Patricia Seybold Group

Strategic Technologies, Best Practices, Business Solutions

Java™ 2 Platform, Enterprise Edition

Ensuring Consistency, Portability, and Interoperability

By Anne Thomas
June 1999

Prepared for Sun Microsystems

85 Devonshire Street, 5th Floor, Boston, MA 02109 • Phone 617.742.5200 • Fax 617.742.1028 • www.psgroup.com

Table of Contents

Illustrations and Tables

Java™ 2 Platform, Enterprise Edition

Ensuring Consistency, Portability, and Interoperability

By Anne Thomas
Prepared for Sun Microsystems

Executive Summary

Supporting Enterprise Computing

Sun Microsystems' Java™ 2 platform, Enterprise Edition (J2EE) defines a Java platform that is geared specifically to support the rigorous requirements of enterprise computing. The term "enterprise" implies extremely robust computing. Enterprise applications support core business operations, and any failure of these applications causes an interruption in business operations.

E-Commerce Requirements

The Internet has added a significant level of complexity to enterprise applications, which must now support all business operations, including those conducted with external business partners and customers. In other words, these applications must support business-to-business and business-to-consumer electronic commerce. The definitive characteristics of this new class of enterprise applications are scalability, availability, reliability, security, transactional integrity, and distribution.

Multitier Distributed Object Architecture

To support this robust level of service, enterprise applications are often designed using a multitier, distributed object architecture. A multitier application is implemented as a set of server-side application components. This type of application can be distributed across multiple physical systems, enabling unmatched scalability characteristics. It can also support any type of thin-client interface. These multitier applications rely on a variety of middleware services, such as naming, security, transactions, messaging, and databases.

Enterprise Java APIs

On April 12, 1997, Sun announced an initiative to develop the Java Platform for the Enterprise. Using the open Java Community Process, Sun fostered the development of a collection of standard Java extensions known as the Enterprise Java™ APIs. These application-programming interfaces (APIs) provide vendor-independent programming interfaces for a variety of middleware implementations. The keystone of the Enterprise Java APIs is the Enterprise JavaBeans™ API, which defines a server-side component model and a vendor-independent programming interface for Java application servers.

J2EE Platform

Two years later, the specifications for the Enterprise Java APIs are complete, and many vendor implementations are available. In particular, Enterprise JavaBeans (EJB) has become the de facto standard component model for Java application servers. There are more than twenty EJB implementations available. But EJB by itself isn't enough to guarantee portability, interoperability, and platform consistency. EJB defines a comprehensive programming model and a standard API, but it does not specify implementation details such as communications and transaction protocols. Meanwhile, the Enterprise Java APIs are still classified as standard extensions to the Java platform. Because they are extensions, there's no guarantee that the APIs will be installed on a specific system. By defining an enterprise platform, Sun provides a mechanism to certify that a system supports a complete Java environment that contains the Enterprise Java APIs based on interoperable protocols. A certified J2EE platform provides a consistent, integrated Java runtime environment that guarantees a certain quality of service and ensures application portability and interoperability.

Java 2 Platform, Enterprise Edition

J2EE Platform: Supporting Electronic Business

Business Drivers

As we approach the new millennium, businesses are faced with ever-increasing competition. Every business has to figure out new ways to produce better products in less time and at less cost, to devise better ways to attract and retain customers, and to find better ways to adapt its products and services as fast as possible to maintain a competitive edge.

Electronic Business

In response, many companies are turning to electronic business solutions. Electronic business can open up new channels to increase revenues. Business-to-business e-commerce can streamline supply chains to improve efficiency. Self-service customer care systems can increase customer loyalty while dramatically reducing costs.

Increased Complexity

But electronic business also adds a significant level of complexity to enterprise information systems. Enterprise application systems now need to reach out well beyond the confines of the traditional corporation. Companies must begin to integrate their internal systems with those of their partners, suppliers, and distributors. Customers now expect to be able to use the Internet to place and track orders, report problems, and manipulate account information. The Internet places much more challenging requirements on these systems, including:

- **Scalability.** The Internet is huge. In the past, enterprises needed to support hundreds to perhaps thousands of concurrent users. Now they must design systems that can support hundreds of thousands to perhaps millions of concurrent users.

- **Availability.** The Internet never goes to sleep. Enterprise application systems now have to be continuously available. System managers no longer have the luxury of doing maintenance during scheduled downtime.
- **Security.** The Internet has no walls. An enterprise can no longer rely on physical boundaries to protect systems and data. Anyone on the Internet can potentially infiltrate the enterprise application systems. Enterprise computing on the Internet requires different methods to identify and authenticate users, guard against unauthorized use of the systems, and protect the integrity of the data. Enterprise applications must also protect the privacy of individuals and customers.

Supporting New Types of Clients

In addition to supporting the Internet, many businesses are finding new ways to increase revenues, reduce costs, and increase efficiency by extending the reach of enterprise applications to new types of clients. Wireless and satellite communications are seeping into the mainstream. Enterprise systems can directly communicate with the computers embedded in a variety of electronic devices, such as mobile phones, pagers, personal data assistants (PDAs), gas pumps, vending machines, well heads, and utility meters.

Java 2 Platform, Enterprise Edition

The Java 2 platform, Enterprise Edition is designed to support the rigorous requirements of modern, extended, e-business-oriented, enterprise application systems. J2EE provides a component-based, server-centric, multitier application architecture. Illustration 1 provides an overview of the J2EE environment.

Enterprise JavaBeans

Enterprise JavaBeans (EJB) technology forms the foundation for the J2EE platform. It defines a model for developing and deploying reusable Java server components, and it defines a standard programming interface for Java application servers.

EJB Application Servers

EJB components execute within a Java application server, which provides a mission-critical runtime environment that can support any type of client device. An application server combines traditional OLTP technologies with new distributed object technologies to provide a high-performance, highly scalable, robust execution environment. An application server simplifies the development of scalable, server-based application systems by automatically managing and recycling expensive system resources. Examples of such system resources include operating system processes, threads, memory, database connections, and network sessions. All of these are very expensive to create, so, for optimal performance, an application server creates these resources once and then saves and recycles them for subsequent requests.

Simplifying Development

EJB technology also provides an integrated application framework that dramatically simplifies the process of developing enterprise-class application systems. An EJB server automatically manages a number of tricky middleware services, such as transactions, state, persistence, and security, on behalf of the application components. EJB component builders can concentrate on writing business logic

rather than complex middleware. As a result, applications get developed more quickly, and the code is of better quality.

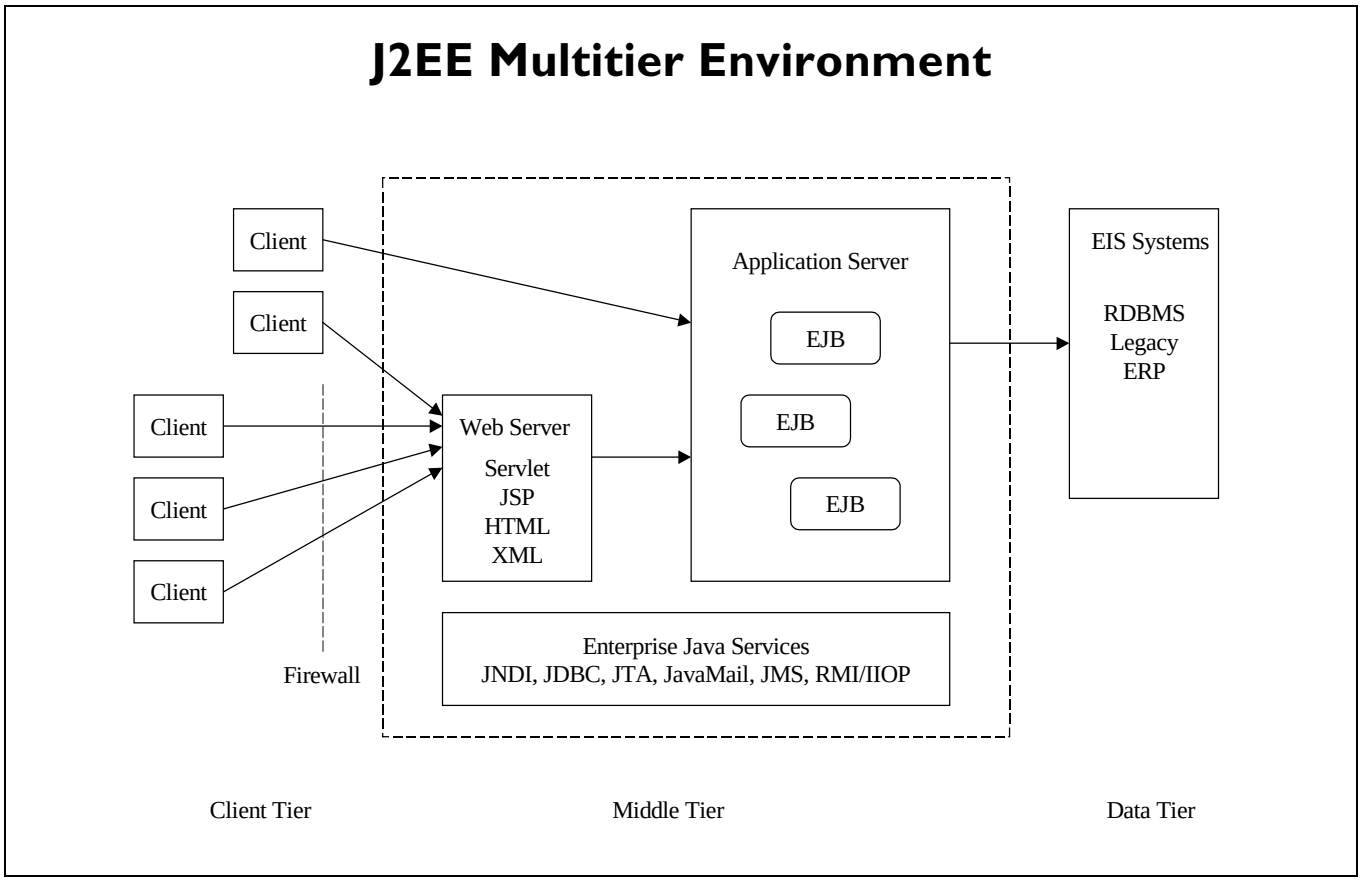

Illustration 1. J2EE provides a component-based, e-business-oriented, multitier application architecture.

Enterprise Java APIs

Enterprise applications require access to a variety of distributed middleware services, such as naming, security, transactions, messaging, and database. The Enterprise Java APIs, which are available as Java Standard Extensions, provide access to these middleware services. These APIs are designed to layer on multivendor, heterogeneous infrastructure services. Each API provides a common programming interface to a generic type of infrastructure service. Table 1 provides an overview of the Enterprise Java APIs.

Servlets and JSP

E-business requirements demand that modern enterprise applications be accessible to users across the Internet. J2EE fully supports Web clients using servlet and JavaServer Pages™ (JSP) technology. Servlets and JSP are server components that normally run within a Web server. Servlets are written as Web server extensions (i.e., separate from the HTML page), while JSP embeds the Java code directly in HTML. At deployment time, the JSP Java code is automatically converted into a servlet. Servlets process Web requests, pass them into the back-end enterprise application systems, and dynamically render the results as HTML or XML client interfaces. Servlets also

manage the browser user's session information, so that users don't need to repeatedly input the same information.

Enterprise Java APIs

API	Description
EJB	Enterprise JavaBeans is a server component model that provides portability across application servers and implements automatic services on behalf of the application components.
JNDI	Java Naming and Directory Interface provides access to naming and directory services such as DNS, NDS, LDAP, and CORBA Naming.
RMI/IIOP	Remote Method Invocation creates remote interfaces for Java-to-Java communications. This extension uses the CORBA standard IIOP communications protocol.
Java IDL	Java Interface Definition Language creates remote interfaces to support Java-to-CORBA communications. Java IDL includes an IDL-to-Java compiler and a lightweight ORB that supports IIOP. (Java IDL is part of the Java 2 platform, Standard Edition, formerly known as the Java Development Kit.)
Servlets and JSP	Java servlets and Java Server Pages are server components that run in a Web server that supports dynamic HTML generation and session management for browser clients.
JMS	Java Messaging Service supports asynchronous communications using either a reliable queuing or publish and subscribe programming model.
JTA	Java Transaction API provides a transaction demarcation API.
JTS	Java Transaction Service defines a distributed transaction management service based on CORBA Object Transaction Service.
JDBC™	JDBC Database Access API provides uniform access to relational databases such as DB2, Informix, Oracle, SQL Server, and Sybase. (JDBC is part of the Java 2 platform, Standard Edition.)
JavaMail	JavaMail provides a protocol independent framework to build mail and messaging applications. (JavaMail requires the JAF API.)
JAF	JavaBeans™ Activation Framework provides standard services to determine the type of an arbitrary piece of data and activate an appropriate JavaBeans component to manipulate the data.

Table 1. An overview of the Enterprise Java APIs.

Overview of the Java Platforms

Java Technology

Java technology is a portable object-oriented programming environment. A Java environment consists of the Java programming language, a Java language compiler, and a Java virtual machine. The Java programming language consists of a number of classes and interfaces that are used to write Java applications. A Java language compiler compiles the programming language into a platform-independent

executable called Java bytecode. Java bytecode executes in a Java virtual machine. The Java virtual machine dynamically compiles or interprets the platform-independent Java bytecode, translating it into platform-specific instructions.

WORA

The essence of Java technology is defined by the slogan "Write Once, Run Anywhere™" (WORA). Any Java application can run in any Java virtual machine on any platform, without modification. Java virtual machines are available for nearly every computing platform—from the smallest smart cards to the largest supercomputers.

Java APIs

The Java programming language consists of a collection of Java classes and interfaces. Developers interact with these classes and interfaces through a set of specified application programming interfaces. The Java APIs are distributed in a hierarchical tree structure of class libraries. The primary Java APIs are packaged together in what are called the Java Core Classes, which for the most part fall within the java.* class hierarchy[1]. As the language has matured and the language requirements have expanded, the industry has developed a number of additional Java APIs that are referred to as Java Standard Extensions. These APIs generally fall within the javax.* class hierarchy.

Java Platform Specifications

To ensure consistency across platforms and to achieve WORA portability, Sun has defined a set of Java platform specifications. A Java platform specifies a minimum Java configuration that must be supported in a given runtime environment to ensure Java application portability. A platform specification defines exactly which Java APIs must be supported by a Java virtual machine.

Java Compatible Logo

Sun ensures the integrity of the Java brand and the WORA message using platform compatibility test suites and the Java Compatible logo. Sun supplies a Java platform compatibility test suite that vendors use to test their products. Any Java platform product that passes the compatibility test suite may display the Java Compatible logo. The logo indicates that the product is certified to support the minimum Java configuration as defined in the Java platform specification.

Four Java Platform Specifications

Since not all Java applications require the complete set of Java APIs, Sun has defined four different Java platforms.

- **Java 2, Standard Edition.** Java 2 is the latest release of Java technology. Java 2, Standard Edition (J2SE) defines the standard Java configuration required to support Java applications on a general purpose computing system. A J2SE platform includes complete support for the Java core classes APIs.

[1] The Java core classes also include the javax.accessibility, javax.swing.*, org.omg.CORBA.*, and org.omg.CosNaming.* APIs.

- **Java 2, Enterprise Edition.** Java 2, Enterprise Edition is a superset of J2SE, defining an extended Java configuration that supports enterprise-class application systems. A J2EE platform includes the Java core classes APIs and a number of Enterprise Java APIs.
- **PersonalJava.** PersonalJava is a subset of the standard Java platform that is designed to support the resource-constrained environment of an information appliance.
- **Java Card.** Java Card is a subset of the PersonalJava platform that defines the minimal Java configuration required to support Java applications on a smart card.

J2EE Platform Deliverables

According to standard Java technology practices, a Java platform must include a platform specification, a compatibility test suite, and a reference implementation. The J2EE platform also includes an application model to help developers build applications on this platform. The J2EE platform is defined by the following deliverables:

- **J2EE Platform Specification.** The platform specification defines the specific Java APIs that must be supported to guarantee the minimum quality of service. The specification indicates the specific release levels of the Java APIs and practices that will ensure compatibility, portability, and integration.
- **J2EE Compatibility Test Suite.** A vendor uses the compatibility test suite to verify that its implementation of the J2EE platform complies with the J2EE platform specification.
- **J2EE Reference Implementation.** The reference implementation is an implementation of the J2EE platform specification. It provides an operational definition of the J2EE platform, which can be used to demonstrate the capabilities of the platform or to test J2EE applications.
- **J2EE Application Model.** The application model provides guidance to help developers build enterprise applications that can run in a J2EE platform. The application model includes examples and describes a number of design patterns that have proven to be successful for enterprise deployments.

J2EE Architecture

Server-Based Computing

J2EE relies on a server-based computing architecture. Server-based computing moves the majority of application processing to a server or server cluster. Server-based computing affords two critical advantages over other application architectures.

- **Multiple Clients.** A server-based architecture requires a clean separation between the client presentation layer and the server-based application-processing layer. This approach allows a single application to simultaneously support any number

of different client interfaces, including rich GUI interfaces for corporate desktop devices, interactive multimedia interfaces for high-speed browser-based users, efficient text-based interfaces for low-speed browser-based users, landscape-constrained interfaces for wireless PDA users, and interactive voice response interfaces for telephone-based clients.

- **Robust Operations.** A server-based architecture supports unparalleled scalability, reliability, availability, and recoverability. Server-based applications can be partitioned and distributed across multiple processors. Application components can be replicated to support instant failover and recovery.

J2EE Architecture

A J2EE platform provides a prerequisite set of Java APIs and services to support enterprise applications. The complete platform can be implemented on a single system, or the platform services can be distributed across a variety of systems, but all specified APIs must be included somewhere within the total system. Illustration 2 shows an abstract overview of the architecture.

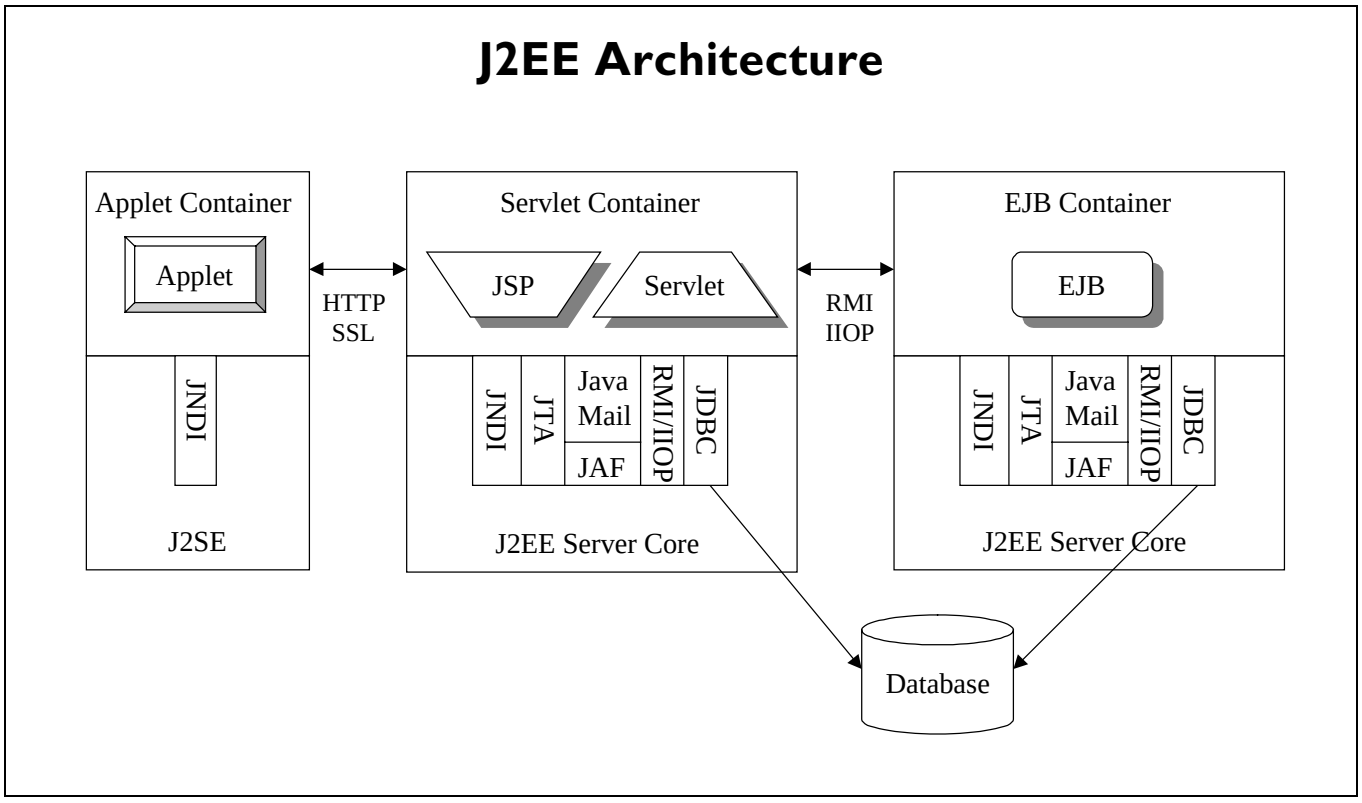

Illustration 2. J2EE supports a component-based multitier application architecture. The environment supports three types of application components: applets, servlets (including JSP), and Enterprise JavaBeans.

J2EE Server Core

The J2EE server core is an application server environment that provides resource and transaction management services. A J2EE platform vendor typically implements the J2EE server core based on an existing transaction processing infrastructure. The J2EE server core must include an implementation of the Java 2 Standard Edition Java virtual machine and the Java development kit (J2SE), it must support the J2EE Service APIs, and it must provide component containers for applets, servlets, JavaServer Pages components, and EJB components.

J2EE Service APIs

The J2EE platform defines a set of standard services that each J2EE platform must support. These standard services include the HTTP, JTA, RMI/IIOP, JavaIDL, JDBC, JNDI, JavaMail, and JAF APIs. Vendors can provide additional services, such as JMS, or connectors to other non-J2EE application systems.

Components

J2EE provides extensive support for components. Components are pre-developed pieces of application code that can be assembled into working application systems. Components are not standalone applications. They run within another application environment called a *container*. A container provides an application context for one or more components and provides management and control services for the components. In practical terms, a container provides an operating system process or thread in which to execute the component.

Component Containers

The J2EE architecture supports three different component types:

- **Applets.** Applets are mobile user interface components. They can be transported across the Internet and executed in a Web browser or in other applet containers. Applets require a container that supports J2SE, the applet programming model, and JNDI.
- **Servlets and JavaServer Pages.** Servlets and JavaServer Pages are server components that form a bridge between multiple clients and the back-end application system. Servlets and JSP normally run in a Web server, managing HTTP requests for services. The servlets and JSP process the requests and then render the results in a form that can be consumed by the client (normally in HTML or XML). Servlets and JSP require a container that supports J2SE, and the servlet, JSP, JNDI, JTA, JavaMail, JAF, RMI/IIOP, and JDBC APIs.
- **Enterprise JavaBeans.** Enterprise JavaBeans (EJB) are server components that implement the back-end application services. EJB runs in an application server. An EJB container must support J2SE, and the EJB, JNDI, JTA, JavaMail, JAF, RMI/IIOP, and JDBC APIs.

Database

The J2EE platform must include a database for the persistent storage of business data. Applications and systems access the database using the JDBC API. Other types of persistent data stores are permitted in addition to relational databases.

Resource Manager Drivers

A J2EE platform must supply access to various external resources, such as databases and external application systems. A *resource manager driver* is a system-level software component that provides access to an external resource manager. A vendor can implement a resource manager driver either by extending one of the J2EE standard service APIs (e.g., building a JDBC driver) or by defining and implementing a new API connector using the J2EE service provider interface (J2EE SPI). A driver that uses the J2EE SPI to attach to the J2EE platform can work with all J2EE platform implementations.

Platform Specifics

Ensuring Consistency and Portability

One of the primary goals of the J2EE platform is to establish a consistent environment that can support application portability. Enterprise applications require an extended set of products and services, many of which may come from different vendors. When a user has to mix and match products from multiple vendors, the project usually involves a fair amount of integration testing to ensure that the products actually work together. But in a J2EE platform, components are certified to work together, even if some of the platform components come from different vendors. It's the vendor's responsibility to perform the integration testing and to certify a complete platform.

Specific Release Levels

To simplify the integration testing process, the J2EE specification defines the specific release levels of the various Java APIs that must be used in the platform. The J2EE vendor must support that specific version of the API, even if a newer release might be available. The newer release may not be compatible with the rest of the platform.

Required Java APIs

The J2EE platform specification defines the specific Java APIs that must be included in a Java runtime environment in order to qualify as a J2EE platform. The J2EE platform supports three separate Java runtime environments—one for each of the three supported J2EE component types. Table 2 identifies the runtime requirements of the three environments.

Reference Implementation

Sun Microsystems will supply a reference implementation of the J2EE platform. The reference implementation will consist of an implementation of an application server that supports the required component models and implementations of the required Enterprise APIs. The reference implementation will be a basic functional environment that is designed to support evaluation and testing efforts rather than production application systems. The reference implementation is not likely to provide the high-performance and fault-tolerant capabilities that you could expect from a vendor product. Rather, its intent is to provide a semantically correct implementation of the specification.

Requirements of J2EE Runtime Environments

Standard Extension	Applets	Servlets	EJBs
J2SE 1.2 (includes Java IDL)	Y	Y	Y
JDBC 2.0	N	Y	Y
RMI/IIOP 1.0	N	Y	Y
EJB 1.1	N	N	Y
Servlets 2.2	N	Y	N
JavaServer Pages 1.1	N	Y	N
JNDI 1.1	Y	Y	Y
JTA 1.0	N	Y	Y
JavaMail 1.1	N	Y	Y
JavaBeans Activation Framework 1.0	N	Y	Y

Table 2. Java APIs required by the J2EE runtime environments.

Vendor Implementations

Numerous vendors will supply production-quality J2EE platforms. Potential vendors include BEA Systems, Bluestone, GemStone, IBM, Inprise, Iona, Oracle, Persistence, Sun/Netscape Alliance, and many others. Most vendor platforms will be centered on an application server product. A single vendor is not required to supply all components in a J2EE platform, but the vendor will identify any third-party products that are required to complete the platform. It is the vendor's responsibility to certify that the complete package complies with the J2EE compatibility test suite.

Competitive Landscape

Two Possible Contenders

J2EE is an extremely comprehensive runtime environment that has garnered support from most of the software industry. Very few vendors will produce platforms that compete with J2EE. Rather than competing, most vendors are choosing to implement a J2EE-compatible platform. J2EE has only two possible competitors: CORBA and Microsoft.

CORBA

The Object Management Group (OMG) has defined a comprehensive distributed object infrastructure based on an object request broker (CORBA) and a set of distributed object services (CORBAservices). OMG is in the process of defining a server-side component model similar to EJB called CORBA Component Model (CCM). Although the CORBA environment could be considered a competitor to J2EE, it is actually much more complementary than competitive. All J2EE platforms will be implemented using a CORBA foundation. J2EE uses CORBA's standard communication protocol, IIOP, which ensures interoperability with CORBA

systems. And CCM is being defined as a language-independent superset of EJB. A CCM-compatible application server will be able to host EJB components. More to the point, all major CORBA vendors (BEA Systems, IBM, Inprise, and Iona) provide EJB-compatible application servers and have endorsed J2EE. CORBA technology represents more of an enabling infrastructure to J2EE than a competitor.

Microsoft Windows DNA

Microsoft is the only vendor to offer an enterprise application development platform that competes head-to-head with J2EE. Windows Distributed interNet Applications Architecture (Windows DNA) is an enterprise application development model for the Windows platforms. As with J2EE, Windows DNA provides a distributed computing infrastructure for the development of enterprise applications that support the Internet and a wide range of client devices. Windows DNA is tightly integrated with the Windows operating systems, and it relies on inherent Windows distributed objects services, such as COM, Microsoft Transaction Server (MTS), the Distributed Transaction Coordinator (DTC), the Windows Registry, and the Windows security services. Windows DNA is based on a multitier application model that is similar to the one used in J2EE. In fact, there are a lot of similarities between the two platforms, as seen in Table 3.

Three Component Models

Windows DNA supports three application component models: ActiveX, Active Server Pages (ASP), and MTS components. These component models map one-to-one to the component models used in J2EE. ActiveX components correspond to applets. An ActiveX component can be transported across the Internet and executed in a Web browser that supports ActiveX, such as Internet Explorer. Active Server Pages correspond to servlets and JSP. An ASP runs in Microsoft Internet Information Server (IIS) connecting Web clients to back-end applications. An ASP processes Web requests and renders the results in HTML or XML. MTS components correspond to EJB components. An MTS component runs in the MTS application server, implementing the back-end application services.

Differences

Although the two enterprise application platforms are conceptually similar, there are fundamental differences between Windows DNA and J2EE. These differences can be summarized as vendor independence versus language independence.

Vendor Independence

J2EE is a vendor-independent platform. Consumers have a choice when selecting development products, deployment products, or deployment platforms. J2EE implementations will be available from a wide selection of vendors for a wide selection of hardware and operating systems. Even after selecting a particular product implementation, the consumer isn't locked into that choice. J2EE applications are portable from one vendor's platform implementation to another's. But there is a limitation: J2EE applications must be implemented in Java.

Language Independence

Windows DNA is a language-independent platform. Consumers have a choice when selecting the language and tools used to implement Windows DNA applications. Windows DNA supports a wide variety of programming languages, such as Java, C++, Visual Basic, Delphi, and PowerBuilder. But consumers don't have a choice

when it come to deployment products or deployment platforms. Windows DNA applications must be deployed using IIS, MTS, DTC, and OLE DB. Windows DNA server-side applications must be deployed on Windows NT (or, in the future, Windows 2000). Web clients are strongly encouraged to use Internet Explorer (especially if using ActiveX components). Non-Web clients must be deployed on a Win32 platform (Windows CE, 95, 98, or NT).

Windows DNA and J2EE

Platform Service	Windows DNA	J2EE
Operating Systems	Windows CE/95/98/NT (future: Windows 2000)	Any operating system
Browser	Internet Explorer	Any browser
Browser Components	ActiveX components	Applets
Web Server	Internet Information Server	Any Web server
Web Server Components	Active Server Pages	Servlets and JavaServer Pages
Application Server	Microsoft Transaction Server	Any EJB application server (over 20 to choose from)
Server Components	MTS components	Enterprise JavaBeans
Communications Protocol	DCOM	IIOP
Database Access	ADO and OLE DB	JDBC and SQLJ
Transaction Management	Microsoft Distributed Transaction Coordinator	Any transaction service through JTA
Security	Microsoft security services	Java security services
Directory	Microsoft Registry (future: Active Directory)	Any directory through JNDI

Table 3. Comparison between Windows DNA and J2EE.

Integrated Solutions

Prior to J2EE, Windows DNA offered another critical advantage: Windows DNA is an integrated solution. Microsoft makes the effort to ensure that the various products that make up the Windows DNA platform actually work together. Prior to J2EE, when using Java to implement enterprise applications, the consumer was responsible for integrating products from multiple vendors. But J2EE levels the playing field. J2EE is also an integrated solution. The J2EE platform vendor does the work to test and integrate the various products that make up the J2EE platform.

Conclusions

Reduce Costs and Time to Market

Java technology is a highly productive programming environment that allows development teams to produce application solutions more rapidly and at less cost. J2EE brings these same benefits to the enterprise application development space.

Supporting Enterprise-Class Computing

J2EE defines a consistent Java environment that can support enterprise-class computing requirements. Any application server environment that carries the J2EE brand is certified to provide support for the critical enterprise services defined in the platform. The consistency of the platform supplies a level of confidence to organizations that are looking to implement enterprise applications using Java technology.

Freedom of Choice

Because J2EE is vendor-independent, consumers can choose platform implementations from a wide variety of vendors. Vendors can differentiate their products to support specific requirements, such as tight security, extreme scalability, bulletproof reliability, complex data structures, heterogeneous transactions, legacy integration, packaged application support, and ease-of-use. Consumers can select a platform implementation that best fits their specific requirements.

Integrated Services

Regardless of the vendor selected, the J2EE brand indicates that the platform services are integrated and guaranteed to work together. Consumers are no longer responsible for making numerous products from multiple vendors work together.

Third-Party Applications

The J2EE brand indicates that the certified environment can support any application that has been developed for the J2EE platform. Numerous third party vendors, such as IBM, Integral Development, The Theory Center, Sanga International, and Tradex, are building applications based on the EJB component model, and these applications will require a J2EE deployment platform. The J2EE brand ensures application portability across multiple vendor implementations.

Supporting E-Business

The Internet is forcing many businesses to adopt a new approach when building application systems. These new systems require a multitier distributed application architecture based on a scalable, robust infrastructure. J2EE provides the kind of solid, integrated platform that organizations need to tackle e-business.