

J2EE Platform Overview

An Alpha Information Systems India Pvt. Ltd. Perspective

**Need help energizing your enterprise computing needs?
Give us a call at +91-836-4251105,
E-mail: info@alpha.net
or visit our website www.alpha.net for more information
on how we can help you with your next project.**

A BRIEF PERSPECTIVE

From its introduction to the world in 1994 to current day, the Java™ programming language has revolutionized the software industry. Java has been used in a myriad of ways to implement various types of systems. As Java started becoming more and more ubiquitous, spreading from browsers to phones to all kinds of devices, we saw it gradually hone in on one particular area and establish its strength and value proposition: That area is the use of Java on servers. Over time, Java has become the chosen platform for programming servers.

Java provides its Write Once Run Anywhere™ advantage to IT organizations, application developers, and product vendors. IT organizations leverage the benefits of vendor independence and portability of their applications. The increasing availability of skilled Java programmers promoted Java's adoption in the industry. Unbelievably, the number of Java programmers has rocketed to 2.5 million developers in only five years.

The simplicity of the language and the explosive growth of its use on the Internet and the intranet urged numerous developers and IT organizations to embrace Java as the de facto programming language for their projects.

The client-server application architecture, a two-tier architecture, over time evolved to a multitier architecture. This natural progression occurred as additional tiers were introduced between the end-user clients and backend systems. Although a multitier architecture brings greater flexibility in design, it also increases the complexity for building, testing, deploying, administering, and maintaining application components. The J2EE platform is designed to support a multitier architecture, and thus it reduces this complexity.

During this time, corporate Internet usage changed. Corporations transitioned from providing a simple corporate Web site to exposing some of their not-so-critical applications to the external world. In this first phase of Internet experimentation, IT managers were still skeptical and the security police were adamantly unfriendly to the idea of using the Internet to run and expose business services.

Before long, more and more companies started to embrace the power of the Internet. For example, customer service organizations began to provide service on the Web, in addition to the traditional methods of supporting customers by phone and email. Such organizations recognized the major cost implications of providing online service. Customers could now help themselves for most problems, and call a customer service agent only for more serious issues.

Customers liked using the Web too, as it improved their productivity. Soon, customers started expecting more and more online services from companies, and companies had to step up and provide these services. If they did not, someone else would.

Since then, almost everything has gone online—banking, bill payment, travel, ticketing, auctioning, car buying services, mortgages and loans, pharmacies, and even pet food! New companies were created that had no business model (now we know) other than opening shop online. They thrived and they thrashed. Established companies had to make their online presence felt to face the challenges of these new kids on the block. This tremendous growth fueled the need for a robust, enterprise class, Web-centric application infrastructure.

Application Servers—The New Breed

As the acceptance and adoption of Java on the server side became more established, and the demand for Web-centric application infrastructure rose, we saw an emergence of a new breed of infrastructure applications—application servers. Application servers provided the basic infrastructure required for developing and deploying multitiered enterprise applications.

These application servers had numerous benefits. One important benefit was that IT organizations no longer needed to develop their proprietary infrastructure to support their applications. Instead, they could now rely on the application server vendor to provide the infrastructure. This not only reduced the cost of their applications, but also reduced the time-to-market.

Each application server had its own benefits and disadvantages. Because there were no standards for application servers, no two application servers were completely alike. Some application servers were based on Java, and these allowed you to write only Java components to run on that server, while others used different languages for development.

Convergence of Java Technologies

In the area of Web applications, there were significant developments in Java as well. The Common Gateway Interface (CGI) approach for developing Web-centric applications was resource-intensive and did not scale well. With the introduction of servlet technology, Java developers had an elegant and efficient mechanism to write Web-centric applications that generated dynamic content. However, writing servlets still took some effort and Java expertise.

Then, the Java Server Pages (JSP) technology was introduced, particularly for Web and graphic designers accustomed to Hypertext Markup Language (HTML) and JavaScript scripting. JSP technology made it easier for Web front developers to write Web-centric applications. One need not know Java and servlet programming to develop pages in JSP.

JSP technology addresses the need for a scripting language for Web application clients. Web designers skilled at HTML and JavaScript can quickly learn JSP technology and use it to write Web applications. Of course, the Web server translates JSPs into servlets, but that happens "under the wraps." Effectively, servlets and JSPs separate Web application development roles.

The standard approach for database access in Java applications is Java Database Connectivity (JDBC). The JDBC API (application programming interface) gives programmers the ability to make their Java applications independent of the database vendor. One can write a JDBC application that accesses a database using standard Structured Query Language (SQL). If the underlying database changes from one vendor's product to another, the JDBC application works without any code change, provided that the code is properly written and does not use any proprietary extensions from the first vendor. JDBC API is offered as part of the core APIs in the Java™ 2 Platform, Standard Edition (J2SE™).

J2SE (formerly known as Java Development Kit or JDK) is the foundation for all Java APIs. J2SE consists of a set of core APIs that define the Java programming language interfaces and libraries. Java developers use the J2SE as the primary API for developing Java applications. As requirements expand and the Java language matures over the years, the J2SE offers additional APIs as standard extensions.

As Java established its permanent role on the server side, and the adoption of various Java APIs became widespread, Sun put together an initiative to unify standards for various Java technologies into a single platform. The initiative to develop standards for enterprise Java APIs was formed under the open Java Community Process (JCP). Enterprise Java APIs are a collection of various APIs that provide vendor-independent programming interfaces to access various types of systems and services. The enterprise Java APIs emerged as the Java™ 2 Platform, Enterprise Edition (J2EE™).

The Rise of the J2EE Platform

The Enterprise Java Beans™ (EJB™) technology is one of the prominent, promising technologies in the J2EE platform. The EJB architecture provides a standard for developing reusable Java server components that run in an application server. The EJB specification and APIs provide a vendor-independent programming interface for application servers. EJB components, called enterprise beans, provide for persistence, business processing, transaction processing, and distributed processing capabilities for enterprise applications. In short, the EJB technology offers portability of business components.

Various application vendors, having come together with Sun under the open JCP to develop this standard, adopted and implemented the EJB specification into their application server products. Similar to JDBC application portability, EJB applications are portable from one application server vendor to another. Again, this is true if the application does not

use any vendor-dependent feature of the application server. J2EE technologies are now a proven and established platform for distributed computing for the enterprise.

Java Message Service (JMS) is another standard API in the J2EE platform. It brings the same kind of standardization to messaging as JDBC brought for databases. JMS provides a standard Java API for using message-oriented middleware (MOM) for point-to-point and publish/subscribe types of enterprise messaging. As with the other technologies, JMS brings vendor independence in the MOM products for Java.

In each of these areas, Sun and other companies collaborated in coming up with an acceptable standard under the auspices of the open JCP. The JCP coordinated the activities to develop these standards. This cooperation is a foundation for the success of these APIs.

J2EE Value Proposition

The J2EE platform, built on the Java programming language and Java technologies, is the application architecture that is best suited for an enterprise-distributed environment. The J2EE platform is a standard that brings the following benefits to IT organizations, application developers, and product vendors:

- Vendors develop products that can run on any system that supports the J2EE platform. With virtually no extra effort, their products are available on a wide range of system platforms.
- Corporate IT developers benefit from the advantages of portable component technology. IT applications become vendor-independent and release the IT organizations from the clutches of vendor lock-in.
- IT developers can focus on supporting business process requirements rather than building in-house application infrastructure. The application servers handle the complex issues of multithreading, synchronization, transactions, resource allocation, and life-cycle management.
- IT organizations can take advantage of the best available products built on a standard platform. They can choose among products and select the most suitable and cost-effective development products, deployment products, and deployment platforms based on their requirements.
- Adopting the J2EE platform results in a significant productivity increase. Java developers can quickly learn the J2EE APIs.
- Companies protect their investment by adopting the J2EE platform, since it is an industry-supported standard and not a vendor-defined lock-in architecture.
- Development teams can build new applications and systems more rapidly. This decreases time-to-market and reduces the cost of development.
- A standard development platform for distributed computing ensures that robust applications are built on a proven platform.
- The J2EE platform provides a clear, logical, and physical partitioning of applications into various tiers, thus naturally addressing multitiered application requirements.
- Developers can either build their own J2EE component or procure it from the rapidly growing third-party components market. Vendors are able to offer their components individually, and customers are able to buy these software parts as needed.

J2EE PLATFORM

The previous section described the core technology components of the J2EE platform, such as servlet, JSP, EJB, JDBC, and JMS. In this section, we take a look at the J2EE architecture model and describe other aspects of the J2EE platform that complete the platform definition.

J2EE Architecture

The J2EE architecture is a multitiered architecture.

The J2EE architecture consists of the following tiers:

- **Client tier**—The client tier interacts with the user and displays information from the system to the user. The J2EE platform supports different types of clients, including HTML clients, Java applets, and Java applications.
- **Web tier**—The Web tier generates presentation logic and accepts user responses from the presentation clients, which are typically HTML clients, Java applets, and other Web clients. Based on the received client request, the presentation tier generates the appropriate response to a client request that it receives. In the J2EE platform, servlets and JSPs in a Web container implement this tier.
- **Business tier**—This tier handles the core business logic of the application. The business tier provides the necessary interfaces to the underlying business service components. The business components are typically implemented as EJB components with support from an EJB container that facilitates the component life cycle and manages persistence, transactions, and resource allocation.
- **EIS tier**—This tier is responsible for the enterprise information systems, including database systems, transaction processing systems, legacy systems, and enterprise resource planning systems. The EIS tier is the point where J2EE applications integrate with non-J2EE or legacy systems.

Java 2 Standard Edition

J2SE is the underlying base platform for J2EE, hence a brief discussion on the J2SE platform is relevant to the J2EE platform. The J2SE platform includes two deliverables:

- Java 2 SDK, Standard Edition (J2SE SDK)
- Java 2 Runtime Environment, Standard Edition (JRE)

J2SE SDK, formerly the JDK, is the Java programming language's core API set. J2SE provides the Java language functionality as well as the core libraries required for Java development. The core libraries are the classes within the `java.*` packages. In addition, J2SE provides auxiliary interfaces and libraries as extensions. It makes these standard extensions available as `javax.*` packages.

J2SE includes tools and APIs for developing applications with graphical user interfaces (GUIs), database access, directory access, Common Object Request Broker Architecture (CORBA), fine-grained security, input/output functions, and many other functions.

J2EE Application Components and Containers

The J2EE component container supports application components in the J2EE platform. A container is a service that provides the necessary infrastructure and support for a component to exist and for the component to provide its own services to clients. A container usually provides its services to the components as a Java compatible runtime environment.

The core application components in the J2EE platform are as follows:

- **Java application components**—standalone Java programs that run inside an application container.
- **Applet components**—Java applets that run inside an applet container, and which are usually supported via a Web browser.
- **Servlets and JSPs**—Web-tier components that run in a Web container. Servlets and JSPs provide mechanisms for dynamic content preparation, processing, and formatting related to presentation.
- **EJB components**—Coarse-grained business components that are run inside an EJB container (usually bundled in an application server product). EJB components, or enterprise beans, come in two types: session beans and entity beans. Session beans are enterprise beans that are suitable for processing or workflow. Session beans come in two flavors: stateful and stateless. A stateful session bean retains client state between method invocations. A stateless session bean does not retain client-specific state between client-invoked methods. Stateless session beans are used when no state needs to be stored between method invocations, and they may offer performance benefits over stateful session beans, which must be used when some state needs to be retained between invocations. Session bean instances pertain to a single user session and are not shared between users. Entity beans are used when a business component needs to be persisted and shared among multiple users. Entity bean persistence can be managed in two ways: bean-managed persistence (BMP) and container-managed persistence (CMP). BMP is used when the bean developer implements all mechanisms for persisting the state in the bean. CMP is used when the bean developer does not implement the persistence mechanisms in the bean. Instead, the bean developer specifies the necessary mapping between the bean attributes and the persistent storage and lets the container do the job.

The core focus of the J2EE patterns in this book is the design and architecture of applications using servlets, JSPs, and enterprise bean components.

Standard Services

The J2EE platform specifies the following standard services that every J2EE product supports. These services include APIs, which every J2EE product must also provide to application components so that the components may access the services.

- **HTTP**—Standard protocol for Web communications. Clients can access HTTP via the `java.net` package
- **HTTP over Secure Socket Layer (HTTPS)**—Same as HTTP, but the protocol is used over Secure Socket Layer for security.
- **JDBC**—A standard API to access database resources in a vendor-independent manner.
- **JavaMail**—An API that provides a platform-independent and protocol-independent framework to build mail and messaging applications in Java.
- **Java Activation Framework (JAF)**—APIs for an activation framework that is used by other packages, such as JavaMail. Developers can use JAF to determine the type of an arbitrary piece of data, encapsulate access to it, discover the operations available on it, and instantiate the appropriate bean to perform these operations. For example, JavaMail uses JAF to determine what object to instantiate depending on the mime type of the object.
- **Remote Method Invocation/Internet Inter-ORB Protocol (RMI/IIOp)**—Protocol that enables Remote Method Invocation (RMI) programmers to combine the benefits of using the RMI APIs and robust CORBA IIOp communications protocol to communicate with CORBA-compliant clients that have been developed using any language compliant with CORBA.
- **Java Interface Definition Language (JavaIDL)**—A service that incorporates CORBA into the Java platform to provide interoperability using standard IDL defined by the Object Management Group. Runtime components include Java ORB (Object Request Broker) for distributed computing using IIOp communication.
- **Java Transaction API (JTA)**—A set of APIs that allows transaction management. Applications can use the JTA APIs to start, commit, and abort transactions. JTA APIs also allow the container to communicate with the transaction manager, and allow the transaction manager to communicate with the resource manager.
- **JMS**—An API to communicate with MOM to enable point-to-point and publish/subscribe messaging between systems. JMS offers vendor independence for using MOMs in Java applications.

- **Java Naming and Directory Interface (JNDI)**—A unified interface to access different types of naming and directory services. JNDI is used to register and look up business components and other service-oriented objects in a J2EE environment. JNDI includes support for Lightweight Directory Access Protocol (LDAP), the CORBA Object Services (COS) Naming Service, and the Java RMI Registry.

J2EE Platform Roles

The J2EE platform uses a set of defined roles to conceptualize the tasks related to the various workflows in the development and deployment life cycle of an enterprise application. These role definitions provide a logical separation of responsibilities for team members involved in the development, deployment, and management of a J2EE application.

The J2EE roles are as follows:

- **J2EE product provider**—Provides component containers, such as application servers and Web servers, that are built to conform to the J2EE specification. The product provider must also provide tools to deploy components into the component containers. These tools are typically used by the deployer. In addition, the product provider must provide tools to manage and monitor the applications in the container. The system administrator typically uses these latter tools. This role is fulfilled by the product vendors.
- **Application component provider**—Provides business components built using the J2EE APIs. These components include components for Web applications as well as for EJB applications. This role is fulfilled by programmers, developers, Web designers, and so forth.
- **Application assembler**—Assembles, or puts together, a set of components into a deployable application. The assembler obtains the application components from the component providers. The application assembler packages the application and provides the necessary assembly and deployment instructions to the deployer.
- **Application deployer**—Deploys the assembled application into a J2EE container. The deployer may deploy Web applications into containers—Web containers, EJB containers, and so on—using the tools provided by the J2EE product provider. The deployer is responsible for installation, configuration, and execution of the J2EE application.
- **System administrator**—Has the responsibility of monitoring the deployed J2EE applications and the J2EE containers. The system administrator uses the management and monitoring tools provided by the J2EE product provider.
- **Tool provider**—Provides tools used for development, deployment, and packaging of components.

Deployment Descriptors

An application assembler puts a J2EE application together for deployment, and at the same time provides the assembly and deployment instructions in special files called deployment descriptors. The J2EE specification defines deployment descriptors as the contract between the application assembler and the deployer. Deployment descriptors are XML documents that include all the necessary configuration parameters required to deploy the J2EE application or J2EE components. Such configuration parameters specify external resource requirements, security requirements, environment parameters, and other component-specific and application-specific parameters. The deployer may use a deployment tool provided by the J2EE product provider to inspect, modify, customize, and add configuration parameters in these deployment descriptors to tailor the deployment to the capabilities of the deployment environment.

Deployment descriptors offer flexibility for the development and deployment of J2EE application components by allowing changes to configurations and dependencies as needed during the different application phases: the development, deployment, and administration phases. Much of this flexibility is due to descriptors defining parameters in a declarative fashion, rather than having the parameters be embedded in the program code.

J2EE PATTERNS AND J2EE PLATFORM

As you can see from the overview, the J2EE platform standardizes a number of different technologies to provide a robust platform for building distributed multitier enterprise class applications. The J2EE platform is built on the J2SE platform. Since the J2SE platform forms the foundation of the J2EE platform, a Java developer can learn the J2EE technologies with relative ease.

However, there is a belief that learning a new technology by itself is sufficient to make us adept at designing systems based on that new technology. We respectfully disagree with this. We believe that in addition to learning the technology, we need other insights to build successful systems. Patterns can help facilitate the process of knowledge accumulation and knowledge transfer. Patterns help us to document and communicate proven solutions to recurring problems in different environments. Using patterns effectively, we can prevent the "re-invent the wheel" syndrome.

Our J2EE patterns are derived from our experience with the J2EE platform and technologies. The J2EE patterns described in this book address different requirements spread across all the J2EE tiers. In our tiered approach (see " " on page†), we have modeled the J2EE multiple tiers as five tiers: client, presentation, business, integration, and resource tiers. This model allows us to logically separate responsibilities into individual tiers. In our model, for example, we separate the EIS tier into an integration tier and a resource tier. By doing so, we make it easier to separately address the requirements of integration and resources. Thus, the tiers in our model are a logical separation of concerns.

We've categorized the J2EE patterns described in this book into three of these five tiers—presentation, business, and integration. In our opinion, the client and resource tiers are not direct concerns of the J2EE platform. The patterns related to servlets and JSP technologies are described in Chapter 7, "Presentation Tier Patterns." The patterns related to enterprise beans and JNDI technologies, and those related to bridging the presentation and business tier components, are described in Chapter 8, "Business Tier Patterns." Finally, the patterns related to JDBC and JMS technologies, aimed at bridging the business tier with the resource tier, are described in Chapter 9, "Integration Tier Patterns."

Because our most intensive work has been in these core areas of the J2EE platform, we currently do not address patterns other than these aforementioned technologies. We feel that the developer community gains a huge benefit if we first document the patterns in these core areas. We also believe that this categorization allows us to be flexible, and as new patterns are observed, we will categorize and document them.

We believe that these patterns will prove useful to you as they did to us and our fellow architects. They may be reused as solutions to the problems you may encounter during your J2EE design and architecture experience. We are also aware that patterns evolve over time, and we expect that our patterns are no exception. The patterns presented here have been refined many times. They have been written and rewritten to make them better. This process of evolution will continue.

SUMMARY

While this chapter provided an overview of the J2EE platform, it also included a flurry of terminologies and acronyms. If you are interested in learning more, the following online resources are recommended:

- The Story of the Java Platform—<http://java.sun.com/nav/whatis/storyofjava.html>
- Java Technology—An Early History—<http://java.sun.com/features/1998/05/birthday.html>
- Java Community Process—<http://java.sun.com/aboutJava/communityprocess/>
- J2SE Platform Documentation—<http://java.sun.com/docs/index.html>
- J2EE home page—<http://java.sun.com/j2ee>
- J2EE Blueprints—<http://java.sun.com/j2ee/blueprints/index.html>
- EJB home page—<http://java.sun.com/products/ejb>
- Servlets home page— <http://www.java.sun.com/products/servlet>
- JSP home page—<http://www.java.sun.com/products/jsp>
- JDBC home page—<http://www.java.sun.com/products/jdbc>
- JMS home page—<http://www.java.sun.com/products/jms>
- JNDI home page—<http://java.sun.com/products/jndi>
- Connector home page—<http://java.sun.com/j2ee/connector>

Aalpha Information Systems India Pvt. Ltd.

Give us a call at +91-836-4251105,

E-mail: info@aalpha.net

or visit our website www.aalpha.net for more information
on how we can help you with your next project.