implemented as Enterprise JavaBean components, as shown in FIGURE 2. These enterprise beans allow service developers to concentrate on the business logic and let the EJB server handle the complexities of delivering a reliable, scalable service.



EJB Components Implement Business Logic in the Middle Tier

JavaServer Pages technology and servlets present middle-tier functions to the client tier as simple-to-access Internet-style services. JavaServer Pages (JSP) technology makes it easy for user interface developers to present dynamically generated pages to anyone with a browser. Servlets give more sophisticated developers of Java technology-based applications the freedom to implement dynamic presentations completely in the Java programming language.

The Client Tier

The J2EE platform supports several types of clients.

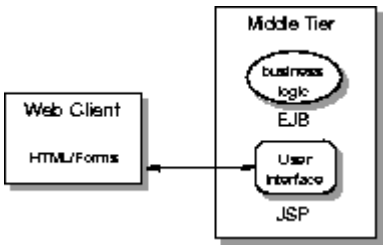
Many J2EE services will be designed to support web browser clients. These services interact with their clients via dynamically generated HTML pages and forms.

More sophisticated services will interact with their first-tier clients by directly exchanging business data. Here, JSPs and Servlets are used to format this business data in a way that is easy for J2EE clients to work with. These clients can be both Java applets running in a web browser and Java technology-based programs.

It is important to note that security is a key part of all multi-tier services. In J2EE, security is handled almost entirely by the platform and its administrators. In most cases, neither the service nor its clients require developer-written security logic.

HTML Page Based Clients

A service can be presented directly to a user's web browser as dynamically generated HTML pages. JavaServer Pages technology is an easy way to dynamically compose these pages using a familiar scripting paradigm that combines HTML and Java technology-based code, as shown in FIGURE 3. In some cases, a service may require some fairly complex code. This can be handled by placing code in a JavaBeans component and calling it from a JSP. A service can also be directly programmed in the Java programming language using a servlet.

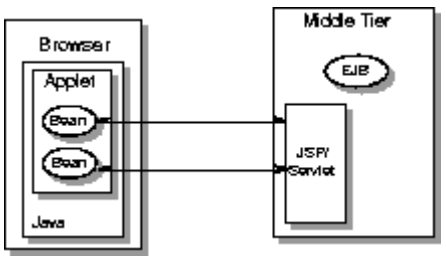


Presenting Services Directly to a Browser

HTTP Content Based Clients

It is often useful to provide functionality directly at the client that helps a user organize and interact with the service's information. In this case, the service exchanges raw content with the client instead of HTML pages. This content is typically in the form of XML documents that are exchanged between the client and the service using the HTTP protocol.

Typically this XML content is handled in the first-tier by JavaBeans components that are provided by the service in an applet that is automatically downloaded into a user's browser, as shown in FIGURE 4. To avoid problems caused by old or non-standard versions of the Java runtime environment in a user's browser, the J2EE application model provides special support for automatically downloading and installing the Java Plug-in, Sun's Java runtime environment that can be dynamically loaded into the most popular browsers. This content can also be handled by a Java technology-based program acting as a J2EE client. This flexible client model provides the developer with a broad range of choices for presenting a distributed application's user interface on the internet.



Providing JavaBeans components to a Browser

Intranet Clients

Both HTML page based services and HTTP content based services can be effectively used on an enterprise's intranet as well as the Internet.

In addition, the intranet provides the extra infrastructure that allows Java programs to directly access

EJBs within the intranet domain.

Other Client Types

J2EE services presented via standard HTTP, HTML and XML are easily accessible to all clients including Microsoft clients such as Visual Basic and Office 2000.

One goal of Enterprise JavaBeans technology is to define CORBA standard RMI-IIOP as the required interoperability mechanism. This will make any J2EE service available to any CORBA client, ensuring more complete integration between the J2EE platform and existing enterprise information systems. While most elements of this standard are complete there are a few items that are still in progress. After the final work is complete, J2EE will add this interoperability requirement. In the interim, many J2EE vendors will support the parts of the standard that are available.

In conjunction with J2EE, Sun will provide white papers and technology demonstrations that illustrate techniques for integrating Microsoft COM objects with EJBs using RMI-IIOP. These will cover how to access EJBs from first-tier clients such as Visual Basic and Windows 2000 via COM, as well as using EJBs in combination with middle-tier functions implemented in Microsoft Transaction Server.

The Enterprise Information Systems

A service's middle-tier business functions must access and update the information in the EIS-tier.

The following standard Java service APIs provide basic access to these systems:

- **JDBC -** the standard API for accessing relational data from Java.
- **Java Naming and Directory Interface (JNDI)** - the standard API for accessing information in enterprise name and directory services.
- **Java Message Service (JMS)**[1] - the standard API for sending and receiving messages via enterprise messaging systems like IBM MQ Series and TIBCO Rendezvous.
- **JavaMail** - the standard API for sending E-mail.
- **JavaIDL** - the standard API for calling CORBA services.

J2EE Declarations

An important goal of the J2EE application model is to minimize application programming.

One of the ways that this is accomplished is to shift the burden of implementing common tasks to the J2EE platform. These common tasks include enforcing an application's security roles, implementing its transaction semantics, and linking its components to the resources and other components they require.

J2EE provides a simple, declarative way to specify these behaviors. These declarations are separated from component code and stored in a *deployment descriptor* that is part of the application package. These XML-based declarations enable Application Deployers to modify the behavior of an application without having to modify any of the components themselves.

J2EE Platform

The J2EE platform is the standard environment for running J2EE applications. The J2EE platform is composed of the following elements:

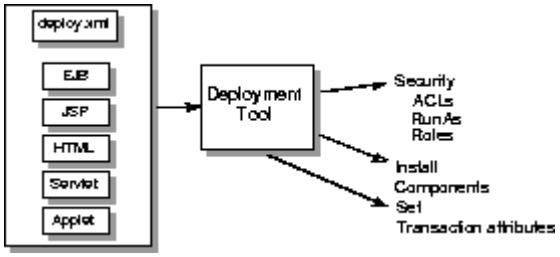
- **J2EE deployment specification** - a standard that defines a common way of packaging applications for deployment on any J2EE compatible platform.
- **Java technology standards for the J2EE platform** - a set of standards that all J2EE platform products must support.
- **IETF standards for the J2EE platform** - a set of standards defined by the Internet Engineering Task Force that all J2EE platform products must support.
- **CORBA standards for the J2EE platform** - a set of CORBA standards upon which the J2EE platform bases its middle-tier interoperability.

The J2EE platform defines the rich set of facilities that are needed to implement enterprise-class, multi-tier services. The J2EE platform is based on proven, open standards to deliver the broadest adoption and highest level of portability.

J2EE Application Assembly and Deployment

A J2EE application is packaged into one or more standard units for deployment to any J2EE platform-compliant system. Each unit contains a functional component or components (enterprise bean, JSP page, servlet, applet, etc.), a standard deployment descriptor that describes its content, and the J2EE declarations which have been specified by the application developer and assembler.

Once a J2EE unit has been produced, it is ready to be deployed to a J2EE platform, as shown in FIGURE 5.



Deploying J2EE Applications

Deployment typically involves using a platform's deployment tool to specify location-specific information, such as a list of local users that can access it and the name of the local database. Once deployed on the local platform, the application is ready to run.

Java Technology Standards for the J2EE Platform

The primary element of the J2EE platform is the list of Java technology standards that all J2EE products are required to support.

Since the J2EE platform is focused on the end-to-end development of enterprise solutions, it goes beyond simply requiring that each Java API be supported. It requires that each API be fully integrated with the platform. This insures that the platform delivers a consistent end-to-end environment for the deployment of J2EE applications.

IETF Standards for the J2EE Platform

The emergence of the Internet has had a major impact on the way enterprise applications are developed.

This revolution is based on standards set by the Internet Engineering Task Force (IETF), including HTML, HTTP, and now XML, the internet standard for communicating structured data.

The Java programming language, having grown up with IETF standards, has become the preferred way of writing applications for them. The J2EE application model and the J2EE platform continue this trend. In its current iteration, the J2EE platform supports HTML and HTTP clients, and can support XML clients. In addition, J2EE deployment descriptors make use of XML to provide application information in a platform-independent way. Future versions of the J2EE platform will likely define greater integration of XML for communicating data between tiers, thus further enhancing portability of J2EE applications.

CORBA Technology Standards for the J2EE Platform

The Object Management Group (OMG) in conjunction with Sun has produced the RMI-IIOP specification. This standard defines how the CORBA IIOP protocol is used by the Java Remote Method Invocation facility.

The EJB specification uses the application mapping for RMI-IIOP as its standard for calling EJBs. The J2EE platform strongly supports the use of RMI-IIOP. Sun is working closely with the other OMG members on future directions involving EJB technology and CORBA.

J2EE Compatibility Test Suite

J2EE platform vendors will need to verify that their implementations conform to the J2EE platform specification. Toward that end, Sun Microsystems, Inc. will license to platform vendors the J2EE Compatibility Test Suite (CTS).

Licensees will deploy, configure, and run this test suite (via its GUI framework) on their platform implementations. The suite will include tests for ensuring that the J2EE APIs are implemented. The tests will verify that the J2EE component technologies are available and working together properly. It will also include a set of fully functional J2EE applications to verify that all platforms are capable of deploying and running them consistently.

J2EE Reference Implementation

The J2EE reference implementation fulfills several roles.

Its primary role is as an operational definition of the J2EE platform. In this role, it is used by vendors as the J2EE platform's "gold standard" to determine what their implementation must do under a particular set of application circumstances. It is also used by developers to verify the portability of an application. Most importantly, it is used as the standard platform for running the J2EE Compatibility Test Suite.

A secondary, but more visible, role for the reference implementation is as a freely available platform for popularizing Java 2 platform, Enterprise Edition. Although it is not a commercial product and its licensing terms will prohibit its commercial use, it will be freely available in binary form for demonstrations, prototyping and academic research.

The reference implementation will also be made available in source form.

JMS is not required for J2EE 1.0; however, it will be made mandatory in a later release.