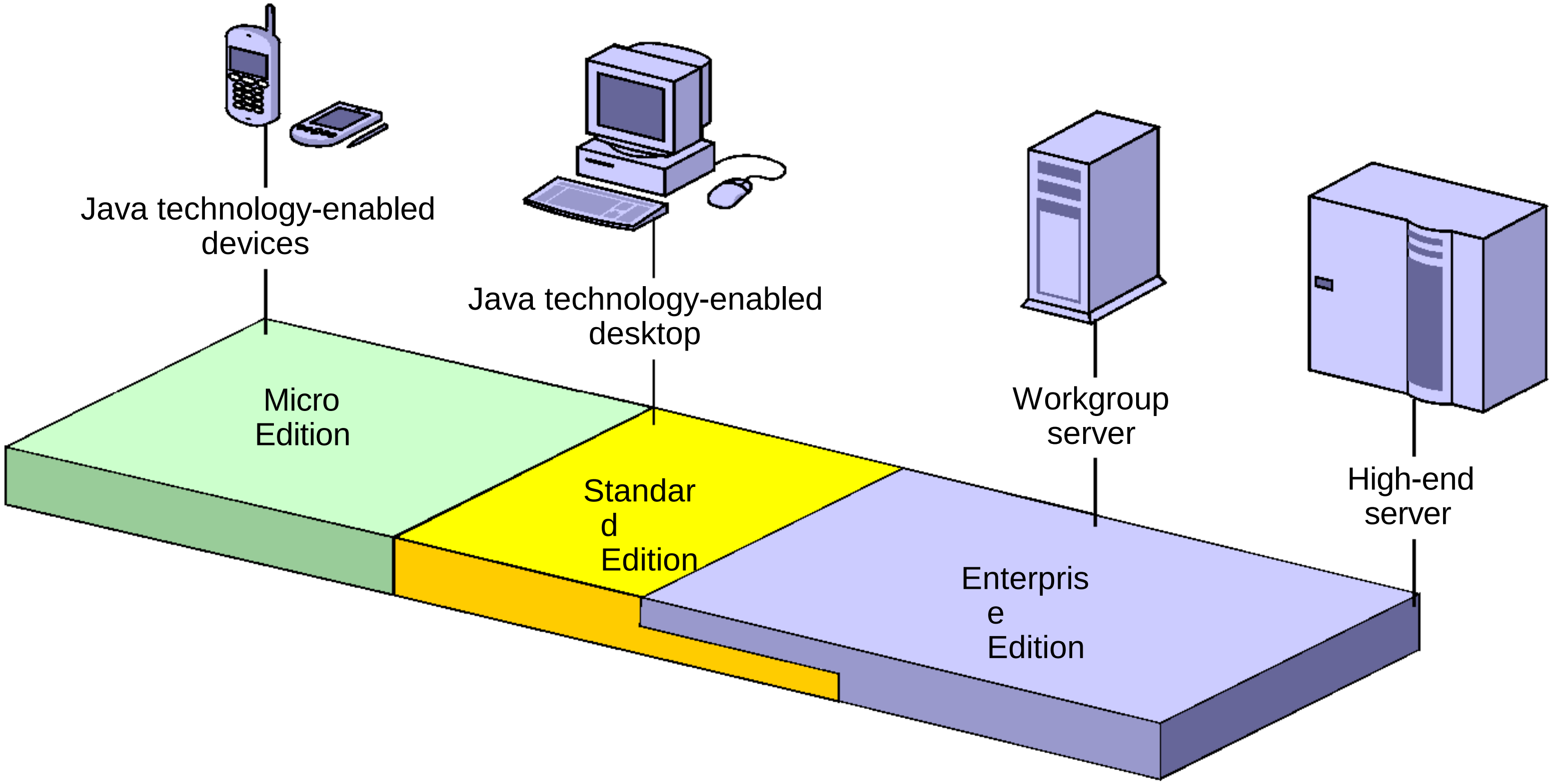
Placing the JavaTM EE Model in Context

Requirements of Enterprise Applications

The Java EE platform:

- Is an architecture for implementing enterprise-class applications
- Uses Java and Internet technology
- Has a primary goal of simplifying the development of enterprise-class applications through an application model that is:
  - Vendor-neutral
  - Component-based

JavaTM Technology Platforms



Enterprise Application Infrastructure Technologies

Single-User Business Application

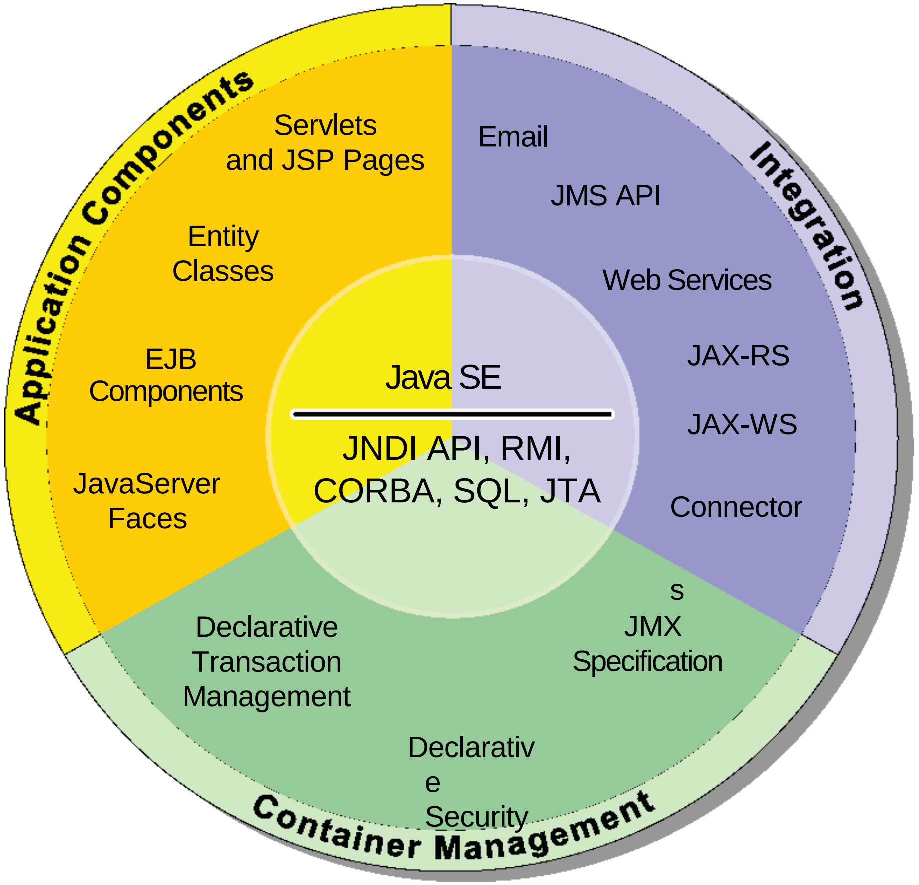
+

Enterprise Infrastructure Requirement	Technology or Domain
User authentication	Security domain technologies
Multi-user support	Multithreading technologies
Data preservation	Persistence technologies
Data integrity	Transaction technologies
Client-tier communications	Networking and distributed object technologies
Communication with other enterprise and legacy applications	Messaging, connector, and related technologies
Service location assistance	Naming service technologies

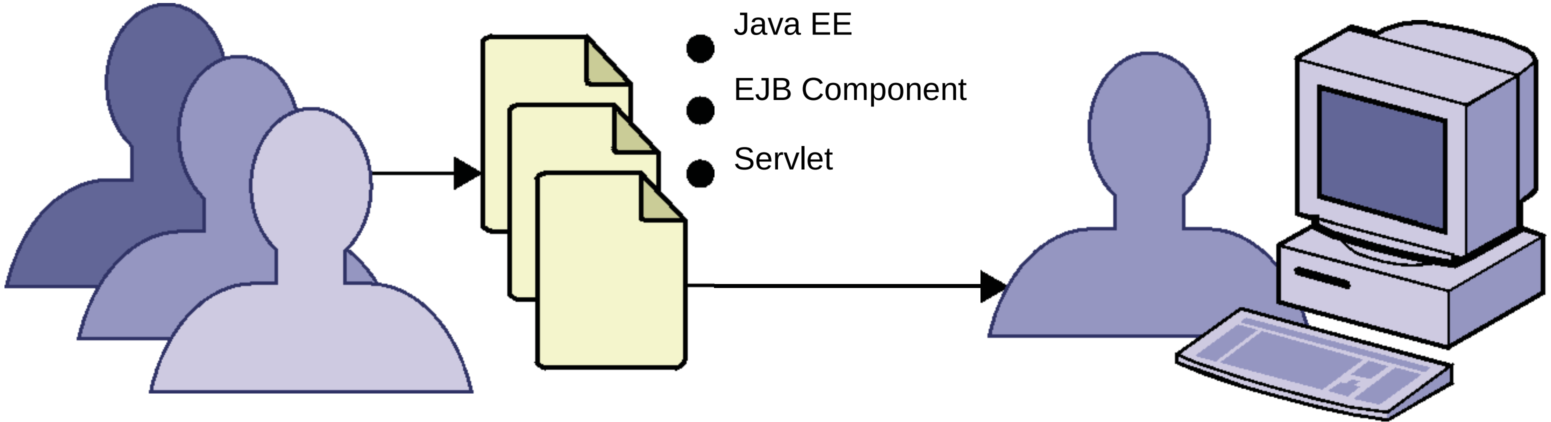
=

Multi-User Enterprise Application

Java EE Technology Suite



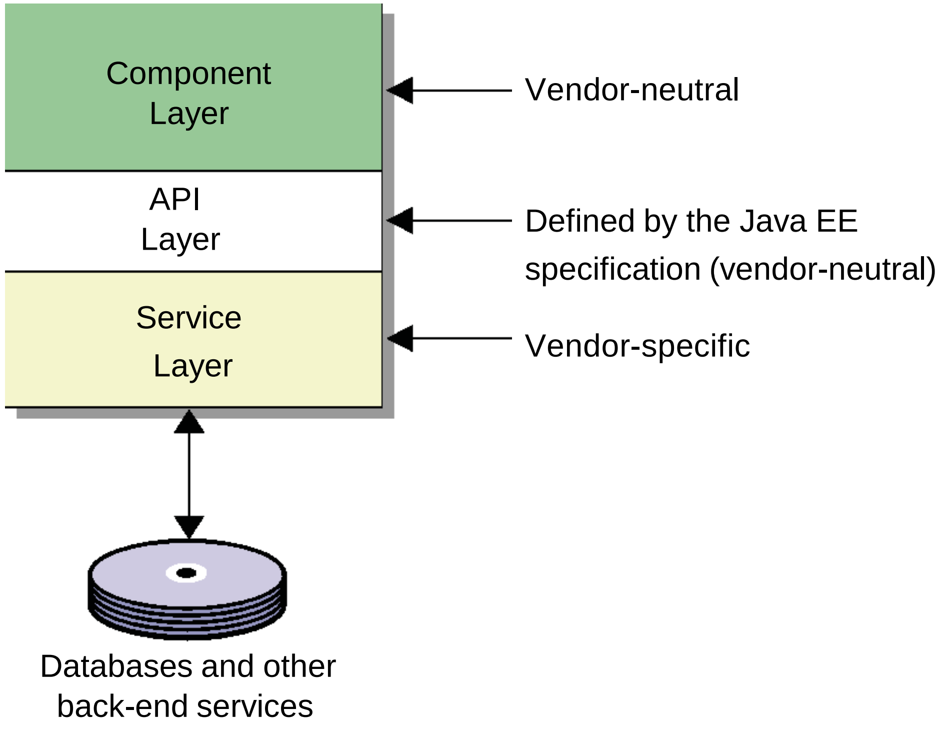
Java EE Specifications and the Java Community Process^{SM}



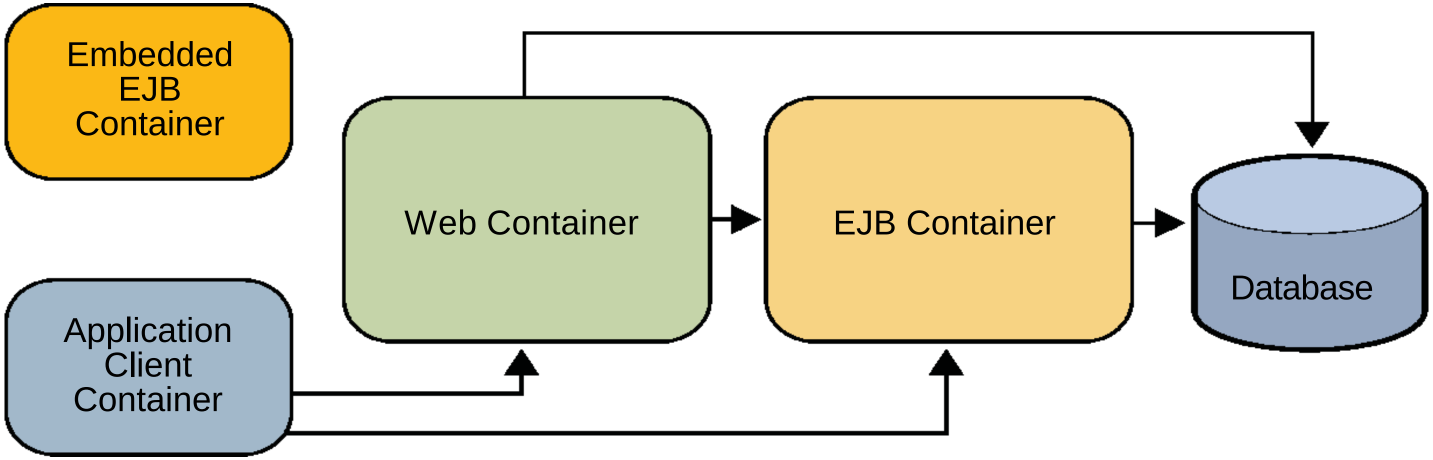
- Java EE
- EJB Component
- Servlet

Java Community Process Specifications and APIs Developers

Component, API, and Service Layer

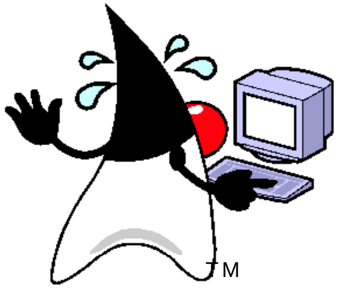

Component Layer
API Layer
Service Layer
Vendor-neutral
Defined by the Java EE specification (vendor-neutral)
Vendor-specific
Databases and other back-end services

Java EE Component Containers



Advantages of Using Server-Provided Services

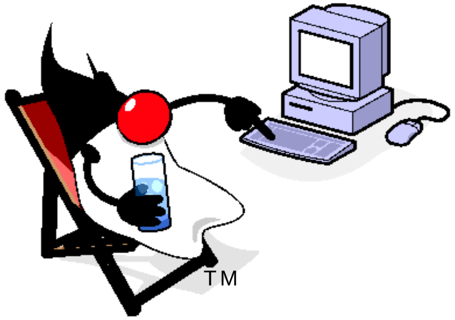
Build From the Ground Up



Developer's Checklist

□ Business services
□ Persistence
□ Transaction management
□ Multi-threading
□ Security management
□ Networking
□ Service publishing

Use Application Component Server



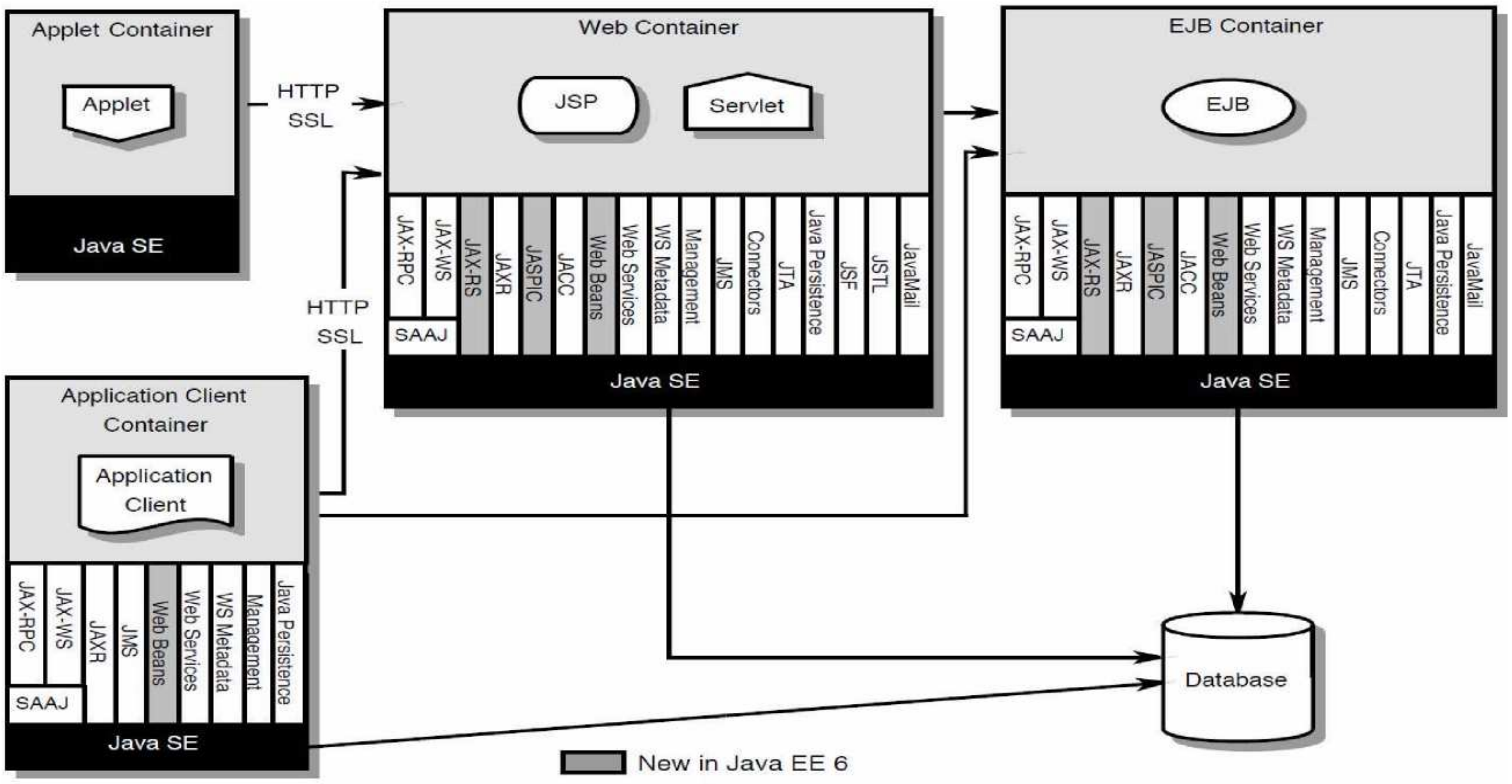
Developer's Checklist

□ Business services

Services Provided by Server

☑ Persistence
☑ Transaction management
☑ Multi-threading
☑ Security management
☑ Networking
☑ Service publishing

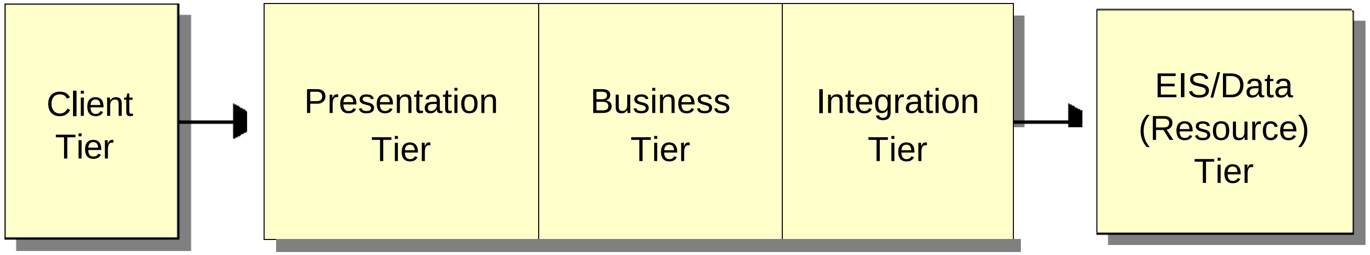
Java EE Service Infrastructure



Java EE Platform Tiers and Architecture

- **The Java EE specification outlines an architectural model based on tiers that developers are encouraged to use**
- **The historical motivation for tiering:**
  - **Division of labor around specialized servers**
  - **Formal definitions of application responsibilities based on the division of labor**

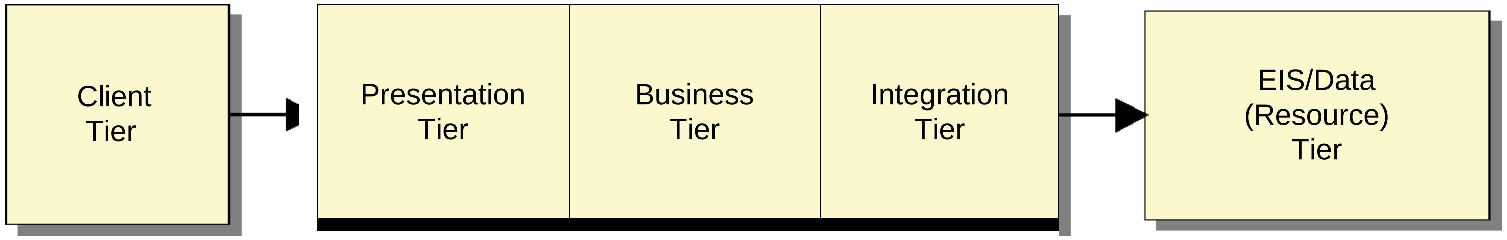
N-Tier Architectural Model



The N-tier architectural model:

- **Programmatically separates application functionality across three or more tiers**
- **Has tier components and tier infrastructure that is uniquely suited to a particular task**
- **Has programmatic interfaces that define the tier boundaries**

Java EE Tiered Architecture



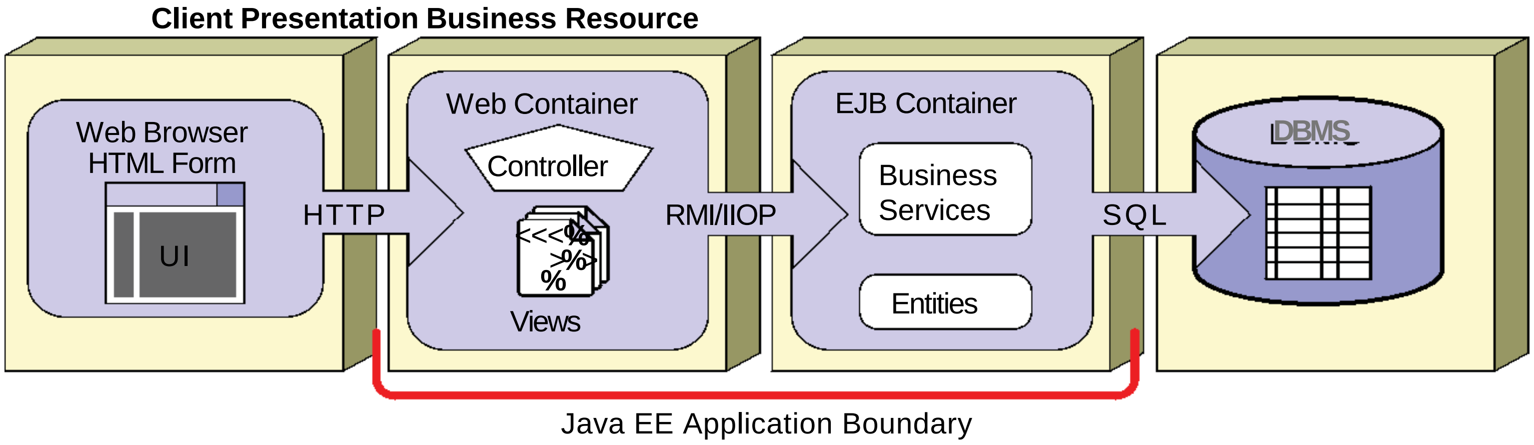
Client Tier

Presentation Tier

Business Tier

Integration Tier

EIS/Data (Resource) Tier



Client Presentation Business Resource

Web Browser HTML Form

UI

HTTP

Web Container

Controller

Views

RMI/IIOP

EJB Container

Business Services

Entities

SQL

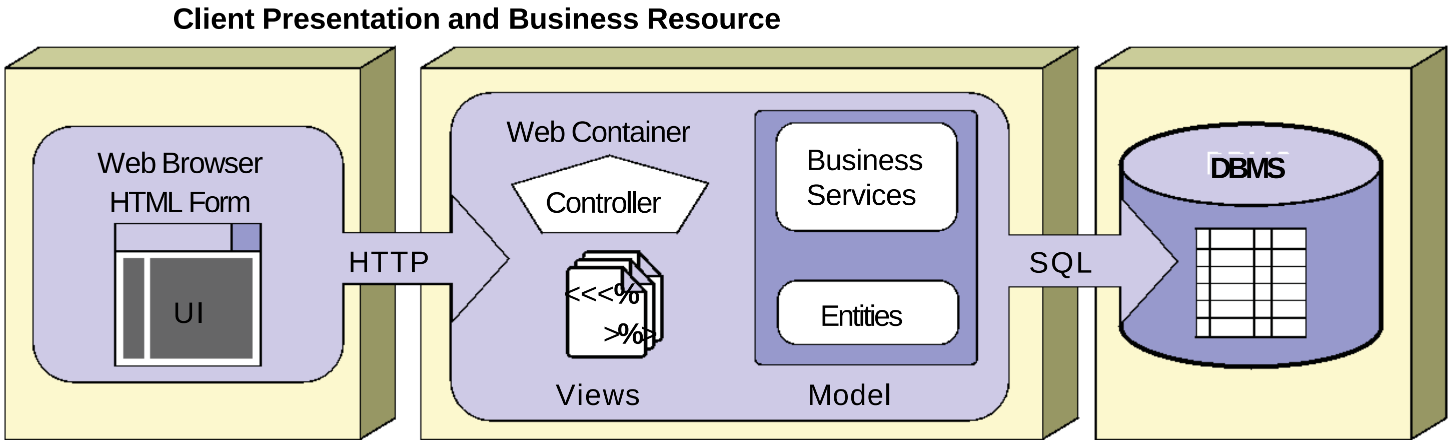
DBMS

Java EE Application Boundary

Java EE Application Architecture

- **Web-centric architecture**
- **Combined web and EJB component-based architecture, sometimes called EJB component-centric architecture**
- **Business-to-business (B2B) application architecture**
- **Web service application architecture**

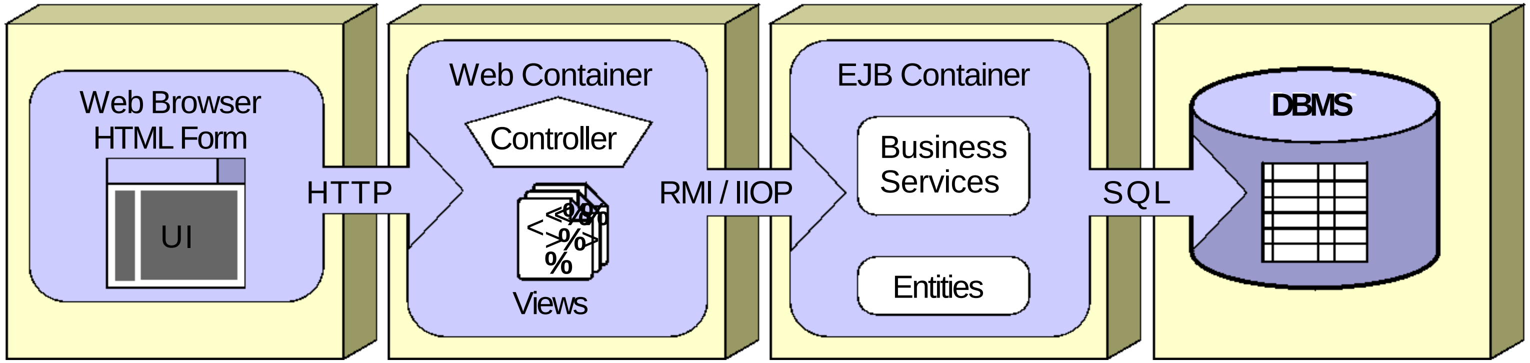
Java EE Web-Centric Architecture



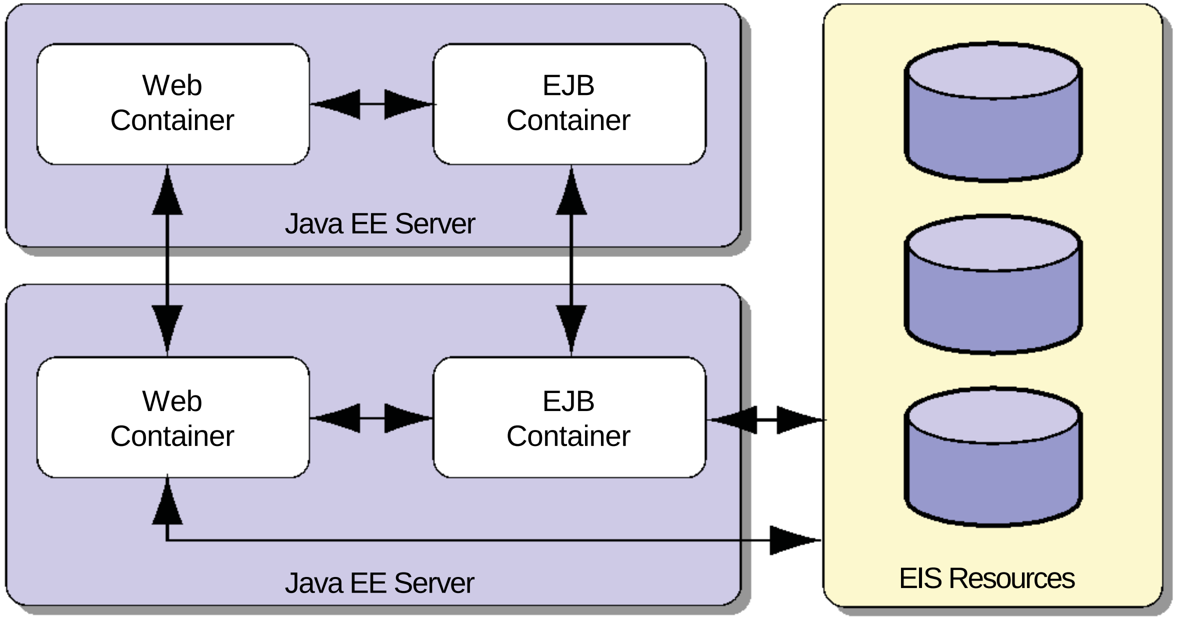
The introduction of EJB Lite in Java EE 6 allows the use of some EJB technology in web-centric architectures.

Java EE EJB Component-Centric Architecture

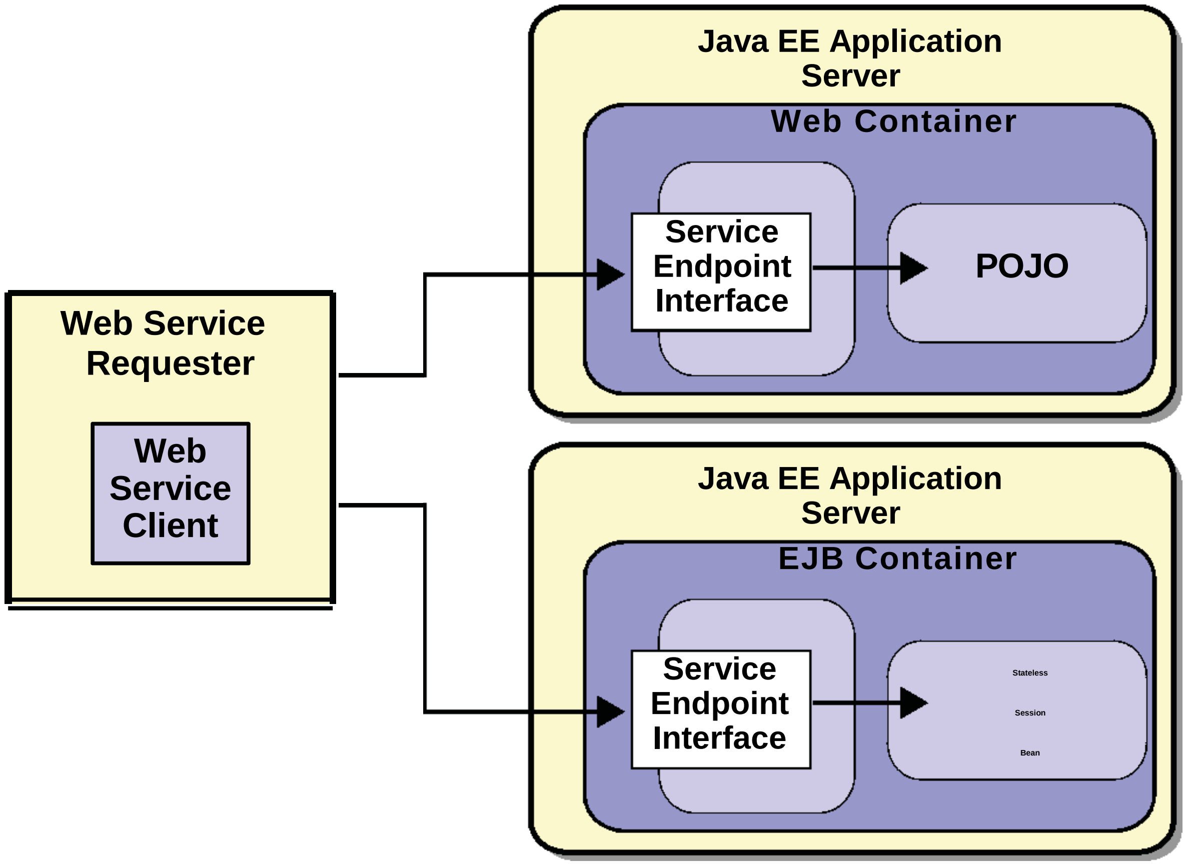
Client Presentation Business Resource



B2B Application Architecture



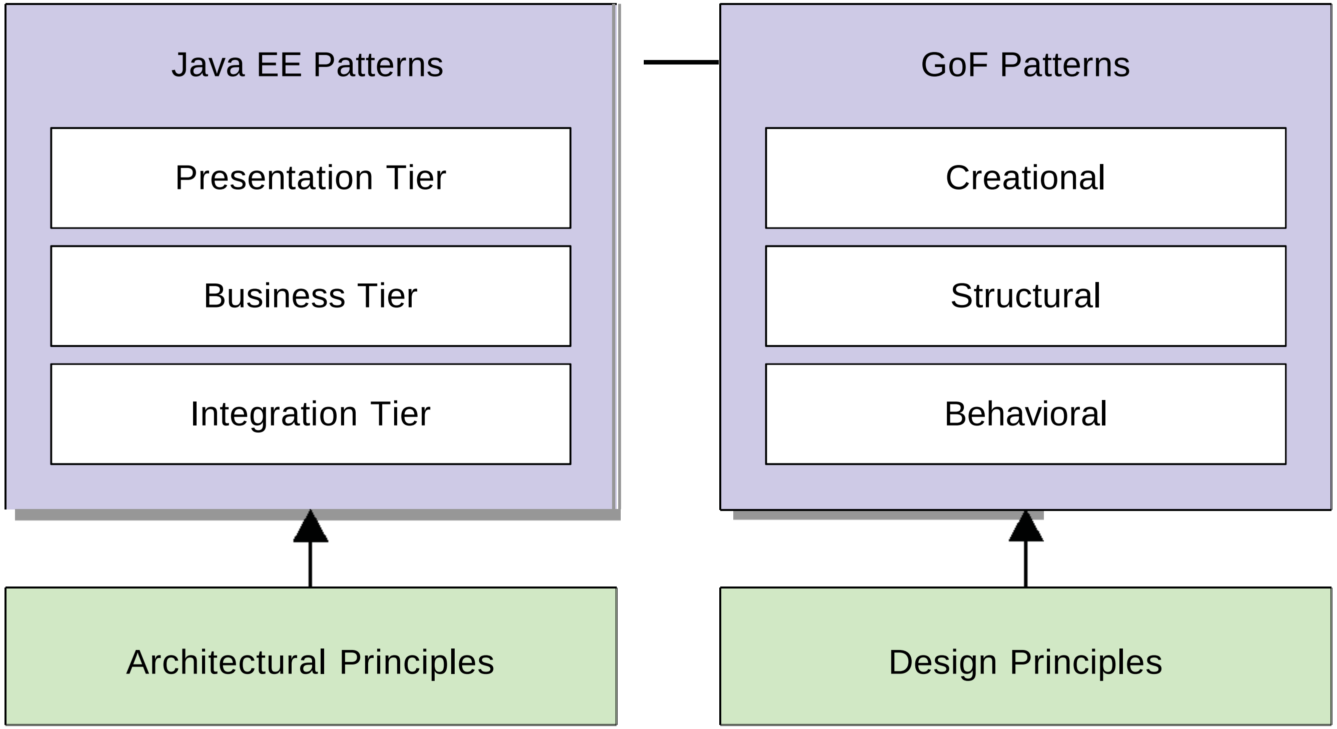
Java EE Web Service Architecture


Web Service Requester
Web Service Client
Java EE Application Server
Web Container
Service Endpoint Interface
POJO
Java EE Application Server
EJB Container
Service Endpoint Interface
Stateless
Session
Bean

Java EE Patterns

- **Patterns provide a standard solution for well understood programming problems.**
- **The Java EE pattern catalog:**
  - **Helps a developer create scalable, robust, high-performance, Java EE technology applications**
  - **Presupposes the use of the Java programming language and the Java EE technology platform**
  - **Are, in many places, closely related to the Gang of Four (GoF) patterns**

Java EE Pattern Tiers



Java EE Patterns

Presentation Tier

Business Tier

Integration Tier

GoF Patterns

Creational

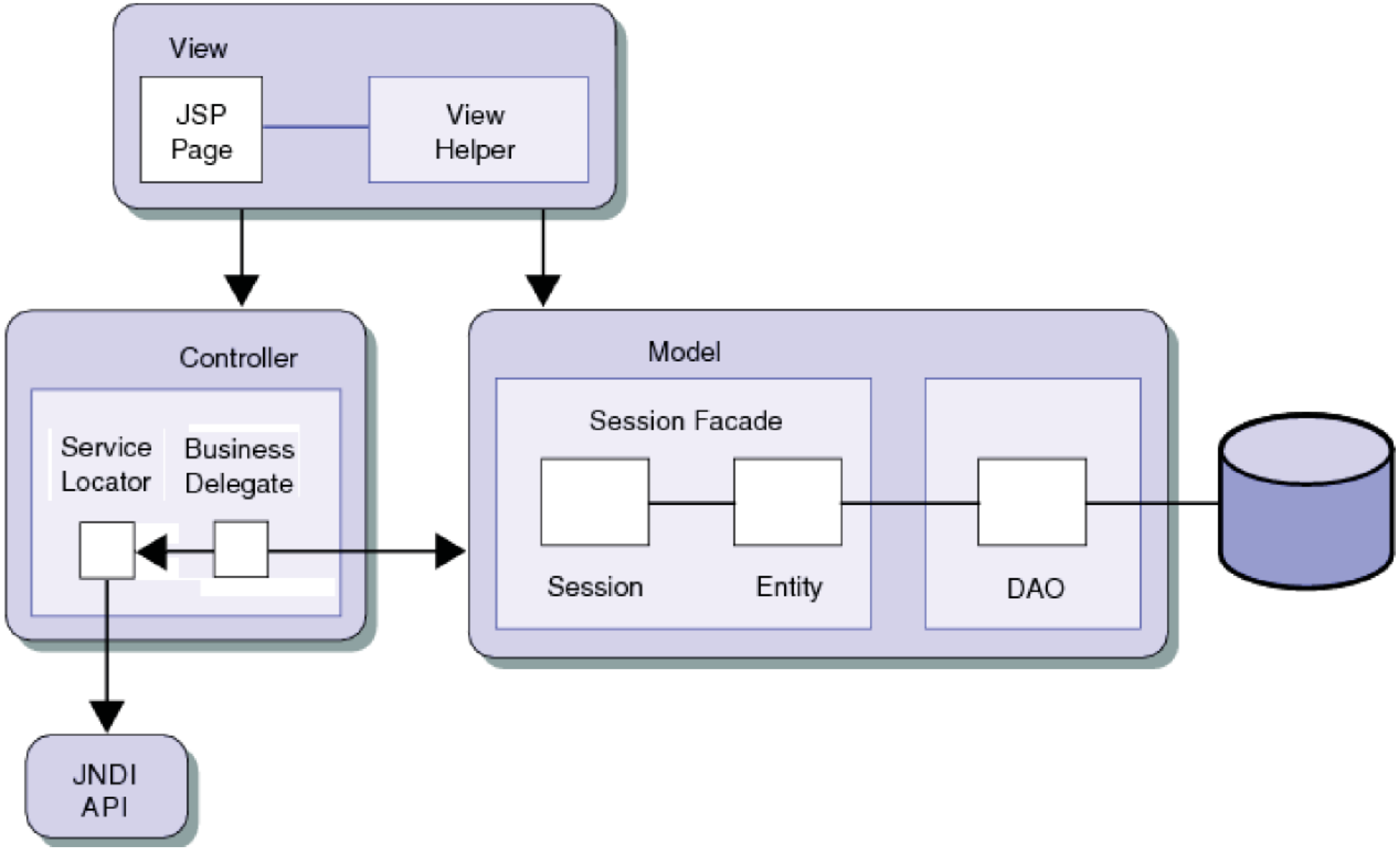
Structural

Behavioral

Architectural Principles

Design Principles

Using Java EE Patterns



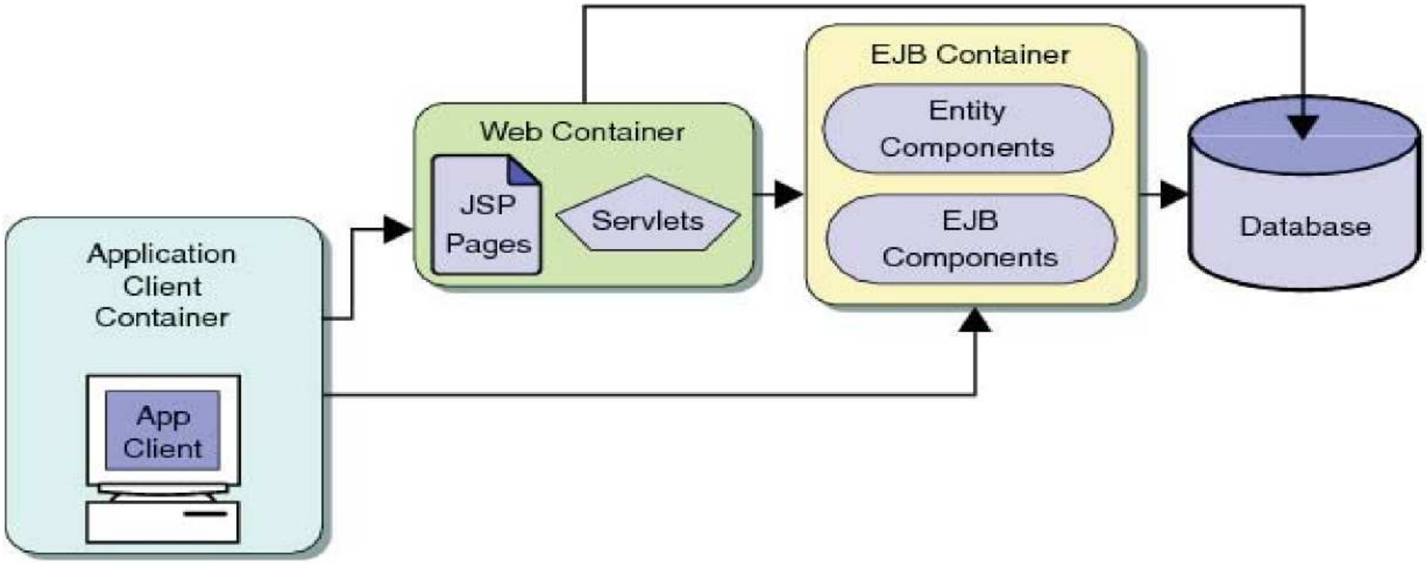
Java EE BluePrints

- **Developed by the Java software group**
- **Provide a set of guidelines and a sample application**
- **Used as a reference when designing and developing a Java EE application or Java EE application components**
- **Known as the *Java BluePrints Solutions Catalog for Java EE***

Principles of Component-Based Development

- **The EJB specification was designed from the outset to support integration of components from different vendors.**
- **EJB components can be authored without knowing the environment in which they will be used.**
- **Applications based on EJB components are loosely coupled:**
  - **Loosely coupled systems are easier to test and maintain.**
  - **Components of a loosely coupled system are easier to reuse.**

Java EE Components



Java EE Component Characteristics

- **State and properties**
- **Encapsulation by a container**
- **Support for local and distributable component interactions**
- **Location transparency**
- **Component references obtained using a naming system**

Component State and Properties

- **State is associated data that has to be maintained across a series of method calls**
  - **A component might or might not be stateful**
  - **Stateless components might have performance advantages over stateful components**
- **A property is a component feature that can be either read *and* written or read *or* written by its clients**
  - **A property might be represented internally by an instance variable**
  - **Properties are modeled as accessor and mutator method pairs**

Encapsulated Components

- **Encapsulation is an important concept in object-oriented programming**
- **Java EE encapsulates components in containers that:**
  - **Provide life-cycle management**
  - **Isolate components from other components**
  - **Isolate components from the runtime environment**

Component Proxies

Some Java EE components, such as EJB, are utilized through proxies.

- **There is no direct reference to the component.**
- **The** `new()` **operator should not be called on the component.**
- **The Java EE container provides the proxy.**
- **A proxy allows the container to intercept method calls and provide container based functionality such as security checks and transaction management.**
- **Some components require the developer to write an interface for the proxy. Java EE 6 eliminates the need for the interface in some cases.**

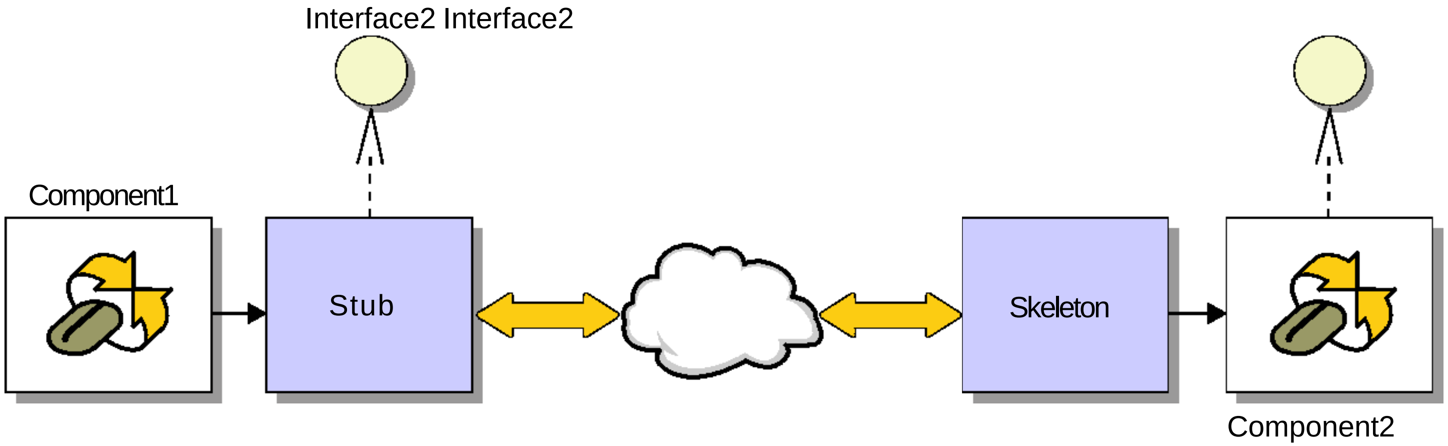
Distributable and Local Component Interactions

The developer specifies whether an interaction is to be local or distributable.

- **Local – The application server makes components available to each other in the same JVM machine.**
- **Distributable – The application server provides an RMI infrastructure by which components communicate.**

Both strategies have associated costs and benefits.

Distributed Components and RMI



Distributed Components and RMI

The RMI infrastructure must be able to manage the following design issues:

- **Marshalling and unmarshalling of arguments and return values**
- **Passing distributed exceptions**
- **Passing security context and transaction context**

Advantages and Disadvantages of a Distributed Component Model

The following advantages derive from location transparency:

- **Increased fault tolerance**
- **Improved load sharing between hosts**

The following disadvantages derive from RMI overhead:

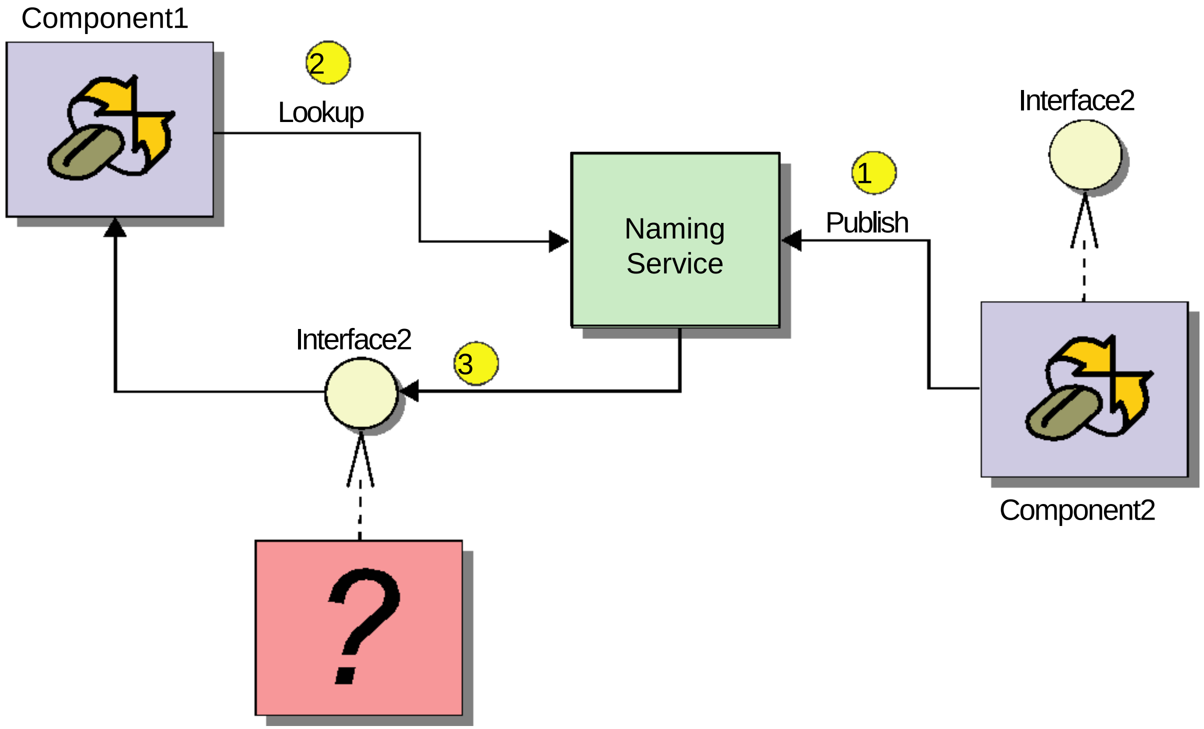
- **Data marshalling overhead**
- **Network latency**
- **More complex exception handling**

Location Transparency

Location transparency is a design goal of the distributed component model in the Java EE platform:

- **The calling component is not concerned with the physical location of the target component.**
- **A component can be deployed in more than one host, which has these benefits:**
  - **Load balancing**
  - **Fault tolerance**
- **The application server vendor is responsible for realizing these benefits.**
- **The developer is responsible for developing specification-compliant components.**

Naming Services in the Component Model



Use of the Java Naming and Directory InterfaceTM (JNDI) API in the Java EE Component Model

In the Java EE platform environment, the JNDI API:

- **Implements a general lookup service for:**
  - **Java EE components**
  - **External resources**
  - **Component environment**
- **Abstracts the underlying naming protocols and implementation:**
  - **CORBA naming service**
  - **LDAP**
  - **Vendor-specific protocols**

The `Context` Interface and the `InitialContext` Object

The `Context` interface is the basis for all naming operations.

- **The `InitialContext` object is a specific implementation of the `Context` interface.**
- **An `InitialContext` object represents the entry point to the naming service.**
- **The namespace can be hierarchical.**
- **A** lookup **operation on a `Context` object results either in an object or in a subcontext.**

The `Context` Interface and the `InitialContext` Object

A subcontext also implements the `Context` interface. The following two code snippets have the same effect:

```
Context c = new InitialContext();
Object o = c.lookup("aaa/bbb");
```

Or

```
Context c = new InitialContext();
Context subcontext = (Context) c.lookup("aaa"); Object
o = subcontext.lookup("bbb");
```

Configuring the `InitialContext` Object

Configuration of the `InitialContext` object differs within a Java EE component and in a standalone application:

- **Within a Java EE component, the container provides configuration to the `InitialContext` object:**

  ```
  Context c = new InitialContext();
  ```

- **In a standalone application, the `InitialContext` object may require configuration:**

  ```
  Hashtable env = new Hashtable();
  env.put ("java.naming.factory.Initial",
     "com.sun.jndi.cosnaming.CNCtxFactory");
  env.put("java.naming.provider.url", "iiop://hostname:3700"); Context
  c = new InitialContext(env);
  ```

Using JNDI API as a Resource Locator

In addition to components, JNDI API calls can locate:

- **Connections to relational databases**
- **Connections to messaging services**
- **Message destinations**
- **Component environment variables**
- **Connections to legacy systems that are supported by resource adapters**

Narrowing and Remote Objects

JNDI lookup results differ for non-remote and remote objects:

- **For non-remote objects, the result of a lookup is cast to the appropriate type:**

  ```
  Context c = new InitialContext();
  DataSource ds = (DataSource)c.lookup("jdbc/bank");
  ```

- **For remote objects, the result of a lookup requires *narrowing* to the appropriate type:**

  ```
  Context c = new InitialContext();
  Object o = c.lookup("ejb/BankMgr");
  BankMgr bankMgr = (BankMgr)
       javax.rmi.PortableRemoteObject.narrow (o,
             BankMgr.class);
  ```

- **The Java EE specifications require narrowing for remote objects however some application servers allow casting.**

Using a Component Context to Locate Components

Java EE components have their environment represented by a context object, such as `EJBContext`. A component's context:

- **Is automatically supplied to a component, no lookups are needed**
- **Can be used in place of JNDI for lookups**
- **Simplifies lookup code:**

```
@Resource private javax.ejb.SessionContext context;
public void myMethod() {
  BankMgr bankMgr = (BankMgr)context.lookup("ejb/BankMgr");
}
```

- **Does not require the use of `PortableRemoteObject.narrow` for remote components**

Using Dependency Injection to Locate Components

Dependency injection can be used to locate resources. Containers assign values to annotated variables. Dependence Injection:

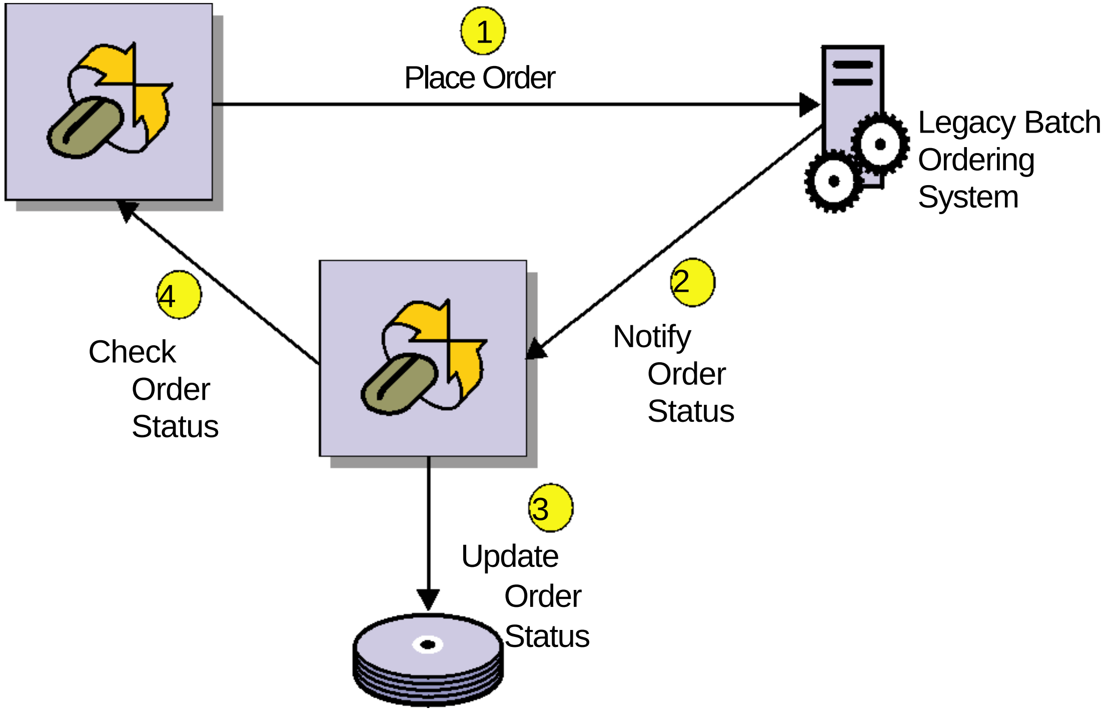
- **Replaces JNDI lookup code**
- **Uses Java annotations:**

  ```
  @EJB private BankMgr bankMgr;
  ```
- **Uses a container to locate resources**
- **Only works in managed components**
- **Can be used in Java EE 5 or 6 components to locate J2EE 1.4 components**
- **Has been updated in Java EE 6 (JSR-299 & JSR-330)**

Asynchronous as Compared to Synchronous Communication

	Synchronous	Asynchronous
Semantics	Request-response	Request-notification
Blocking	Client blocks until operation completes	Client does not block
Response	Client gets a direct response	Client may get a deferred response

Asynchronous Component-to-Component Interaction



Asynchronous Messaging

The application server must provide a messaging service to support asynchronous component interaction.

- **The J2EE 1.4 specification required that a server must provide infrastructure for web services and XML messaging.**
- **Components use the JMS API to send messages to other components or to external resources.**
- **Message-driven beans act as consumers of messages.**
- **Java EE 6 adds support for asynchronous processing that does not require messaging**

Advantages and Disadvantages of Asynchronous Interactions

Asynchronous component interaction results in both benefits and costs, compared to synchronous component interaction.

- **Advantages:**
  - **Reduced coupling between components, which results in reduced long-term costs of management**
  - **Accommodation of operations that take an extended time to complete**
- **Disadvantages:**
  - **Requires a more complex infrastructure**
  - **Is usually less efficient in network resource usage**

Developing Java EE Applications

- **Performed by a group of people**
- **Involves separate roles and responsibilities**

Java EE Roles

Roles related to application development:

- **Application component provider**
- **Application assembler**
- **Deployer**

Other defined roles:

- **System administrator**
- **Tool provider**
- **Product provider**

Java EE Roles

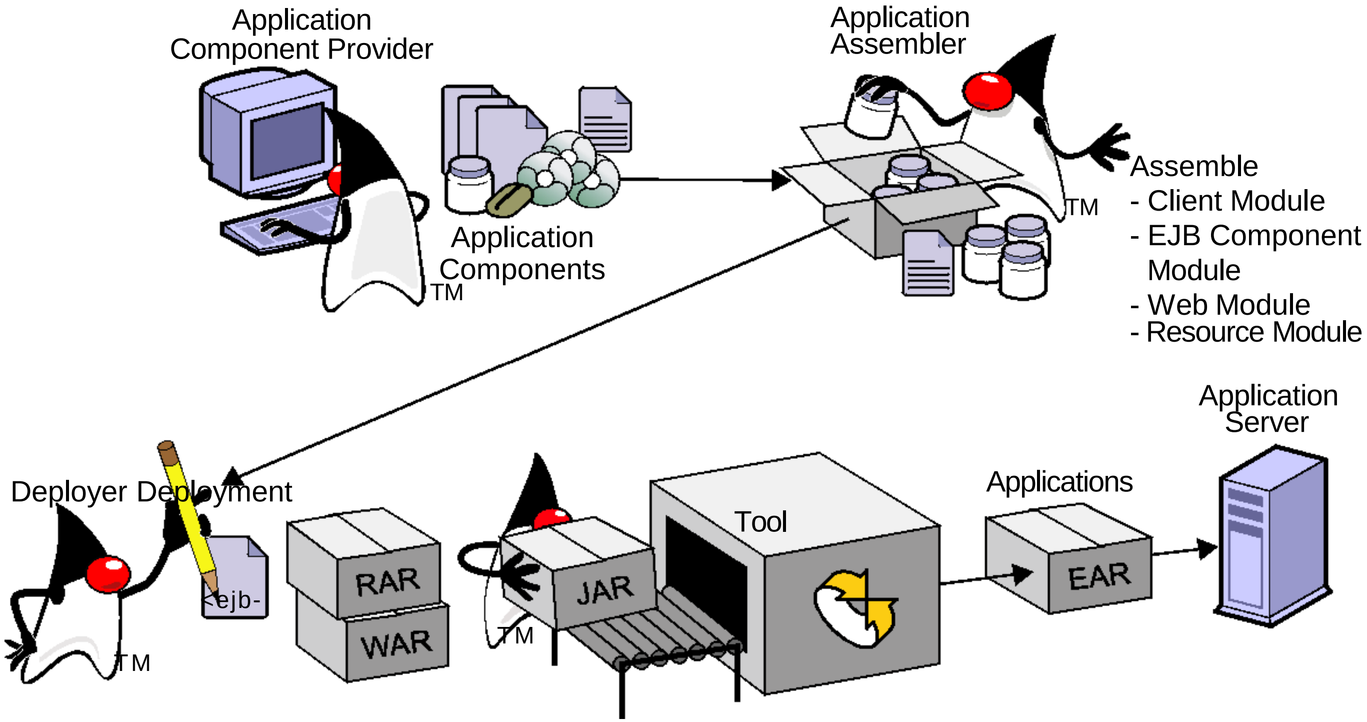
Important role distinctions:

- **Distinction between tool provider and product provider**
- **Distinction between component provider, application assembler, and deployer**

Steps for Developing a Java EE Application

- **Designing**
- **Coding**
- **Creating deployment descriptors**
- **Packaging**
- **Assembly**
- **Deployment**

Java EE Application Development Process



Development Tools

Java EE applications are traditionally development within an integrated development environment (IDE). IDEs provide:

- **An editor**
- **The ability to manage Java EE components in a graphical manner**
- **The ability to compile from within the IDE**
- **The ability to debug source code**
- **The ability to edit deployment descriptors using a graphical tool**
- **The ability to deploy to one or more application servers**

Configuring and Packaging Java EE Applications

- **Developers package individual components into archive files. These archive files contain:**
  - **Relevant class files**
  - **XML deployment descriptors (optional)**
- **These archive files are packaged into a super archive to form a complete application.**
- **The contents and structure of these archive files are mandated by the Java EE specification.**
- **Any compliant application server should be able to accept any compliant application.**

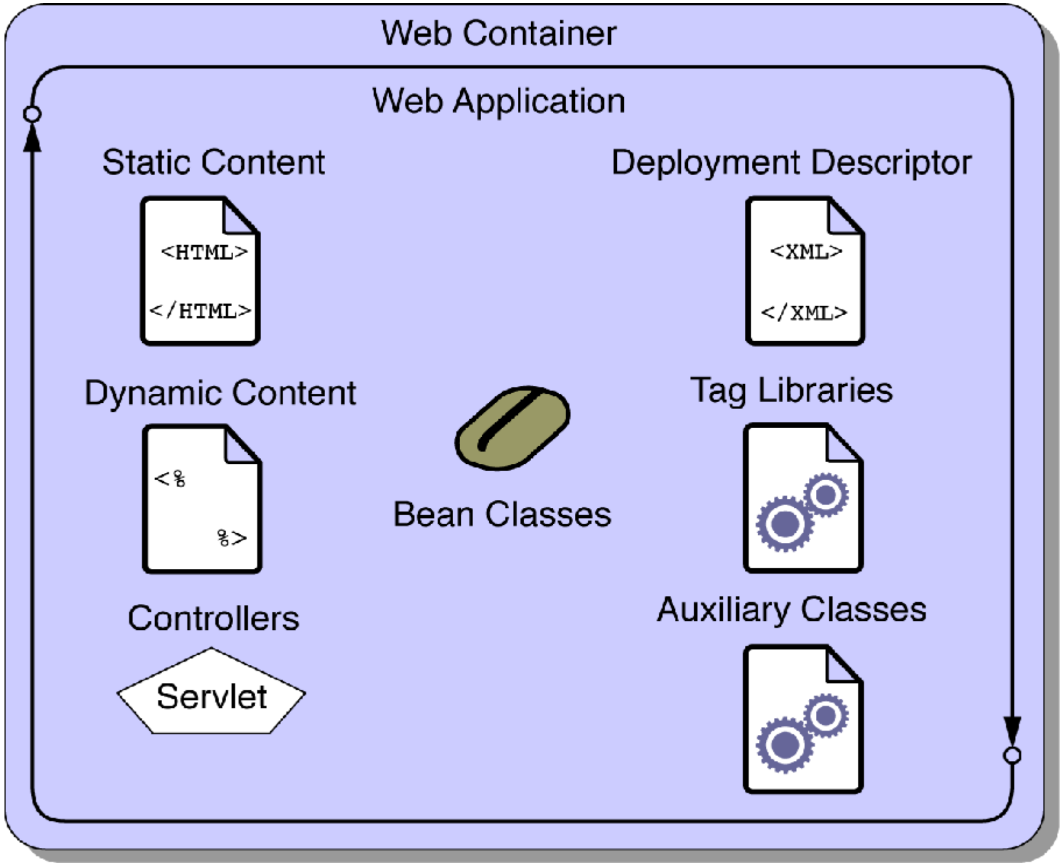
Configuring and Packaging Java EE Applications

There are four basic types of archive files used in a Java EE development project:

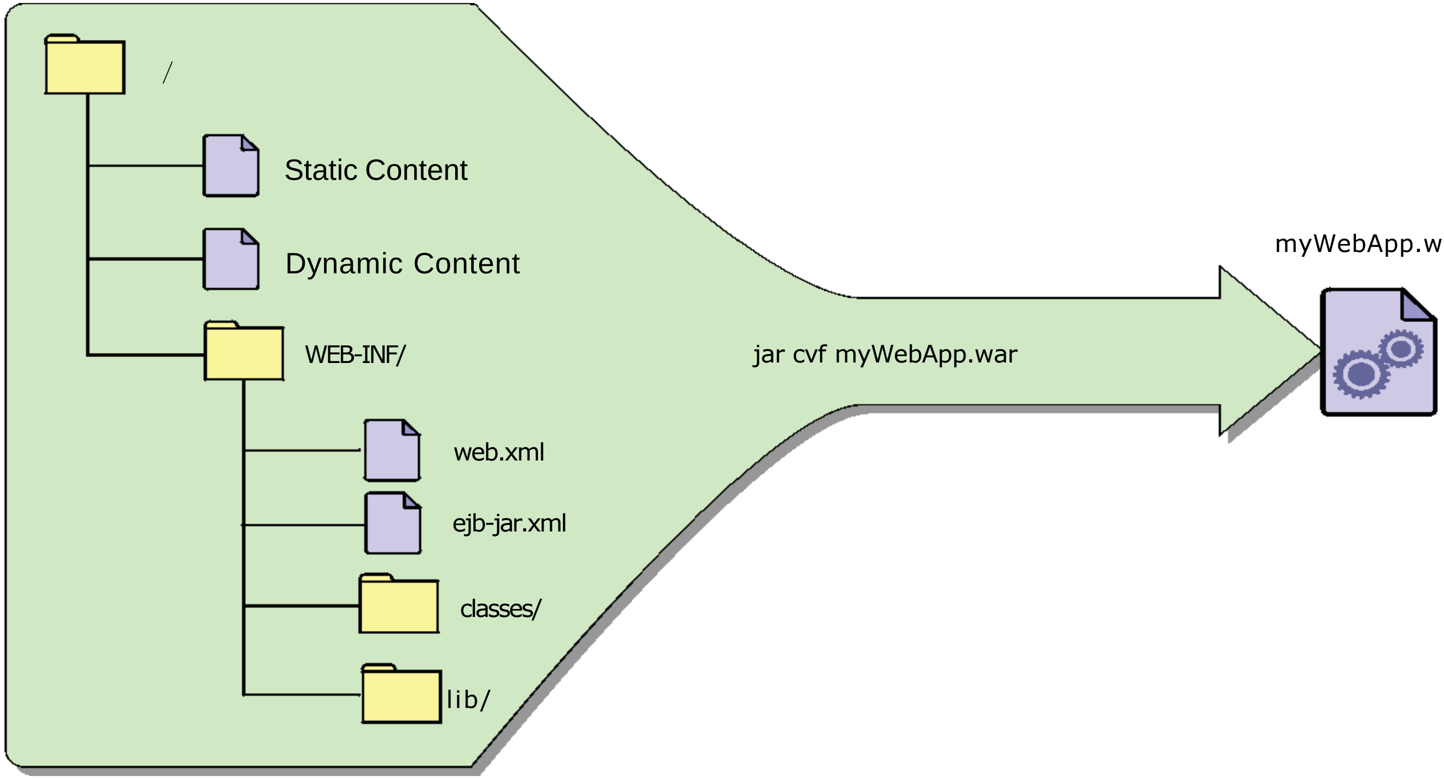
- **WAR files**
- **JAR files**
- **RAR files**
- **EAR files**

Java EE 6 allows EJB component to be packaged in a WAR file.

Web Application Elements



Web Archive File Creation



/

Static Content

Dynamic Content

WEB-INF/

web.xml

ejb-jar.xml

classes/

lib/

jar cvf myWebApp.war

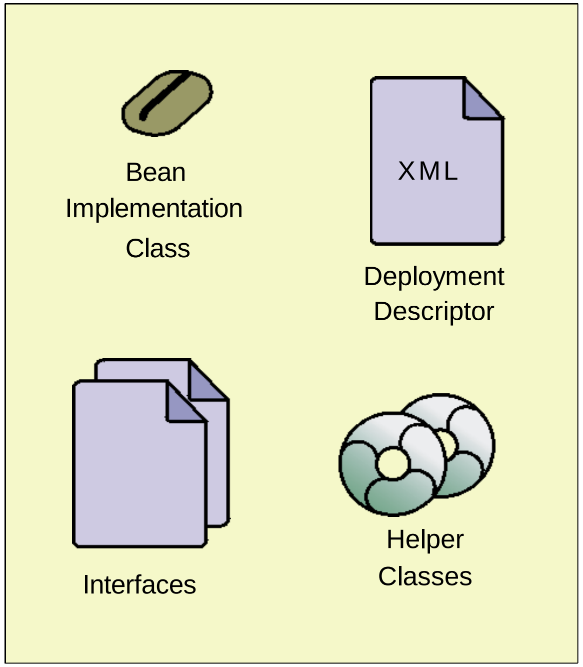
myWebApp.war

Java Archive Files

Java Archive files:

- **Provide a standard mechanism for packaging and distributing Java class files and related resources**
- **Normally given names that end in** `.jar`
- **Are defined by the Java EE specification as the packaging format for EJB components and Java EE clients**

EJB Component JAR File Contents

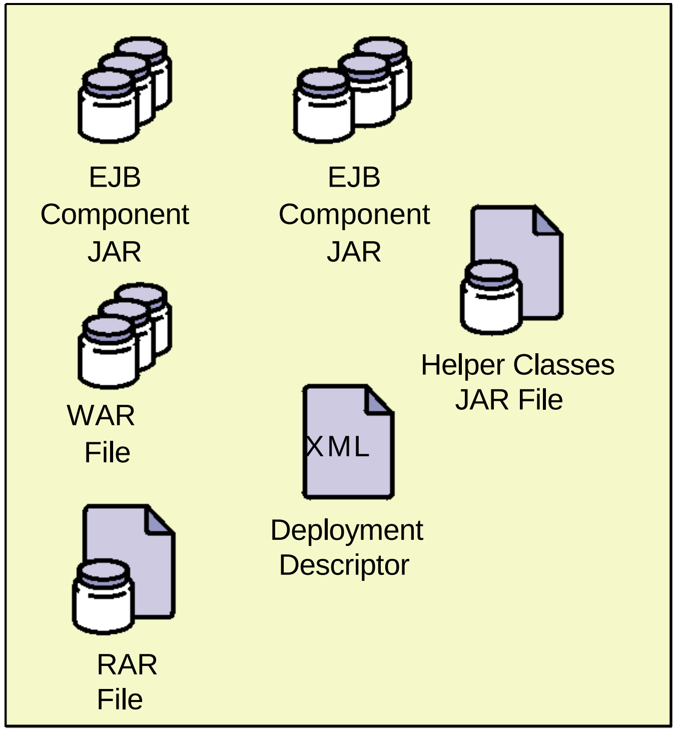


Resource Archive Files

A resource adapter:

- **Is a software component that has hooks into a container's transaction management, security, and resource pooling subsystems**
- **Can request extended access to the system, beyond what would be allowed to an enterprise bean**
- **Can make native calls, create or open network sockets that listen, create and delete threads, and read and write files**
- **Is packaged into RAR files that have names that end in** `.rar`

Enterprise Archive Files



Deployment Descriptors

Deployment descriptors:

- **Are XML-formatted files**
- **Provide a declarative way to describe the interactions between components and between a component and its container**
- **Have their format, naming convention, and other attributes defined in the relevant component specification**
- **Are not always required. In-code annotations can be used by developers.**
- **Application servers may have additional non-portable deployment descriptors to configure vendor specific features.**