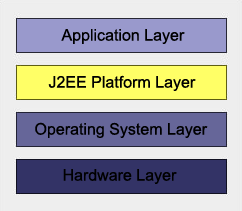
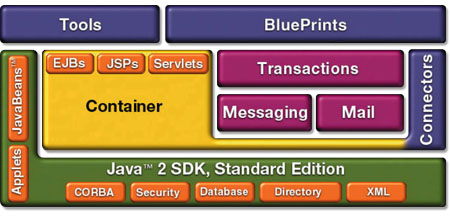
**1. History behind J2EE/EJB :**

* “J2EE describes the overall multi-tier architecture for designing, developing, and deploying component-based, enterprise-wide applications.” [Bass, Clements, Kazman, 406]
* J2EE composed of more than a dozen services (of which EJB is one) that function together to facilitate development of distributed applications

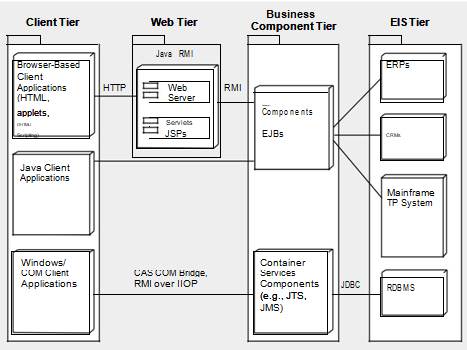


**2. J2EE architecture**

The major features of the J2EE platform are

* A multi-tiered distributed application model
* A server-side component model
* Built-in transaction control

A simple deployment view of the J2EE multi-tier model is given below



The role of each tier is as follows.

* *Client tier.* In a Web application, the client tier comprises an Internet browserthat submits HTTP requests and downloads HTML pages from a Web server. In an application not deployed using a browser, standalone Java clients or applets can be used; these communicate directly with the business compo-nent tier. (See Chapter 17 for an example of using J2EE without a browser.)
* *Web tier.* The Web tier runs a Web server to handle client requests andresponds to these requests by invoking J2EE servlets or JavaServer Pages

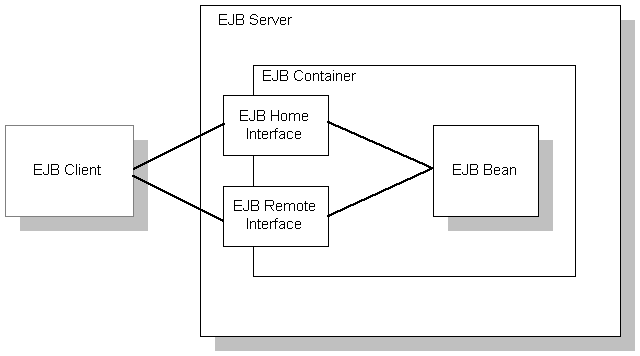
(JSPs). Servlets are invoked by the server depending on the type of user request. They query the business logic tier for the required information to satisfy the request and then format the information for return to the user via the server. JSPs are static HTML pages that contain snippets of servlet code. The code is invoked by the JSP mechanism and takes responsibility for for-matting the dynamic portion of the page.

* *Business component tier.* The business components comprise the core busi-ness logic for the application. They are realized by EJBs (the software com-ponent model supported by J2EE). EJBs receive requests from servlets in the Web tier, satisfy them usually by accessing some data sources, and return the results to the servlet. EJB components are hosted by a J2EE envi-ronment known as the EJB container, which supplies a number of services to the EJBs it hosts including transaction and life-cycle management, state management, security, multi-threading, and resource pooling. EJBs simply specify the type of behavior they require from the container at runtime and then rely on the container to provide the services. This frees the application programmer from cluttering the business logic with code to handle system and environmental issues.
* *Enterprise information systems tier.* This typically consists of one or moredatabases and back-end applications like mainframes and other legacy systems, which EJBs must query to process requests. JDBC drivers are typically used for databases, which are most often Relational Database Management Sys-tems (RDBMS).



3. **EJB architecture**

The EJB architecture looks like this:



* **EJB server**: The EJB server contains the EJB container, which provides the services required by the EJB component. EAServer is an EJB server.
* **EJB client:** An EJB client usually provides the user-interface logic on a client machine. The EJB client makes calls to remote EJB components on a server and needs to know how to find the EJB server and how to interact with the EJB components. An EJB component can act as an EJB client by calling methods in another EJB component.

An EJB client does not communicate directly with an EJB component. The container provides proxy objects that implement the components home and remote interfaces. The component’s remote interface defines the business methods that can be called by the client. The client calls the home interface methods to create and destroy proxies for the remote interface.

* **EJB container:**The EJB specification defines a container as the environment in which one or more EJB components execute. The container provides the infrastructure required to run distributed components, allowing client and component developers to focus on programming business logic, and not system-level code. In EAServer, the container encapsulates:
* The client runtime and generated stub classes, which allow clients to execute components on a remote server as if they were local objects.
* The naming service, which allows clients to instantiate components by name, and components to obtain resources such as database connections by name.
* The EAServer component dispatcher, which executes the component’s implementation class and provides services such as transaction management, database connection pooling, and instance lifecycle management.

**EJB component implementation** The Java class that runs in the server implements the bean’s business logic. The class must implement the remote interface methods and additional methods for lifecycle management.

**EJB component types**

We can implement three types of EJB components, each for a different purpose:

* Stateful session beans
* Stateless session beans
* Entity beans
* Stateful session beans

A stateful session bean manages complex processes or tasks that require the accumulation of data, such as adding items to a Web catalog’s shopping cart. Stateful session beans have the following characteristics:

* They manage tasks that require more than one method call to complete, but are relatively short-lived. For example, a session bean might manage the process of making an airline reservation.
* They typically store session state information in class instance data, and do not survive server crashes unless they are run in a cluster that has persistent storage enabled for the component.
* There is an affinity between each instance and one client from the time the client creates the instance until it is destroyed by the client or by the server in response to an expired instance timeout limit.

For example, if you create a session bean on a Web server that tracks a user’s path through the site, the session bean is destroyed when the user leaves the site or idles beyond a specified time

* Stateless session beans
* A stateless session bean manages tasks that do not require the keeping of client session data between method calls. Stateless session beans have the following characteristics:
* Method invocations do not depend on data stored by previous method invocations.
* There is no affinity between a component instance and a particular client. Each call to a client’s proxy can invoke a different instance.
* From the client’s perspective, different instances of the same component are identical.

Unlike stateful session beans, stateless session beans can be pooled by the server, improving overall application performance.

* Entity beans
* An entity bean models a business concept that is a real-world object. For example, an entity bean might represent a scheduled airplane flight, a seat on the airplane, or a passenger’s frequent-flyer account. Entity beans have the following characteristics:
* Each instance represents a row in a persistent database relation, such as a table, view, or the results of a complex query.
* The bean has a primary key that corresponds to the database relation’s key, and is represented by a Java datatype or class.

4. **Quality attributes, tactics and patterns(patterns you need to apply its not present in case study) used in the case study**

**5.Drawbacks of J2EE/EJB architecture**

 1 Requires application server  
 2 Requires only java client. For other language client, you need to go for webservice.  
 3 Complex to understand and develop ejb applications.  
 4 Complicated and large specification  
 5 Increased time of development  
 6 Complexities are added in comparison with straight Java classes  
 7 Potential to create a more essential, costly & complex solution  
 8 Continual revisions of the specifications  
 9 Lots of resources are used & have also lots of artifacts.

**6. Today’s technologies to overcome drawbacks of J2EE/EJB architecture(write in points)**

* list some new technology and how they improve the drawbacks of J2EE/EJB with the help of quality attributes
* **Spring** is also a container, but Spring can run in any java code (a simple main class, an applet, a web app or a JavaEE enterprise app). Spring can do almost everything EJB can do and a lot more, but I'd say it's most famous for dependency injection and non-intrusive transaction management
* **Hibernate** was the first big ORM (Object relational mapper) on the Java Platform, and as such has greatly inspired **JPA** (which is part of the EJB3 standard but can be used without an EJB container). I would suggest coding against JPA and only using hibernate as a provider, that way you can easily switch to EclipseLink etc.