BITS-PILANI M. TECH (SOFTWARE SYSTEMS) COURSE WORK

J2EE Architecture Evaluation

Bhaskara Srinivas Prayaga, 2018ht12553

11/15/2018

This document is an attempt to deep dive into J2EE Architecture and evaluate the benefits offered by the Platform while building Enterprise Systems

Table of Contents

1. INTRODUCTION

This document helps in understanding the following aspects of J2EE Architecture.

- *How did it evolve?*
- *Why does it exist?*
- *What are the Salient Features? (Quality attributes that it can address in Enterprise World).*

1.1 Evolution of J2EE

Over the years, the Java platform evolved into three major editions, each addressing a distinct set of programming needs:

A) The Java Platform, Standard Edition (Java SE)

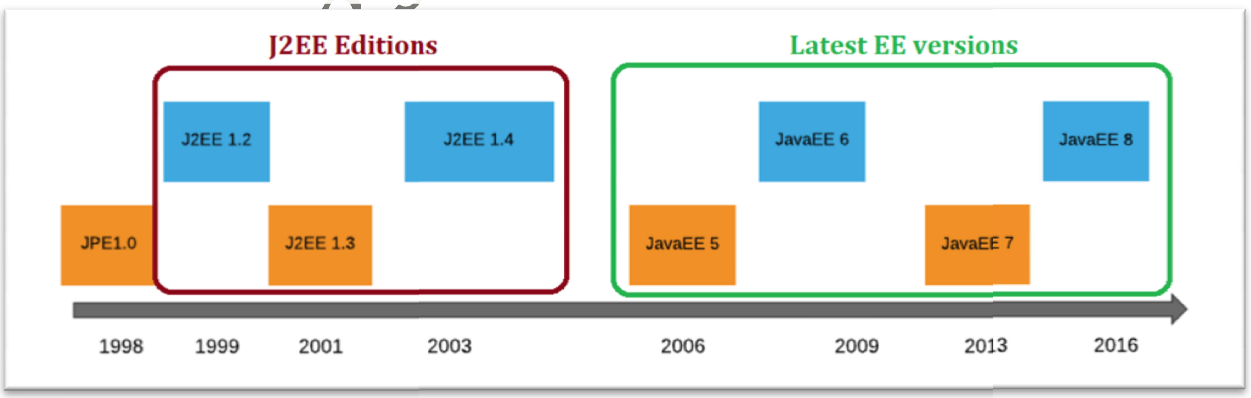
- Most commonly used among the three editions
- Used for developing desktop and console-based applications.
- It consists of:
  - A runtime environment
  - Set of APIs, to build a wide variety of applications comprising standalone applications that can run on various platforms- Windows, Linux, Solaris

B) The Java Platform, Enterprise Edition (Java EE)

- Used for building server-side applications, by using its component-based approach
- Widely Adopted across Enterprises

C) The Java Platform, Micro Edition (Java ME)

- Used for computing micro devices- handheld devices- PDAs, mobile phones – with limited display and memory support



Source: https://www.polyglotdeveloper.com/timeline/2015-09-22-Java-EE-timeline/

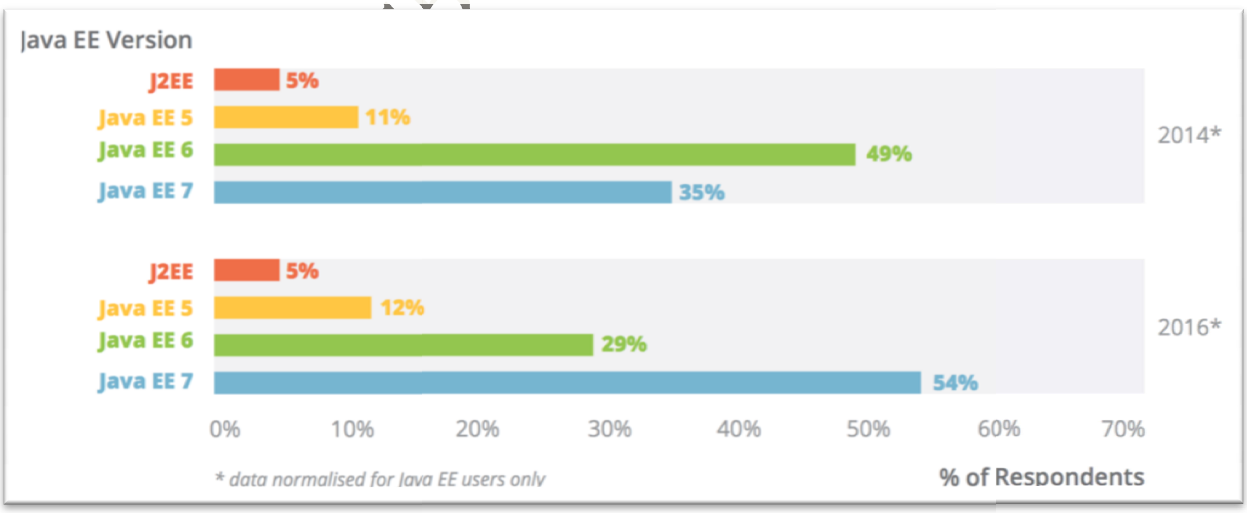
1.2 Overview

- Sun Microsystems, in developing J2EE, aimed to provide a basis for technology that supports the construction of such systems. In particular, as part of the J2EE specification, EJB aims to
  - ✓ provide a component-based architecture for building distributed object-oriented business applications in Java. EJBs make it possible to build distributed applications by combining components developed with tools from different vendors.
  - ✓ make it easier to write applications. Application developers do not have to deal with low-level details of transaction and state management, multi-threading, and resource pooling.

- More specifically, the EJB architecture does the following:
  - ✓ Addresses the development, deployment, and runtime aspects of an enterprise application's life cycle
  - ✓ Defines the contracts that enable tools from multiple vendors to develop and deploy components that can interoperate at runtime
  - ✓ Interoperates with other Java APIs
  - ✓ Provides interoperability between enterprise beans and non-Java applications
  - ✓ Interoperates with CORBA

- J2EE makes it possible to re-use Java components in a server-side infrastructure. With appropriate component assembly and deployment tools, the aim is to bring the ease of programming associated with GUI-builder tools (like Visual Basic) to building server applications. And, by providing a standard framework for J2EE products based on a single language (Java), J2EE component-based solutions are, in theory at least, product independent and portable between the J2EE platforms provided by various vendors

1.3 Adoption Rates



Source: https://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-2016-trends/

2. ARCHITECTURE

2.1 Enterprise Architecture Types

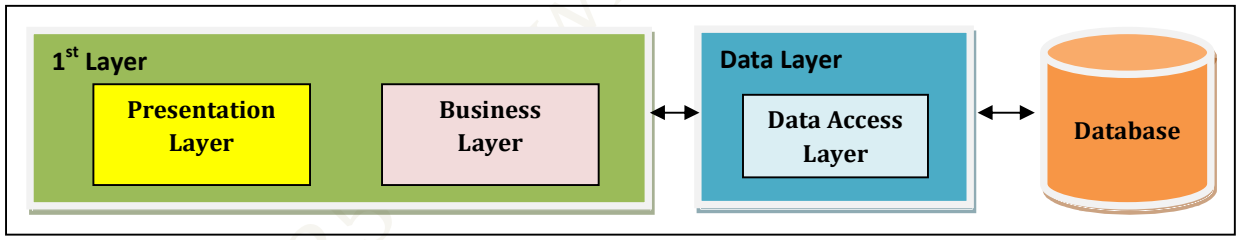
An enterprise application can be designed in many ways. But, here we discuss the following types of enterprise architectures:

- **Single-tier architecture**
  - It consists of the presentation, the business rules, and the data access layers in a single computing layer.
  - Such applications are relatively easy to manage, and implement data consistency, since data is stored in a single location

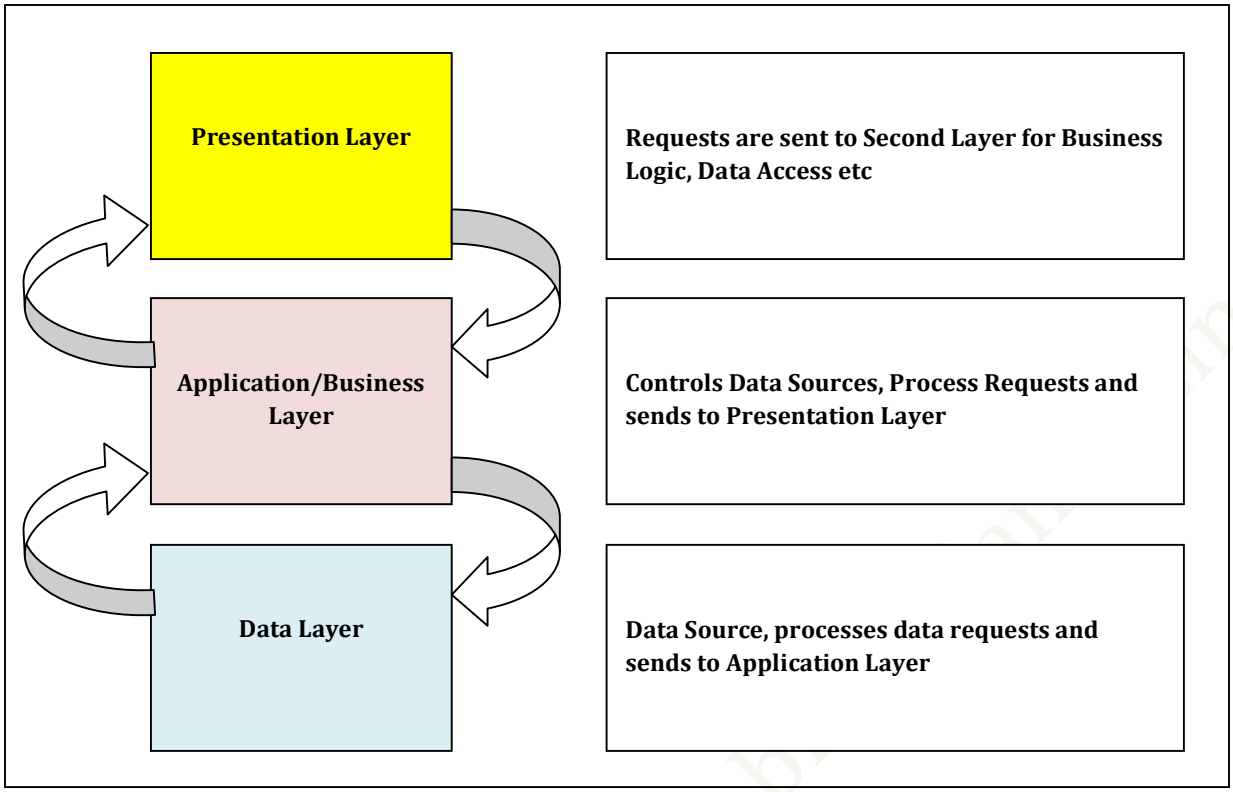
  Downside

  Such applications can't be scaled up to handle multiple users, and they do not provide an easy means of sharing data across enterprise

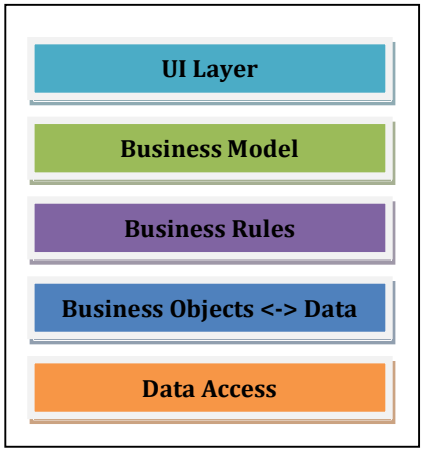
- **2-tier architecture**
  - It separates the Business Layer & Data Layer
  - The application resides entirely on the local machine and the database is typically deployed at a specific and secured location.
  - In 2-tier application (a. k. a. client-server application), the processing load is entrusted with the client, while the server simply controls the traffic between the application and data



- **3-tier architecture**
  - In 3-tier architecture, an application is virtually split into three separate logical layers
    - First tier: 'presentation layer'- consists of a GUI to interact with the user
    - Middle tier: 'business layer'- consists of the application (or business) logic
    - Third tier: 'data layer' – contains data access logic needed for the application.
  - This separation of application logic from the user interface adds enormous flexibility, as compared to the 2-tier system, to the design of an application.
  - By using this architecture, multiple user interfaces can be built and deployed without changing the application logic; provided it defines a clear interface to the presentation layer.



- **n-tier architecture**
  - There can be numerous layers in this kind of architectures.
  - In this model, the user interface logic is retrieved from the application running on the desktop.
  - The application on the desktop is responsible for presenting the user interface to the end user and for communicating to the business logic tier.
  - It is no longer responsible for enforcing business rules or accessing databases.
  - There is no hard & fast way to define an application in this architecture.
  - In fact, n-tier architecture can support a number of different configurations.



- Advantages:
  - ✓ Improved Maintainability
  - ✓ Consistency
  - ✓ Inter-Operability
  - ✓ Flexibility
  - ✓ Scalability

2.2 J2EE Architecture (Overview)

J2EE is a standards-based platform for

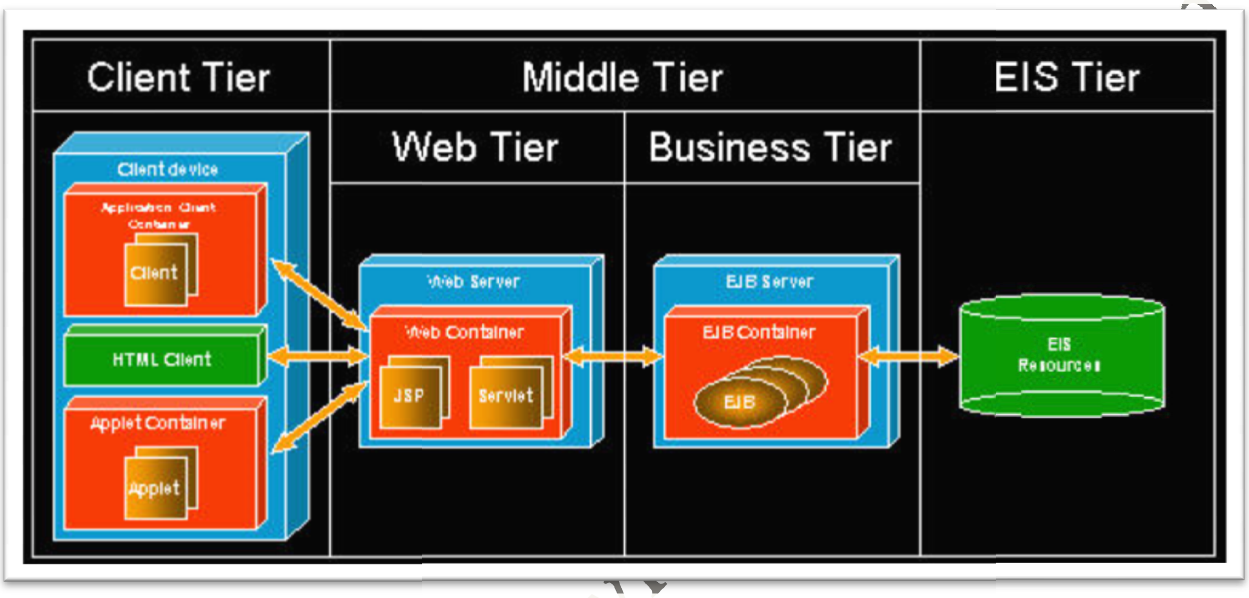
- Developing, deploying and managing multitier, web-enabled, server centric and component based enterprise applications
- J2EE enhances J2SE with:
  - Enterprise JavaBeans components
  - Java Servlets API
  - Java Server Pages
  - XML
- This suite is the Java 2 Enterprise Edition, commonly known as J2EE.
- J2EE (Java2 Enterprise Edition) offers a suite of software specification to design, develop, assemble and deploy enterprise applications.
- It provides a distributed, component-based, loosely coupled, reliable and secure, platform independent and responsive application environment.
- J2EE environment provides a framework for bundling together the components into an application and provide the applications necessary common services such as persistence, security, mail, naming and directory service etc.
- Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing component-based multi-tier distributed enterprise applications.
- A J2EE component is a self-contained functional software unit that is assembled into a J2EE application along with related components.
- Application logic is divided into components according to function

It comprises of the following:

- A runtime infrastructure for managing and hosting applications.
- All runtime applications are hosted in this server.
- A set of Java APIs to build applications.
- These Java APIs describe the programming model for Java EE applications.

2.3 J2EE Architecture (Deep Dive)

Java 2 Platform, Enterprise Edition (J2EE) uses a multi-tier distributed model. J2EE components run in, or are hosted by, J2EE containers generally provided as part of a commercial J2EE server. Containers provide a run-time environment and standard set of services (APIs) to the J2EE components running in the container, in addition to supporting the standard J2SE APIs.



Source: Internet

J2EE defines the following types of containers:

- **Application Client Container**
  - *A J2EE application client runs in an application client container, which supports these J2EE APIs: **JDBC**, **JMS**, **JAXP**, **JAAS**, **JavaMail**, **JAF**, **JSR**, **JAX-RPC**, **SAAJ**, **J2EE Management** and **JMX**.*
  - *Practically, application client containers consist of the standard J2SE installation. The application client container must support the JAAS callback handler interface to satisfy the security constraints of the rest of the enterprise application in the Web and EJB containers.*
- **Applet Container**
  - *An Applet runs in an applet container, which supports the applet programming model and supports standard J2SE APIs. Practically, applet containers are supplied as the Java plug-in to a Web browser*
- **Web Container**
  - *Web Components (JSP Pages & Servlets) run in a Web container provided as part of a J2EE server or provided as a standalone J2EE Web server.*
  - *A Web container supports the following J2EE APIs and packages: **JDBC**, **JMS**, **JAXP**, **JAX-RPC**, **JAXR**, **JAAS**, **Java Mail**, **JAF**, **J2EE Connector Architecture**, **JTA**, **JSR**, **SAAJ**, **J2EE Management**, **Java Servlet**, and **JSP***
- **EJB Container**
  - *EJB Components run in an EJB container, which is provided as part of a J2EE server.*
  - *An EJB container supports the following J2EE APIs and technologies: **EJB**, **JDBC**, **JMS**, **JAXP**, **JAX-RPC**, **JAXR**, **JAAS**, **Java Mail**, **JAF**, **JTA**, **JSR**, **SAAJ**, **J2EE Management**, and **J2EE Connector Architecture***

J2EE containers support all of the J2SE standard APIs, as well as a subset of J2EE APIs depending on the container type. Components within a container can access this available subset. The following table gives a brief description of each API and lists the J2EE containers where they are available.

#	Name	Description	App Client Container	Web Container	EJB Container
1	**EJB 2.1**	The EJB specification defines a component model for EJBs-business tier components thatautomatically support services such as remote communications, transaction management, security andpersistence	✓	✓	
2	**JAAS**	Java Authentication and Authorization Service (JAAS) provides services for authentication andauthorization of users to ensure they have permission to perform an action	✓	✓	✓
3	**JAF 1.0**	JavaBeans Activation Framework (JAF) provides services to identify data and instantiate a JavaBeanto manipulate that data	✓	✓	✓
4	**JAXP 1.2**	Java API for XML Processing (JAXP) provides an abstract interface for XML document processing thatcan be used with compliant parsers and transformers that use DOM SAX or XSLT	✓	✓	✓
5	**JAX-RPC 1.1**	The JAX-RPC specification defines client APIs for accessing web services as well as techniques forimplementing web service endpoints	✓	✓	✓
6	**Web Services for J2EE 1.1**	The Web Services for J2EE specification (JSR-109) defines the capabilities a J2EE application server must support for deployment of web service endpoints	✓	✓	✓
7	**SAAJ 1.2**	The SOAP with Attachments API for Java (SAAJ) provides the ability to manipulate SOAP messages	✓	✓	✓
8	**JAXR 1.0**	The Java API for XML Registries (JAXR) specification defines APIs for client access to XML-based registries such as WebXMLregistries and UDDI registries	✓	✓	✓
9	**JavaMail 1.3**	The JavaMail API provides a framework that can be extended to build Java-based mail applications	✓	✓	✓
10	**JDBC 3.0**	Java Database Connectivity (JDBC) is an API for accessing tabular data sources such as SQLdatabases, spreadsheets, and flat files	✓	✓	✓
11	**JMS 1.1**	Java Message Service (JMS) provides asynchronous messaging services for the transfer of data andnotification of events. With JMS, it is possible to use message-driven EJBs to asynchronouslyprocess messages delivered to JMS topics and queues	✓	✓	✓

12	**JNDI**	Java Naming and Directory Interface Specification (JNDI) provides naming and directory services toregister and lookup distributed components and resources. Clients only need to know the registeredJNDI name for the component or resource and don't need to know their actual network location. *Example: EJBs are registered in the enterprise directory at deployment time, using the deploymentdescriptor ejb-name field. J2EE clients look up an EJB using the JNDI lookup-allclients need to know is the name by which the EJB was registered in the directory. The JNDI lookupreturns a reference to the EJB?s home object*	✓	✓	✓
13	**JTA 1.0**	Java Transaction API (JTA) defines interfaces for managing distributed transaction services betweentransaction manager, resource manager, application server, and application		✓	✓
14	**J2EE Connector 1.5**	J2EE Connector Architecture Service Provider Interface (SPI) defines a standard for connecting EISresources to a J2EE container-an EIS-specific resource adapter (supplied by the EIS vendor) isplugged in to the J2EE container, extending the container so that it provides transactional, securesupport for that EIS. Components in the container can then access the EIS via the J2EE ConnectorArchitecture SPI		✓	✓
15	**JSP 2.0**	JavaServer Pages technology provides Web developers with the ability to create and maintain dynamicWeb pages. JSP pages are text-based and use XML-like tags to perform business logic and generatecustom content. JSP technology allows business logic to be delegated to other components so thatonly the presentation logic needs to be embedded in the JSP page		✓	
16	**Servlet 2.4**	Java Servlets extend the capabilities of the Web server to help build Web-based applications.Servlets are often used in interactive Web applications where the Web server responds to userrequests with dynamically generated content obtained from existing business systems		✓	
17	**RMI-IIOP**	Remote Method Invocation technology run over Internet Inter-ORB Protocol (RMI-IIOP) allows Javacomponents to communicate with legacy CORBA components written in other languages like C++ orSmalltalk	✓	✓	✓
18	**J2EE Management 1.0**	The J2EE Management API provides APIs for management tools to query a J2EE application server to determine its current status, applications deployed, and so on	✓	✓	✓
19	**JMX 1.2**	The Java Management Extensions (JMX) API is used by the J2EE Management API to provide some of the required support formanagement of a J2EE product	✓	✓	✓
20	**JACC 1.0**	The JACC specification defines a contract between a J2EE application server and an authorization policy provider	✓	✓	✓
21	**J2EE Deployment 1.1**	The J2EE Deployment API defines the interfaces between the runtime environment of a deployment tooland plug-in components provided by a J2EE application server		✓	✓

Key functionalities supported by EJB containers.

Remote Communications

- ✓ *EJB containers hide the complexity of remote communications from developers by using container-provided classes (generated by container tools when the EJB is compiled, along with **RMI stub classes** for the use of clients) that implement the EJB interfaces.*
- ✓ *These implementation classes are remote Java objects that a client can access using Java RMI. From the client's perspective, the client simply calls methods on the EJB interface, without any consideration of remote communications*

Concurrency

- ✓ *EJB containers transparently manage **concurrent requests** from multiple clients.*
- ✓ *Clients can act as if they have exclusive access to the EJB. For example, if two clients request the same entity EJB, the container provides each of them with their own instance and manages synchronization internally without the client's knowledge*

Naming

- ✓ *The EJB container provides a **JNDI name space** for locating EJBs deployed in the container. EJB clients can look up EJBs to obtain a Home interface.*
- ✓ *The **Home interface** for an EJB provides methods to find and create EJB instances. As long as the JNDI naming context is available from their location, clients can access the EJBs*

Persistence

- ✓ *EJB developers have the choice of two schemes for the storage of entity EJB persistent data: **Container Managed Persistence** (CMP) and **Bean Managed Persistence** (BMP).*
- ✓ *CMP delegates the responsibility for implementing the data access code to the container, whereas BMP leaves the EJB developer responsible for implementing that code.*
- ✓ *CMP allows the EJB developer to use a standard implementation for access to persistent storage simply by declaring container-managed fields in a deployment descriptor.*

Transaction Management

- ✓ *A transaction is a sequence of operations that succeeds or fails atomically-so that if any operation in the sequence fails, no change is made to the system state. For example, say you want to issue air tickets: you would validate a customer's credit card account, debit that account, and then issue the tickets. This sequence of operations should occur in a single transaction, so that if any operation fails, no change is made to the customer's credit card account and no tickets are issued.*
- ✓ *EJBs can use either **bean-managed transaction demarcation** or **container-managed transaction demarcation***
  - *Bean-managed transaction demarcation*
    - *In bean-managed transaction demarcation, you use a simple API to demarcate transaction boundaries.*
    - *This is the **Java Transaction API** (JTA), which you use to programmatically control transaction demarcation; for example, by calling the begin(), commit(), and rollback() methods of the JTA UserTransaction interface. The developer is responsible for coding rollback logic for transaction exception conditions, as the container does not handle this automatically.*
      Note: Entity EJBs cannot use bean-managed transaction demarcation-they can only use container-managed transaction demarcation
  - *Container-managed transaction demarcation*

- *In container-managed transaction demarcation, you don't supply code to begin and end transactions. Instead, you supply transaction attribute information in the EJB deployment descriptor for each method of your EJB. The transaction attribute (one of Required, RequiresNew, NotSupported, Supports, Mandatory, or Never) tells the container what transaction scope to use for the method. For example, if a client is running within a transaction and it calls a method of your EJB for which the transaction attribute is set to Required, then the method will be called within the scope of the existing transaction.*
- *Use container-managed transaction demarcation rather than bean-managed transaction demarcation whenever possible, so that you don't have to add, debug, and test transaction demarcation code in your component. Instead, the transaction behavior of each of your EJB methods is specified at deployment time, in the deployment descriptor. This means that the transaction behavior can be changed without an expensive code update-debug-test cycle.*

✓ *Distributed Transactions*

- *A distributed transaction is a transaction that must be coordinated across multiple databases and/or multiple applications. This is in contrast to a centralized transaction, such as a single J2EE application server committing transactions to a single database.*
- *A **two-phase commit** is necessary in distributed transactions; for example, where there is more than one database being updated. Some EJB containers (such as BEA WebLogic Server 6.0) supply support for two-phase commit, using **Open Group's XA protocol**. The application programmer does not need to write any code to handle the two-phase commit; the EJB container manages it.*

Security Management

✓ *EJB security is handled by the EJB container, using security information in the deployment descriptor. In the **deployment descriptor**, you declare a set of roles and, for each EJB method, you declare the roles that are authorized to call the method.*

✓ *At run-time, each client of the EJB is assigned to a role, and the EJB container manages access to the EJB's methods by checking that the client role is authorized to call that method.*

✓ *Since the security information is declared in the deployment descriptor, the security behaviour can be changed without an expensive code update-debug-test cycle*

Lifecycle Management

✓ *EJBs move through a series of states during their lifecycle in response to client requests. The EJB container is responsible for managing this lifecycle.*

✓ *At container startup, the container creates a pool of EJB instances in a resource pool (to save startup time when the EJB resource is needed). When an EJB client requests the creation of an EJB, an instance is assigned from the pool. The client can now make requests of the EJB. When an EJB client requests removal of an EJB, that instance is returned to the pool.*

✓ *The container notifies an EJB instance of various events in the EJB lifecycle, using a set of standard callback methods such as:*

- ***ejbCreate()** - called by the container after the EJB instance is created*
- ***ejbRemove()** - called by the container when the EJB instance is about to be deleted*
- ***ejbActivate()** - called by the container after the EJB instance is restored from a passive state*
- ***ejbPassivate()** - called by the container when the EJB instance is about to be passivated*
- ***ejbStore()** - called by the container when the EJB instance is about to be written to a database*
- ***ejbLoad()** - called by the container after the EJB instance fields are loaded from the database*

✓ *Each EJB is required to implement these callbacks although the EJB's implementation of the callback method is often empty. For example, the container calls the EJB's ejbRemove() method to notify the EJB that the EJB is about to be removed (there has been a client request to remove the*

EJB). In the EJB's ejbRemove() method, you would code any operations necessary before the EJB can be removed, such as releasing any resources held by the EJB.

- ✓ *EJBs can be passivated-state information is saved and the EJB instance is freed up for use by the resource pool-as required by the container. A passivated EJB will be activated-state information restored-by the container if a client request to that particular EJB object is received.*

Database connection Pooling

- ✓ *Opening a database connection is slow. Also, database connections could be a scarce resource, due to, for example, licensing restrictions. The EJB container manages this expense through database connection pooling-the container keeps a pool of open connections that can be assigned and unassigned to an EJB as required, resulting in fast and efficient connections.*
- ✓ *For entity **EJBs using CMP**, database connections are handled automatically. No connection or SQL code needs to be written-you simply specify the JNDI name of the JDBC data source in the EJB deployment descriptor and use container-specific deployment tools to generate the connect routines for you. The container manages the database connection pool.*
- ✓ *For entity **EJBs using BMP** or for session EJBs, you need to write connection code to connect to a JDBC data source and write SQL code to access the database. The JDBC data source is still managed by the container-the JDBC data source actually uses a database connection pool maintained by the container*

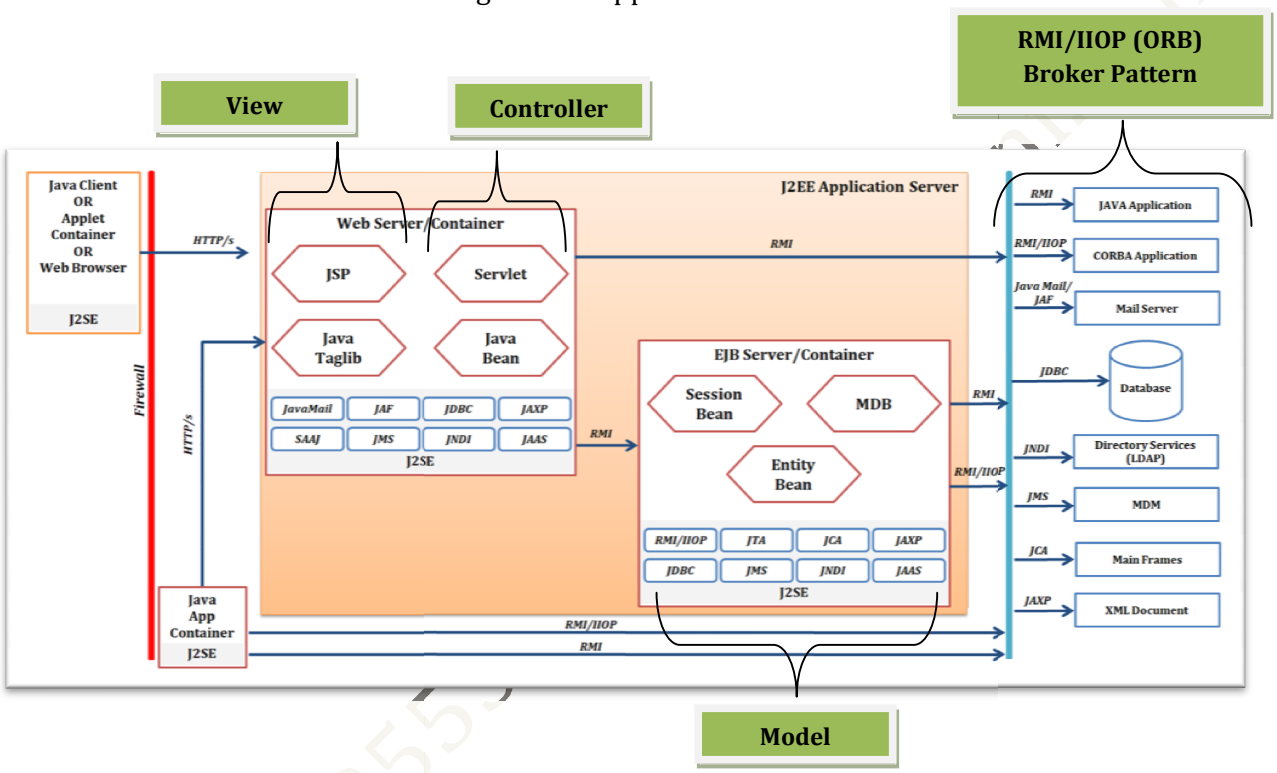
Messaging

- ✓ *EJB containers are required to provide messaging support for the **asynchronous exchange** of messages. **JMS**, or other messaging types, can be used by message-driven EJBs process delivered messages. Because of the JMS's involvement with EJBs, they must support transactional access from Web and EJB container components like servlets, JSP pages, and EJBs.*

2.4 J2EE Architecture Patterns

The Java™ 2 Platform, Enterprise Edition (J2EE™) addresses the need of developing complex n-tiered Enterprise Systems by providing a well-documented, standards-based framework for developing and running distributed, multi-tier, component-based Java applications.

This framework handles much of the low-level complexity of the application, such as ***remote connectivity***, ***naming***, ***persistence***, ***security***, and ***transaction management***, leaving developers free to concentrate on the business logic of the application.



(1) J2EE architecture adopts Model-View-Controller (MVC) Pattern.

- Model

  The model is the principal part of application program. The model is another name for the application logic layer. It expresses business data, it is clearly said that one model is a record in database. It is independent of the data form, in other words one model may provide the data for many views, in this way it reduces the repeated code for our application program.

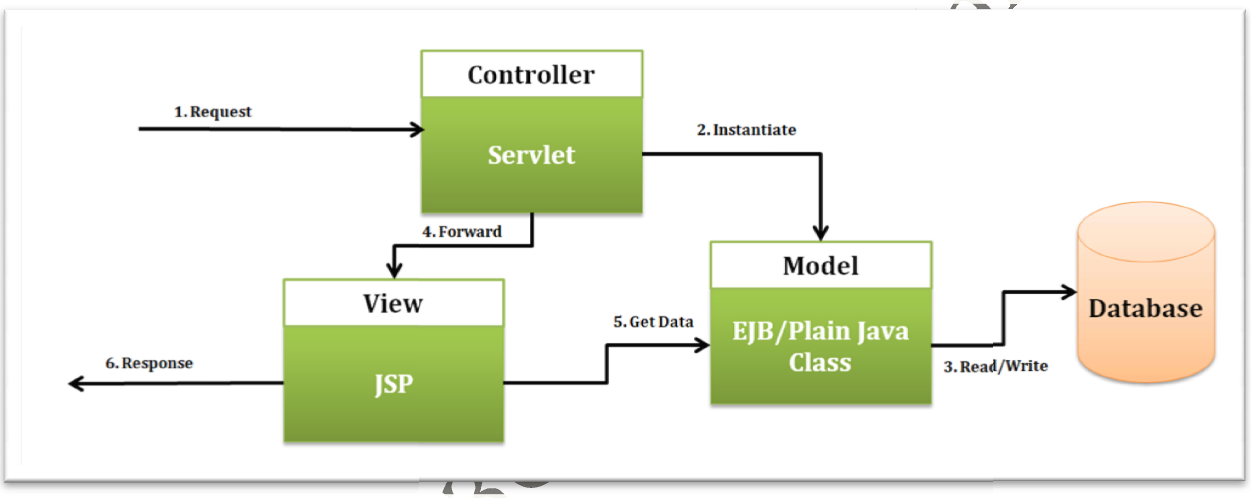
- View

  The view is an interrelated part of User Interface (UI) in application program and a seeing and exchanging interface by users. One of advantages is that it can process many different views for your application program using MVC. Actually does not have

the true processing to take place in the view, and the view is only regarded as a means of output data and user operation.

- Controller

  The controller controls UI data displaying and updates the model object state according to the users input. The controller accepts the user input and calls the model and view to complete the user demand. So when we click the hyperlinks in the web page and send HTML table list, the controller itself does not have any output and make any processing. It only receives the request and determines calling which model to deal with the request, and it confirms to use which view to display the returning data of the model processing.



(2) J2EE architecture adopts Broker Pattern.

- RMI/IIOP calls

  Remote Method Invocation technology run over Internet Inter-Orb Protocol (RMI-IIOP) allows Java components to communicate with legacy CORBA components written in other languages like C++ or Smalltalk.

  This is typical implementation of Broker Pattern.

3. QUALITY ATTRIBUTES

3.1 Enterprise Application Quality Requirements

J2EE enables development of enterprise distributed systems while addressing the following Quality Attributes.

Quality Attribute	Requirement
Scalability	System should support variations in load without human intervention
Availability/ Reliability	System should provide 24/7 availability with very small downtime periods
Security	System should authenticate users and protect against unauthorized access to data
Usability	Different users should be able to access different content in different forms
Performance	Users should be provided with responsive systems
Portability	J2EE should be able to be implemented with minimal work on a variety of computing platforms
Buildability	Application developers should be provided with facilities to manage common services such as transactions, name services, and security
Balanced Specificity	Detailed enough to provide meaningful standard for component developers, vendors, and integrators, but general enough to allow vendor-specific features and optimizations
Implementation Transparency	Provide complete transparency of implementation details so that client programs can be independent of object implementation details (server-side component location, operating system, vendor, etc.)
Interoperability	Support interoperation of server-side components implemented on different vendor implementations; allow bridges for interoperability of the J2EE platform to other technologies such as CORBA and Microsoft component technology
Evolvability	Allow developers to incrementally adopt different technologies
Extensibility	Allow incorporation of relevant new technologies as they are developed

3.2 Quality Attributes offered by J2EE

The following matrix shows the typical quality attributes addressed by J2EE/J2SE APIs, Standards & Technologies.

##	J2SE/J2EE APIs, Technologies & Concepts	QUALITY ATTRIBUTES											
		SCALABILITY	AVAILABILITY & RELIABILITY	SECURITY	USABILITY	PERFORMANCE	PORTABILITY	BUILDABILITY	BALANCED SPECIFICITY	IMPLEMENTATION TRANPERANCY	INTEROPERABILITY	EVOLVABILITY	EXTENSIBILITY
1	EJB 2.1	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
2	JAAS			✓							✓		
3	JAF 1.0	✓	✓			✓	✓	✓		✓		✓	
4	JAXP 1.2		✓		✓						✓		
5	JAX-RPC 1.1	✓	✓					✓	✓	✓	✓		✓
6	Web Services for J2EE 1.1	✓	✓					✓	✓	✓	✓		✓
7	SAAJ 1.2		✓		✓	✓	✓	✓			✓	✓	✓
8	JAXR 1.0	✓	✓		✓	✓			✓	✓	✓		✓
9	JavaMail 1.3	✓			✓	✓					✓	✓	
10	JDBC 3.0		✓			✓	✓		✓	✓	✓		✓
11	JMS 1.1	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
12	JNDI		✓	✓		✓			✓	✓		✓	✓
13	JTA 1.0		✓	✓	✓	✓							
14	J2EE Connector 1.5			✓	✓	✓	✓	✓				✓	✓
15	JSP 2.0	✓			✓	✓		✓					✓
16	Servlet 2.4	✓	✓		✓	✓					✓	✓	✓
17	RMI-IIOP	✓	✓				✓	✓			✓	✓	✓
18	J2EE Management 1.0							✓	✓	✓			
19	JMX 1.2		✓					✓	✓	✓		✓	✓
20	JACC 1.0	✓		✓	✓	✓							✓
21	J2EE Deployment 1.1							✓	✓	✓		✓	✓
22	Deployment Descriptors			✓		✓			✓	✓		✓	✓
23	Standards							✓	✓	✓		✓	✓

← *The End* →