6. JAVA, Distributed Computing & J2EE; Design and Development of J2EE Application

What is a Java Web Application?

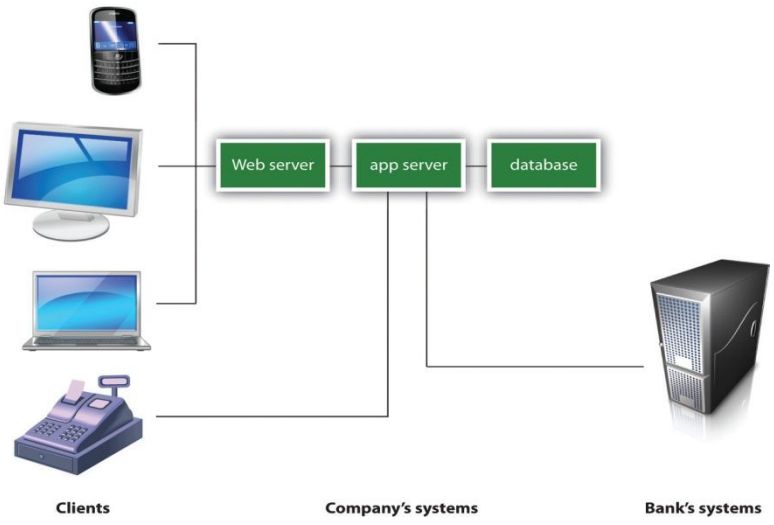
A **Java web application** generates interactive web pages containing various types of markup language (HTML, XML, and so on) and dynamic content. It is typically comprised of web components such as JavaServer Pages (JSP), servlets and JavaBeans to modify and temporarily store data, interact with databases and web services, and render content in response to client requests.

Because many of the tasks involved in web application development can be repetitive or require a surplus of boilerplate code, web frameworks can be applied to alleviate the overhead associated with common activities. For example, many frameworks, such as JavaServer Faces, provide libraries for templating pages and session management, and often promote code reuse.

Java has strong support for web development. Java is frequently used at the server side. A Java web application can be deployed as a **WAR (Web ARchive)** file. A WAR file is a zip file which contains the complete content of the corresponding web application.

Distributed Computing

Distributed computing is a field of computer science that studies distributed systems (A *distributed system* is a software system in which components located on networked computers communicate and coordinate their actions by passing messages). The components interact with each other in order to achieve a common goal. Distributed computing is any computing that involves multiple computers remote from each other that each has a role in a computation problem or information processing.

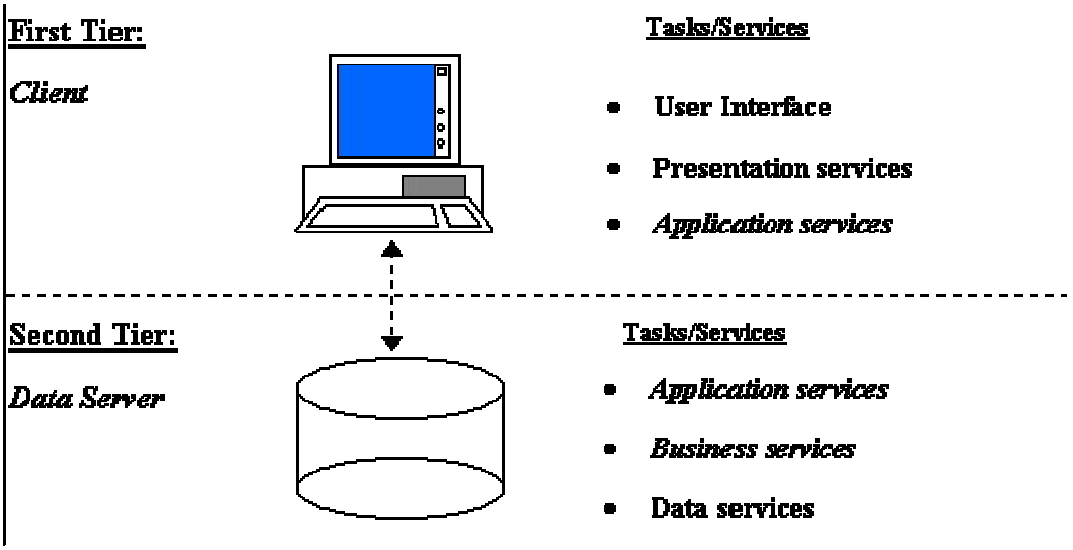


Past History

- Initially two tier architecture (client server applications)
- Client is responsible for data access applying business logic and presentation of data
- Only service provided by Server was that of database server.

Two-Tier Application Architecture

A two-tier architecture is a software architecture in which a presentation layer or interface runs on a client, and a data layer or data structure gets stored on a server. Separating these two components into different locations represents two-tier architecture, as opposed to single-tier architecture.



Drawbacks

- Easy to deploy but difficult to enhance or upgrade.
- It makes reuse of business and presentation logic difficult
- Not scalable and not suited for internet

Java 2 Platform Enterprise Edition (J2EE)

J2EE is architecture for implementing enterprise class applications using Java and Internet Technology. It solves problems of two tier architecture. It is used to develop n tier application. It supports the development of a variety of application types

- small client server systems
- Systems running on Intranets
- Systems on large scale internet e-commerce site

Java EE (Enterprise Edition) is a widely used platform containing a set of coordinated technologies that significantly reduce the cost and complexity of developing, deploying, and managing multi-tier, server-centric applications. Java EE builds upon the Java SE

platform and provides a set of APIs (application programming interfaces) for developing and running portable, robust, scalable, reliable and secure server-side applications.

J2EE Tiers (Packaging of J2EE Applications)

J2EE architecture supports component-based development of multi-tier enterprise applications. A J2EE application system typically includes 3 tiers. J2EE in like fashion runs on this n-tier environment, typically consisting of a client tier, a middle tier app server and multiple servers at the third tier providing services such as data and possibly legacy functionality.

Client tier

In the client tier, Web browsers or standalone application clients are included. The J2EE BluePrint document recommends using Web Browsers as clients whenever possible.

Middle tier

Consists of two subtiers:

1) ***Web Tier:*** The J2EE BluePrint document recommends using JSPs (with supporting servlets) to provide the core of the user interface of your application.
2) ***EJB Tier (Business Tier):*** Here is where the business logic, including data access, resides.

Enterprise Information System tier

The back-end databases and other information sources.

Containers are the heart of the J2EE component model. Containers are the runtime environment provided by J2EE platform providers. J2EE applications are delivered in archive files.

J2EE modules consist of one or more J2EE components of the same type and one component *deployment descriptor* (an XML document that describes how to assemble and deploy an application or application module in the runtime environment).

Module	***Description***
Web Modules	Consists of JSP files, classes for servlets, HTML or XML files, a deployment descriptor and graphics files, all in a Web Archive (WAR) file, which is a JAR file with the .WAR extension.
EJB Modules	Consist of EJB classes and interfaces, plus a deployment descriptor, in a JAR file with the .JAR extension.
Application Client Modules	Consists of class files and a deployment descriptor, in a JAR file with the .JAR extension (included only if the application is providing a standalone Java Client).

J2EE Components & Services

The J2EE platform consists of J2EE components, services, Application Programming Interfaces (APIs) and protocols that provide the functionality for developing multi-tiered and distributed Web based applications.

A J2EE **component** is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and communicates with other components. The J2EE specification defines the following J2EE components:

Component type	Components	Packaged as
Applet	applets	JAR (Java Archive)
Application client	Client side Java codes.	JAR (Java Archive)
Web component	JSP, Servlet	WAR (Web Archive)
Enterprise JavaBeans	Session beans, Entity beans, Message driven beans	JAR (EJB Archive)
Enterprise application	WAR, JAR, etc	EAR (Enterprise Archive)
Resource adapters	Resource adapters	RAR (Resource Adapter Archive)

A **service** is a component that can be used remotely through a remote interface either synchronously or asynchronously (e.g. Web service, messaging system, sockets, RPC etc).

J2EE Application Components

J2EE specification defines the following components:

- Client Components – application clients and applets
- Web Components – Java Servlet and JavaServer Pages (JSP) technology
- Business Components – Enterprise JavaBeans (EJB) components

These components are written in the Java programming language and compiled in the same manner as any other program written in Java.

Client Components

A J2EE application can be either be Web Based or non-Web-based. Non-Web-based components are an extension of the heretofore common client server applications. In a Non-Web-based J2EE application, an application client executes on the client machine. For a Web-based J2EE application, the Web browser downloads Web pages and applets to the client machine.

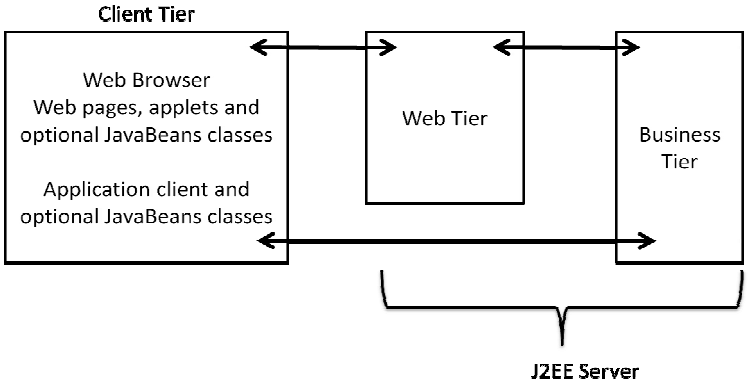
- ***Application Clients***: These run a client machine, providing a way for users to handle tasks such as J2EE system or application administration. There is usually a graphical user interface (GUI) created using Swing or Abstract

Windowing Toolkit (AWT) APIs; a command line interface is also possible. These directly access enterprise beans that run in business tier.

- ***Web Browsers***: The user's Web Browser downloads static and dynamic Hypertext Markup Language (HTML), Wireless Markup Language (WML), eXtensible Markup Language (XML) or pages in other formats from the Web tier. Servlets and JSP pages running in the Web tier will generate dynamic Web pages.
- ***Applets***: Web pages downloaded from the Web tier can include embedded applets. These are small client applications, written in the Java language, which execute in the JVM installed in the Web browser. Client systems often need additional Java plug-in for the applet can successfully execute in the Web Browser.
- ***JavaBeans Component Architecture***: The client tier sometimes includes a component based on JavaBeans component architecture for managing data flow between the application client or applet and components running on the J2EE server. The J2EE specification does not regard JavaBeans as components. JavaBeans components have instance variables as well as get and set methods for accessing the data in those instance variables. They tend to be simple in design and implementation.

J2EE Server Communication

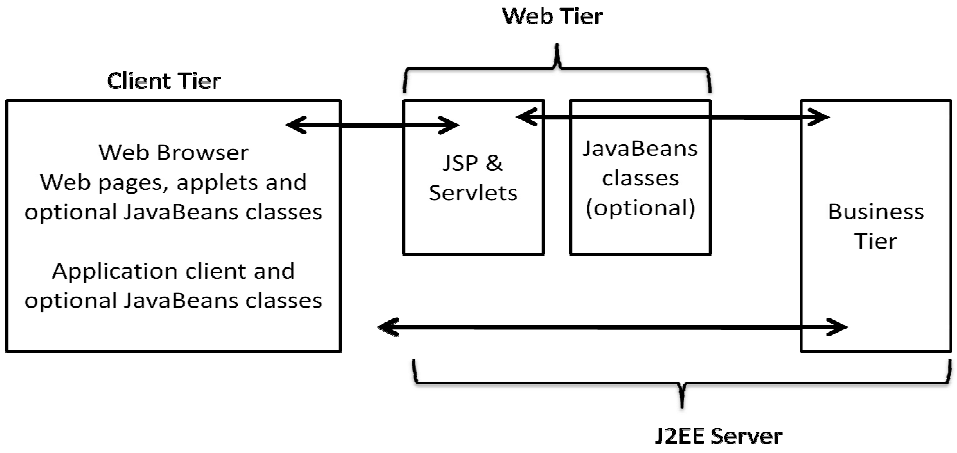
Client communicates with the business tier running on the J2EE server either directly (as in case of client running in browser) or by going through JSP pages or servlets running in the Web Tier.



Web Components

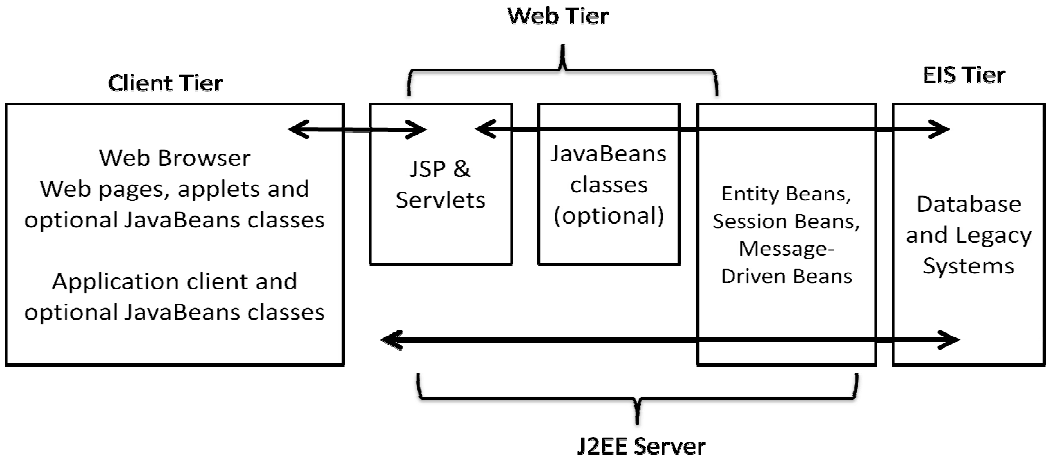
J2EE Web components are either JSP pages or servlets. Servlets are Java classes that dynamically process requests and construct responses. JSP pages are text-based documents containing static content along with snippets of Java code in order to generate dynamic content. When a JSP page loads, a background servlet executes the code snippets, returning a response.

Although static HTML pages and applets are bundled with Web Components during application assembly, they are not considered Web Components by the J2EE specification. The Web Tier might include JavaBeans objects for managing user input, sending that input to enterprise beans running in the business tier to be processed.



Business Components

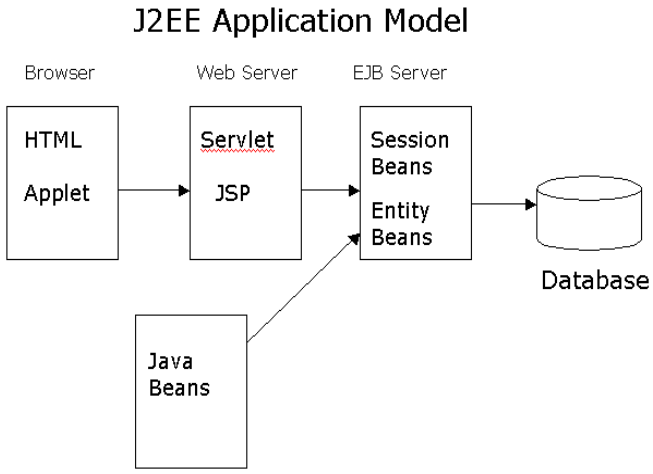
Business code is logic that solves the functional requirements of a particular business domain such as banking, retail or finance. This code is handled by enterprise beans that run in the business tier. The figure demonstrates how an enterprise bean receives data from client program, processes it, and then sends it to the enterprise information system tier to be stored. In addition, an enterprise bean retrieves data from storage, processes it, and then sends it back to the client program.



Enterprise Information System Tier

The EIS tier is giant "catch-all" handling EIS software. It includes enterprise infrastructure systems such as Enterprise Resource Planning (ERP), mainframe transaction processing, database systems and other legacy information systems. J2EE application components access enterprise information systems for functions such as database connectivity.

J2EE Application Model



Browser is able to process HTML and applets pages. It forwards requests to the web server, which has JSPs and Servlets. Servlets and JSPs may access EJB server. Java Standalone runs on java client, which access EJB server using RMI.

Servlet

- A Java servlet is a Java programming language program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on Web servers.
- Such Web servlets are the Java counterpart to other dynamic Web content technologies such as PHP and ASP.NET. It interacts with web client using response request paradigm.

JSP

- JavaServer Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.
- A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application.
- Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands. Using JSP and Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

EJB

- *Enterprise Java beans architecture is a component model for development and deployment of distributed business application.*
- Entity Beans
  - Represent persistent business Entity
  - Persisted in storage system (usually Database),
  - Might contain Application logic intrinsic to entity
- Session Beans
  - Perform work for individual clients on the server
  - Encapsulate complex business logic
  - Can coordinate transactional work on multiple entity beans

Requirements of Web Architecture

The web architecture required for J2EE is analogous to the architecture required to run vendor-based SQL database servers. The qualities of performance, reliability and security must be present for Web application servers to provide a host for an application.

Requirements

- The Speed to Compete
- Service Availability
- Connecting to Existing Data
- Expanded User Definition
- Flexible User Interaction
- Flexible Business Component Model

The Speed to Compete

Speed is key. The competition will win out every time if it is able to provide faster response to the client. The user can click away to a competitor if a response is too slow. This is a difficult task because the user base can change rapidly. J2EE application servers need to be efficient and scalable.

Service Availability

Users want the application to be available 24x7. This is the attraction of doing business on the Web. Users don't have to worry about the doors being closed after hours. The business depends on the application being up and ready to serve. J2EE application server vendors must provide reliable server configurations. J2EE application server vendors also must consider privacy issues.

Connecting to Existing Data

Specialized access to enterprise resource planning and mainframe systems such as IBM's CICS and IMS will be provided in future version of J2EE through the Connector architecture. These systems is highly complex & specialized, each requires unique tools and support to ensure utmost simplicity to application developers.

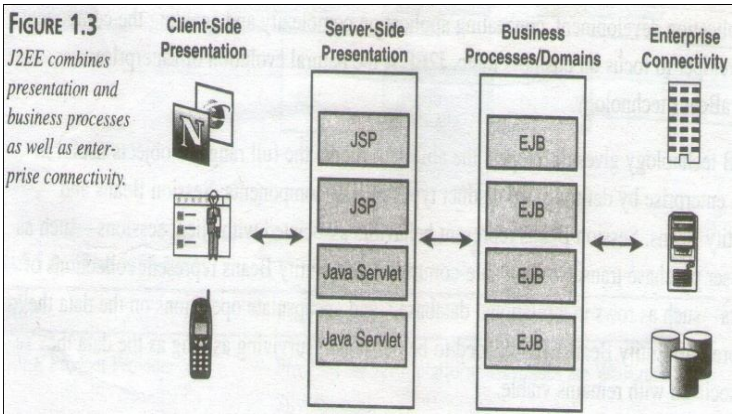


FIGURE 1.3 *J2EE combines presentation and business processes as well as enterprise connectivity.*

Expanded User Definition: Customers, Employees and Partners

A Desktop in past was the sole means of interfacing with an enterprise system. Users today want to connect from virtually anywhere. The access begins during their commute and might continue through the workday and while traveling to remote business sites.

Flexible User Interaction

J2EE provides choices for GUI across a company's intranet or on the WWW. Clients can run on desktops, laptops, PDAs, cell phones, and other devices. Pure client-side user interfaces can use standard HTML and Java Applets. For server-side deployment of dynamic content, J2EE supports both the Java Servlets API and JSP technology.

Flexible Business Component Model

EJB (Enterprise JavaBeans) technology has developed significant momentum in the middleware marketplace. It enables a simplified approach to multitier application development, concealing application complexity and enabling the component developer to focus on business logic.

Web Application Life Cycle

One of the strengths of the J2EE platform is that the implementation process if divided naturally into roles.

- Multiple Developer Roles
- Iterative Development
- Simplified Architecture and Development
- Maps easily to Application Functionality
- Component-Based Architecture
- Support for Client Components
- Support for Business Logic Components

Multiple Developer Roles

With the set of features designed specifically to expedite the process of distributed application development, the J2EE platform offers several benefits, but requires additional developer roles.

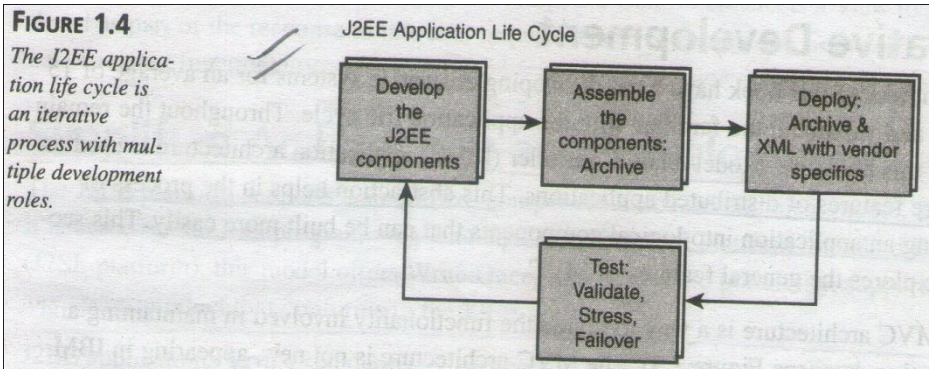


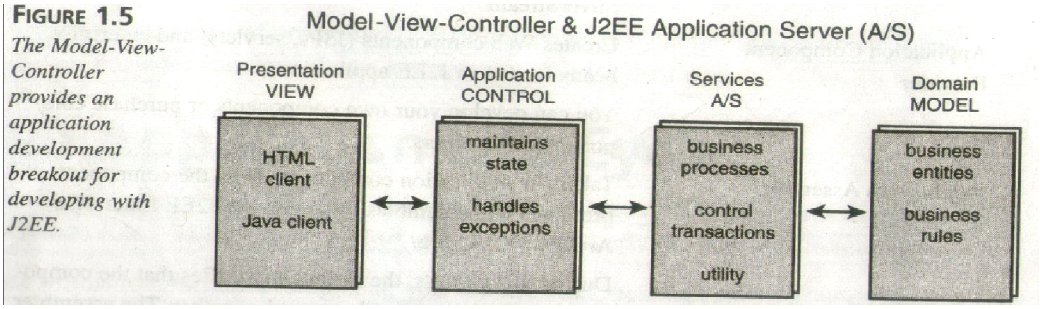
FIGURE 1.4 *The J2EE application life cycle is an iterative process with multiple development roles.*

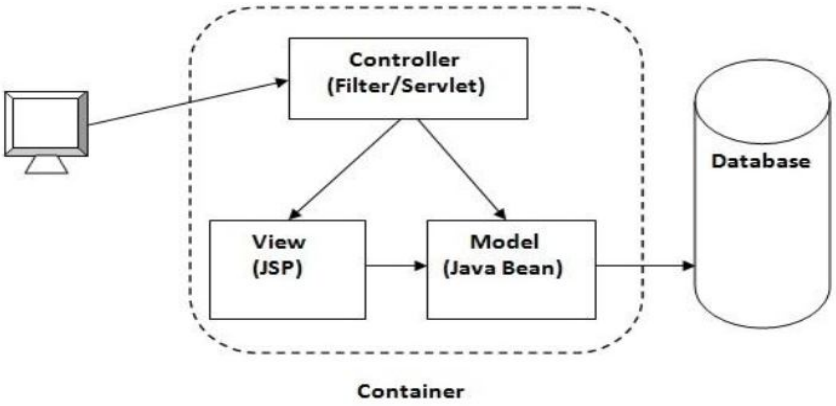
J2EE Roles

- ***J2EE Product Provider:*** Provides J2EE Platform
- ***Application Component Provider:*** Creates Web Components (JSPs, Servlets)
- ***Application Assembler:*** Takes the application components from the CPs and assembles them into J2EE Enterprise Archive.
- ***Deployer:*** Deploys the application in the runtime environment
- ***System Administrator:*** Configures and administers the runtime environment
- ***Tool Provider:*** Provides J2EE development, assembly and deployment tools.

***Iterative Development (*MVC APPLICATION ARCHITECTURE)**

MVC (Model-View-Controller) architecture is used to analyze features of distributed applications. This helps in the process of dividing an application into logical components that can be built more easily. This architecture is a way to divide the functionality involved in maintaining and presenting data.





Model: The model represents the state (data) and business logic of the application.
View: The view module is responsible to display data i.e. it represents the presentation.
Controller: The controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.

- **Advantage of MVC Architecture**
  - Navigation control is centralized now only controller contains the logic to determine the next page.

- Easy to maintain
- Easy to extend
- Easy to test
- Better separation of concerns

- **Disadvantage of MVC Architecture**
  - We need to write the controller code self. If we change the controller code, we need to recompile the class and redeploy the application.

Simplified Architecture and Development

The J2EE platform supports a simplified, component-based development model as it is based on Java Programming Language.

- J2EE applications have a standardized, component based architecture.
- J2EE applications are distributed and multitier; provides server-side and client-side support for enterprise applications.
- J2EE applications are standards-based and portable, defines standard APIs which all J2EE compatible vendors support.
- J2EE applications are scalable. They run in containers which are part of a J2EE server.
- J2EE applications can be easily integrated with back-end information systems.

Maps easily to Application Functionality

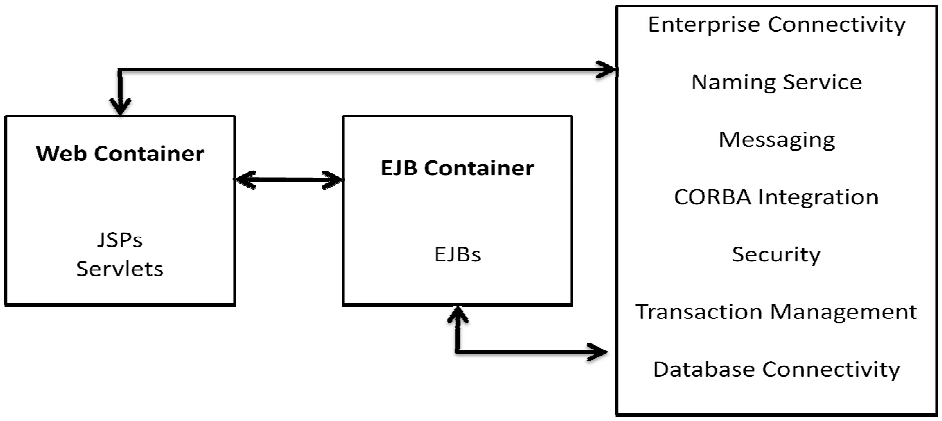
Component-based application models map easily and with flexibility to the functionality desired from an application. The J2EE platform provides a variety of ways to configure the architecture of an application, depending on the factors such as client types required, level of access required to data sources and other considerations. Component-based design also simplifies application maintenance as components can be updated and replaced independently.

Component-Based Architecture

Central to J2EE component-based development model is the notion of containers. Containers are standardized runtimes environments that provide specific component services. Components can expect these services to be available on any J2EE platform from any vendor.

For example, all **J2EE Web Containers** provide runtime support for responding to client requests, performing request-time processing (such as invoking JSP or servlet behavior) and returning results to the client. All **EJB containers** provide automated support for transaction and life cycle management of EJB components, as well as bean lookup and other services.

Containers also provide standardized access to enterprise information systems; for example, providing RDBMS access through the JDBC API.



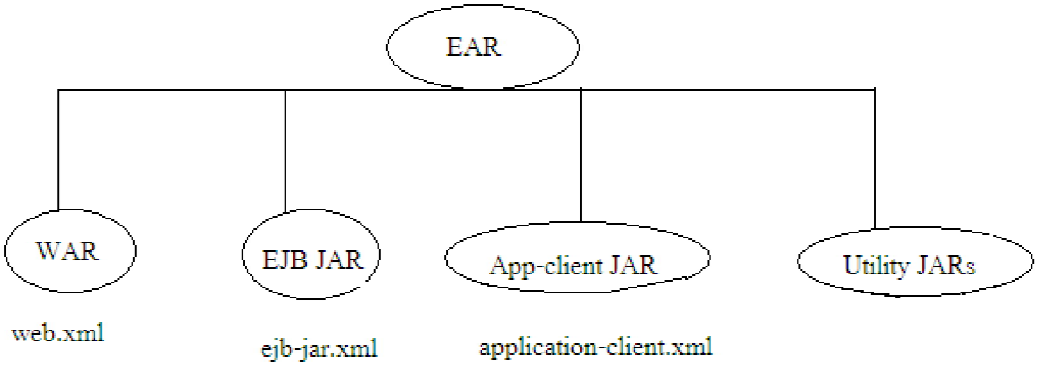
Support for Client Components

The J2EE client tier provides support for a variety of client types, both within the enterprise firewall and outside. Clients can be offered through Web Browsers by using plain HTML pages, dynamic HTML generated with JavaServer Pages technology or Java applets. Clients can also be offered as standalone Java language applications. J2EE clients are assumed to access the middle tier primarily using Web standards namely HTTP, HTML and XML.

Support for Business Logic Components

In the J2EE platform, business logic is implemented in the middle tier as Enterprise JavaBeans components (EJBs). Enterprise beans enable the component or application developer to concentrate on the business logic while the complexities of delivering a reliable, scalable service are handled by the EJB server.

The J2EE application is packaged in an archive or "zip" file known as an Enterprise Archive (EAR). The EAR contains the Web, EJB and client components. The Web, EJB and Client components are encased in their own archive files (WAR, JAR and CAR respectively).

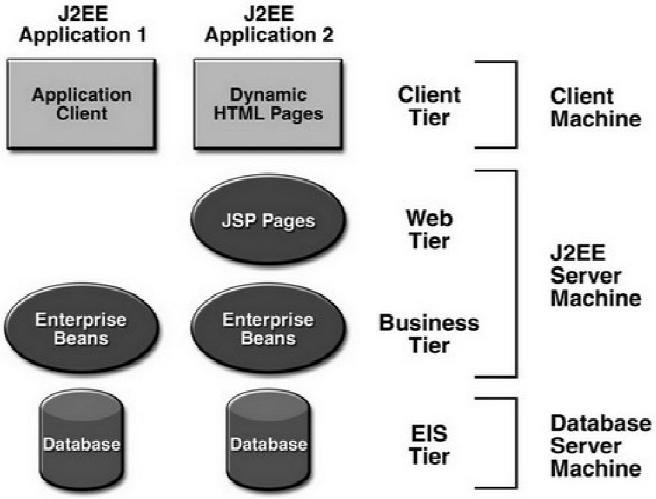


J2EE Layers

The J2EE platform uses a multitiered distributed application model, where application logic is divided into components according to its function. The various components that a J2EE application consists of are installed on different machines. Two multitiered J2EE application divided into the tiers described here:

- Client tier components run on the client machine
- Web tier components run on the J2EE server
- Business tier components run on the J2EE server
- Enterprise information system (EIS) tier software runs on the EIS server.

J2EE multitiered applications are generally considered to be three-tiered applications because they are distributed over three different locations: client machines, J2EE server machine, and the database or legacy machines at the back end.



The Client Layer

The client layer of web application is implemented as a Web browser running on the user's machine. Its function is to display data, providing the user with a place to enter and update data. Generally, one of two common approaches is used for building the client layer:

- A ***pure HTML-only client*** – In this scenario, virtually all of the intelligence is placed in the middle tier. When the user submits the Web pages, all the validation is done on the J2EE server (middle tier). Errors are then posted back to the client. The pure HTML approach is less efficient for end users because all operations require the server for even the most basic functions. The argument in favor of this approach is that it provides a better separation of business logic and presentation.
- A ***Hybrid HTML/Dynamic HTML (DHTML)/JavaScript client*** – In this scenario, some intelligence is included in the Web pages, which run on the client. The client

will do some basic validations. The hybrid client approach is more user-friendly, requiring fewer trips to the server. Typically, DHTML and JavaScript are written to work with more recent versions of mainstream browsers.

The Presentation Layer

The presentation layer generates Web pages and any dynamic content in the Web pages. The dynamic content is typically obtained from a database; for example, content may consist of a list of transactions conducted over the last month. This layer major job is to package requests contained on the Web pages coming back from the client.

The presentation layer can be built with a number of different tools. The presentation layers for the first Web sites were built as *Common Gateway Interface (CGI)* programs. Netscape servers also offered server-side JavaScript for Web sites. Contemporary Web sites generally have presentation layers built using the Microsoft solution *Active Server Pages (ASP)* or Java solution, which utilizes some combinations of *Servlets and Java Server Pages (JSP)*.

Tools provide methods to facilitate embedding dynamic content inside other static HTML in the web page. The presentation layer is generally implemented inside a ***Web Server***. The Web Server typically handles request for several applications in addition to requests for the site's static Web pages. The Web server knows which application to forward the client-based request to (or which static Web page to serve up).

The Business Logic Layer

The bulk of the application logic is written in the business logic layer. The challenge here is to allocate adequate time and resources to identify and implement this logic. Business logic includes:

- Performance of all required calculations and validations
- Workflow management (including keeping track of session data)
- Management of all data access for the presentation layer.

In modern Web applications, business logic is frequently built using the Java solution, with EJB that are built to carry out the business operations. Language-independent Common Object Request Broker Architecture (CORBA) objects can also be build and accessed effortlessly with a Java presentation tier. The main component of CORBA is the Object Request Broker (ORB) which encapsulates the communication infrastructure necessary to locate objects, manage connections and deliver data.

Much like the presentation layer, the business logic layer is generally implemented inside the application server. The application server automates many services such as transactions, security, persistence/connection pooling, messaging and name services.

The Data Layer

The data layer is responsible for data management. A data layer may be as simple as a modern relational database; on the other hand, it may include data access procedures to other data sources such as hierarchical databases or legacy flat files. The data layer provides the business logic layer with required data when needed and stores data when requested.

Development Methodology and Process

A J2EE application is assembled from two different types of modules: enterprise bean and web components. Both of these modules are reusable; therefore, new applications can be built from pre-existing enterprise beans and components. The modules are also portable, so the application that comprises them will be able to run on any J2EE server conforming to the specification.

Modeling Tools

Modeling is a visual process used for constructing and documenting the design and structure of an application. The model is an outline of the application, showing its interdependencies and relationships between the components and subsystems. There are tools available to facilitate this process, showing a high-level view of many objects. Modeling involves the use and reuse of patterns. A pattern is commonly defined as a three-part rule that expresses a relationship between a certain context, a problem and a solution.

Development Tools

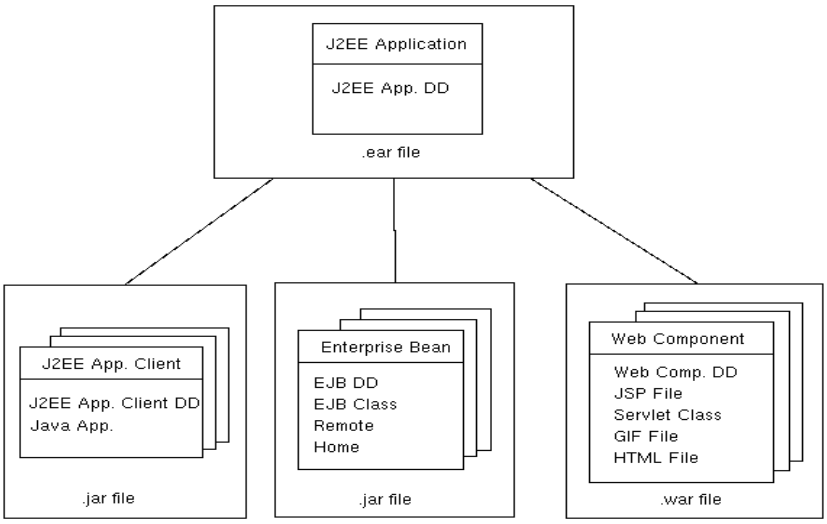
In order to be productive with technology such as J2EE, analysts and programmers will inevitably need visual development tools for building J2EE applications and components. When constructing a J2EE application, a developer must not only create Java code, but also build an archive file to house the classes and other supporting files including XML deployment descriptors and reference resolutions. This archive must then be deployed to a server and tested.

There are application frameworks that provide components and services based on the best patterns, practices and standards available. The ideal framework would implement extendable design patterns on the presentation, business, and data/services layers.

Contents of a J2EE Application

A J2EE application may contain one or more enterprise beans, Web components, or J2EE application clients. An enterprise bean is composed of three class files-- the EJB class, the remote interface, and the home interface. A Web component may contain files of the following types: servlet class, JSP, HTML, and GIF. A J2EE application client is a Java

application that runs in an environment (container) which allows it to access J2EE services.



Each J2EE application, Web component, and enterprise bean has a deployment descriptor (DD). A deployment descriptor is an .xml file that describes the component.

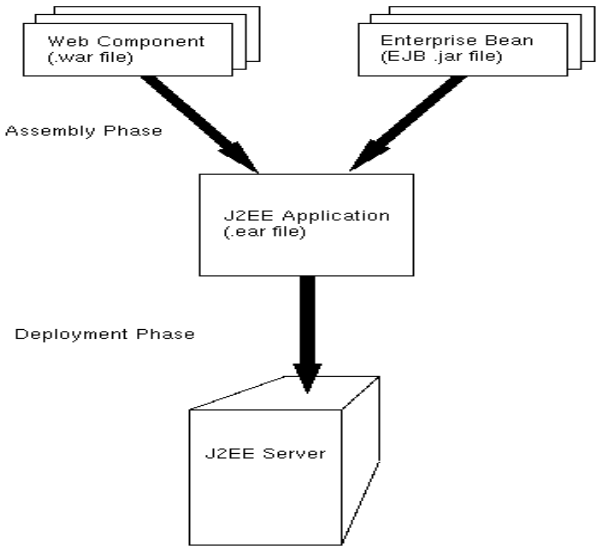
Files Used in a J2EE Application	
Element	**File Type**
J2EE Application	.ear
J2EE Application Deployment Descriptor	.xml
Enterprise Bean	ejb .jar
EJB Deployment Descriptor	.xml
EJB Class	.class
Remote Interface	.class
Home Interface	.class
Web Component	.war
Web Component Deployment Descriptor	.xml
JSP File	.jsp
Servlet Class	.class
GIF File	.gif
HTML File	.html
J2EE Application Client	.jar
J2EE Application Client Deployment Descriptor	.xml
Java Application	.class

Development Phases of J2EE Applications

As a J2EE application evolves, it passes through these development phases:

- Enterprise Bean Creation
- Web Component Creation
- J2EE Application Assembly
- J2EE Application Deployment

In a large organization, each phase might be executed by different individuals or teams. This division of labor works because each of the earlier phases outputs a portable file that is the input for a subsequent phase. For example, in the Enterprise Bean Creation phase, a software developer delivers EJB .jar files. In the J2EE Application phase, another developer combines these EJB .jar files into a J2EE application and saves it in an .ear file. In the final phase, J2EE Application Deployment, a system administrator at the customer site uses the .ear file to install the J2EE application into a J2EE server.



J2EE Application Assembly and Deployment

Because a J2EE application is not required to have both enterprise beans and Web components, you may skip one of the first two phases.

Enterprise Bean Creation

Person: software developer

Tasks:

- Codes and compiles the Java source code needed by the enterprise bean
- Specifies the deployment descriptor for the enterprise bean
- Bundles the .class files and deployment descriptor into an EJB .jar file

Deliverable: The EJB .jar file containing the enterprise bean

Web Component Creation

Persons: Web designer (JavaServer Pages components), software developer (servlets)

Tasks:

- Codes and compiles Java source code for the servlet
- Writes .jsp and .html files
- Specifies the deployment descriptor for the Web component
- Bundles the .class, .jsp, .html, and deployment descriptor files into the .war file

Deliverable: The .war file containing the Web component

J2EE Application Assembly

Person: software developer

Tasks:

- Assembles enterprise beans (EJB .jar) and Web components (.war) created in the previous phases into a J2EE application (.ear)
- Specifies the deployment descriptor for the J2EE application
- Verify that the contents of the EAR file are well formed and comply with the J2EE specification

Deliverable: The .ear file containing the J2EE application

Packager

The packager tool is a command-line script that allows you to package J2EE components.

Verifier

The verifier validates the J2EE component files. There are *three different ways* to run the verifier:

- From within the J2EE Reference implementations deploy tool utility.
- As a command-line utility
- As a standalone GUI utility

J2EE Application Deployment

Person: system administrator

Tasks:

- Adds the J2EE application (.ear) created in the preceding phase to the J2EE server
- Configures the J2EE application for the operational environment by modifying the deployment descriptor of the J2EE application
- Deploys (installs) the J2EE application (.ear) into the J2EE server

Deliverable: an installed and configured J2EE application

Task List for building J2EE Applications

Developing an application is one of those catch phrases that can mean different things. The following are the list of steps involved in J2EE application development:

1. Completing Prerequisite Tasks
2. Designing the Database
3. Creating Tables and Columns
4. Defining the Application
5. Creating a back-end Interface
6. Creating the interface
7. Building pages
8. Creating Data Access Objects

9. Validating your Code
10. Refining your Code

Completing prerequisite Tasks

The project team should have a development approach and an accompanying plan that includes skilled players to carry it to fruition. They will also need an application and perhaps a database logical model and physical manifestation of the model.

Logistic Prerequisites

The developers must decide on the separation of the application components. The developers must decide how to break out the functional requirements into logical tiers that promote satisfaction (e.g.: 2-Tier or 3-Tier).

Physical Prerequisites

Before we begin the physical construction of the application components, certain prerequisite physical items must be in place for use by the development team. Besides an adequate work station and the appropriate server(s), J2EE project libraries accessible to developers should also be available with appropriate permissions in place. Access to RDBMS application database with current maintenance and whatever third party or in-house J2EE development software is needed should be available from each workstation. Developers should be aware of the guidelines and naming standards, the project team has agreed to use to develop the database and the application.

Designing the Database

Determining the Application Entities

An entity is a person, place, object, event or activity that is relevant to the functionality we are creating. For instance, any noun that can represent information of interest to the organization is an entity. The attributes of an entity are the things that describe and define it.

Refining Each Entity and Attribute

After we have collected the business entities and attributes from interviews with the user, check the project standards for naming conventions. If applicable, use firm-wide abbreviations to ensure consistent naming. Determine the primary key for each entity. The primary key of an entity uniquely identifies entity instances (rows). The primary key of a dependent entity includes the parent key and descriptive column.

Determining Relationships

Listen for relationships in interviews with the user. Document the roles of entities in recursive relationships. A series may be a part of another series. Determine the

cardinality of the relationship. A relationship describes a business rule about two or more entities. Cardinality refers to a statement of how many of one item relate to another. A relationship may be optional or mandatory.

Creating Tables and Columns

Typically, when you are using the completed logical design, independent entities are cast as independent tables. Dependent entities become dependent tables. The designer should attempt to retain normalized sub-tables unless the cost of application-required table joins in unacceptable or merging with the super table creates few inapplicable nulls.

Choosing Data Types

The RDBMS has had real impact in the area of data type choices; there are a host of new data types such as date and time. Essentially, the SQL data types are

- Integer
- Floating-point number
- Character string (fixed or variable length)
- Day-time, time interval
- Numeric and decimal

Creating Keys

Besides the primary key and index, secondary keys and indices may be required. In a many-to-one relationship, place the foreign key in the many-side table. In a one-to-one relationship, place foreign keys in the table with fewer rows. The many-to-many relationship becomes an associative or junction table.

Completing Database Physical Design

To really know whether your design will perform you must list, examine and explain critical queries and transactions. Critical queries are high-volume, require quick response or are frequently executed. They are the points of reference for physical design. Create non-clustered indexes for other columns that are used to search or order the data.

Estimating the Size of the Database

It is good idea to find out how much traffic your database is expected to have for the first years of production. This will help in the physical design of the database and SQL access as well. The steps in this process are as follows:

1. Estimate the number of tables
2. Estimate the length of each row
3. Estimate the number of rows for each table
4. Procure additional storage, if needed.
5. Build a spreadsheet to estimate and finalize access requirements
6. Estimate the number of users.

7. Determine the user transaction types
8. Calculate the cost and frequency of each access.

Setting Up the Development Database Environment

It is also good practice to set up your development database to have simulated data for the first year of production loaded. This will help to certify the physical design of the database and SQL access as well. The steps in this process are as follows:

1. Create the database, tables, indices, and user permissions.
2. Use ERwin to generate the database schema.
3. Use RDBMS Enterprise Manager to administer and maintain database objects.
4. Build batch and online procedures to populate and access the database.
5. Use flat-file extracts to load the initial test bed of data.
6. Develop procedures for periodic mass updates (embedded SQL, Java, J2EE session and entity beans).
7. Develop OLTP using stored procedures and SQL.
8. Build ad hoc query tools (dynamic data pages).
9. Procure warehousing tools.
10. Use the DBMS EXPLAIN utility for all SQL with problematic performance.
11. Refine the database physical design.
12. Add secondary indexes to aid access and improve performance.
13. Repartition and relocate physical components.

Defining the Application

Defining the application is to create the application libraries and the point of entry. Defining the application not only includes setting up and defining the application object, but also establishing the rules of interaction between developers.

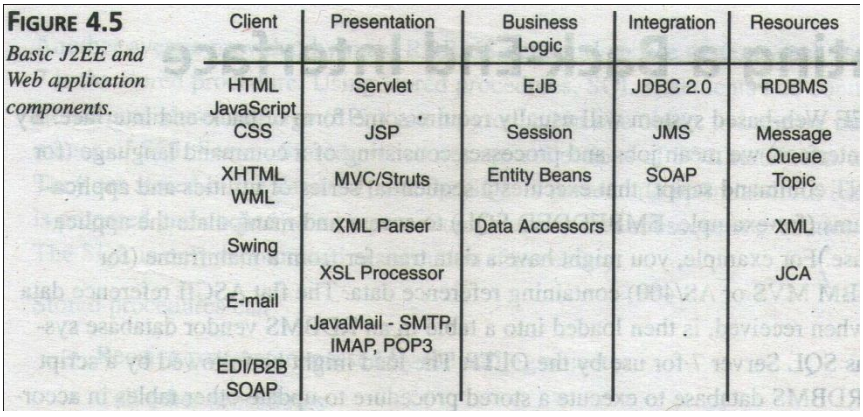


FIGURE 4.5 *Basic J2EE and Web application components.*

Creating a Back End Interface

Even J2EE Web-based system will usually require some form of back-end interface. By back-end interface, we mean jobs and processes consisting of a command language that executes a sequential series of utilities and application programs to access and manipulate the application database.

Creating the Interface

After we have developed the basic presentation/navigation of the application and received user approval, you can build the HTML and JSP pages and navigation/menus that present the data. You also develop the menu(s) or navigation pages that allow the users to move from page to page and perform application tasks. The user approved application presentation and navigation can and should be done at or near the beginning of the development cycle.

Building Pages

Pages are the main interface between the user and the application. Pages can display information from a user, and respond to mouse or keyboard actions. Pages can contain JSP custom tags, which are useful in defining new beans from a variety of possible sources, as well as a tag to render a particular bean to the output response. Java Servlets are designed to handle requests made by browsers & are designed to create pages that can turn static sites into live applications.

Creating Data Access Objects

Create data access objects to retrieve data from the database. This will facilitate the functions that format and validate data, analyze data analyze data through graphs and crossbars, create reports, and update the database. A data access object is typically an object that enables the user to display and manipulate database information using SQL statements in scripts or stored procedures for the particular function.

Validating our code

We can run our application at any time during development. If any problems are discover we can debug our application by setting breakpoints, stepping through our code statement by statement, and looking at variable and structure values during execution.

Refining Our Code

At some point in the application development, key developers will see that certain patterns are being repeated frequently within the application. The developers must respond and refine those parts of the application so that they platform optimally.

We should strive to accomplish the following:

- Code reusability
- Code modularity
- Reduced maintenance costs
- Improved consistency
- Improved performance

We will accomplish these objectives by carrying out the following tasks:

- Optimize data access paths.
- Remove redundant classes.
- Minimize the use of large bitmaps.
- Minimize or isolate commonly used server processing outside of page processing.
- Optimize libraries and classes.

Creating an Executable

J2EE components are packaged and bundled into J2EE application for deployment. Each component, its related files such as images & HTML files or server-side classes, and a deployment descriptor are assembled and added to the J2EE application. A J2EE application consists of one or more enterprise bean, web, or application client component modules. The final enterprise solution can use one or more J2EE applications depending on design requirements.