Fusion

✓ Full printable view

- Java EE
- Application Development Framework
- Enterprise Application Integration
- Customization

Java Enterprise Edition (JavaEE)

With the Java^{TM} Platform, Enterprise Edition (Java EE), development of Java enterprise applications has never been easier or faster. The aim of the Java EE 5 platform is to provide developers a powerful set of APIs while reducing development time, reducing application complexity, and improving application performance.

✓ Full printable view

Contents

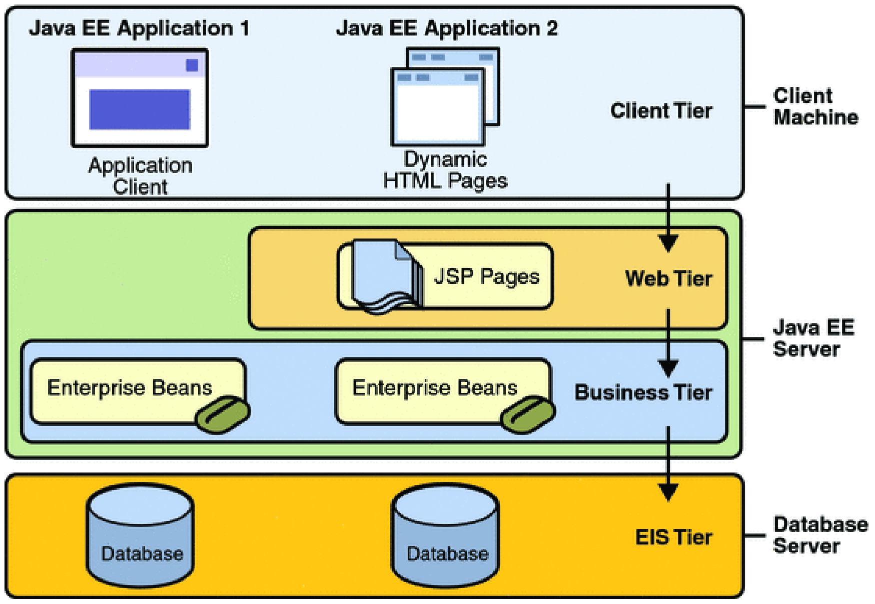
The Java EE 5 platform introduces a simplified programming model. With Java EE 5 technology, XML deployment descriptors are now optional. Instead, a developer can simply enter the information as an annotation directly into a Java source file, and the Java EE server will configure the component at deployment and runtime. These annotations are generally used to embed in a program data that would otherwise be furnished in a deployment descriptor. With annotations, the specification information is put directly in your code next to the program element that it affects.

In the Java EE platform, dependency injection can be applied to all resources that a component needs, effectively hiding the creation and lookup of resources from application code. Dependency injection can be used in EJB containers, web containers, and application clients. Dependency injection allows the Java EE container to automatically insert references to other required components or resources using annotations.

The Java Persistence API is new to the Java EE 5 platform. The Java Persistence API provides an object/relational mapping for managing relational data in enterprise beans, web components, and application clients. It can also be used in Java SE applications, outside of the Java EE environment.

Enterprise Architecture

The Java EE application model begins with the Java programming language and the Java virtual machine. The proven portability, security, and developer productivity they provide forms the basis of the application model. Java EE is designed to support applications that implement enterprise services for customers, employees, suppliers, partners, and others who make demands on or contributions to the enterprise. Such applications are inherently complex, potentially accessing data from a variety of sources and distributing applications to a variety of clients.



To better control and manage these applications, the business functions to support these various users are conducted in the middle tier. The middle tier represents an environment that is closely controlled by an enterprise's information technology department. The middle tier is typically run on dedicated server hardware and has access to the full services of the enterprise.

The Java EE application model defines an architecture for implementing services as multitier applications that deliver the scalability, accessibility, and manageability needed by enterprise-level applications. This model partitions the work needed to implement a multitier service into two parts: the business and presentation logic to be implemented by the developer, and the standard system services provided by the Java EE platform. The developer can rely on the platform to provide solutions for the hard systems-level problems of developing a multitier service.

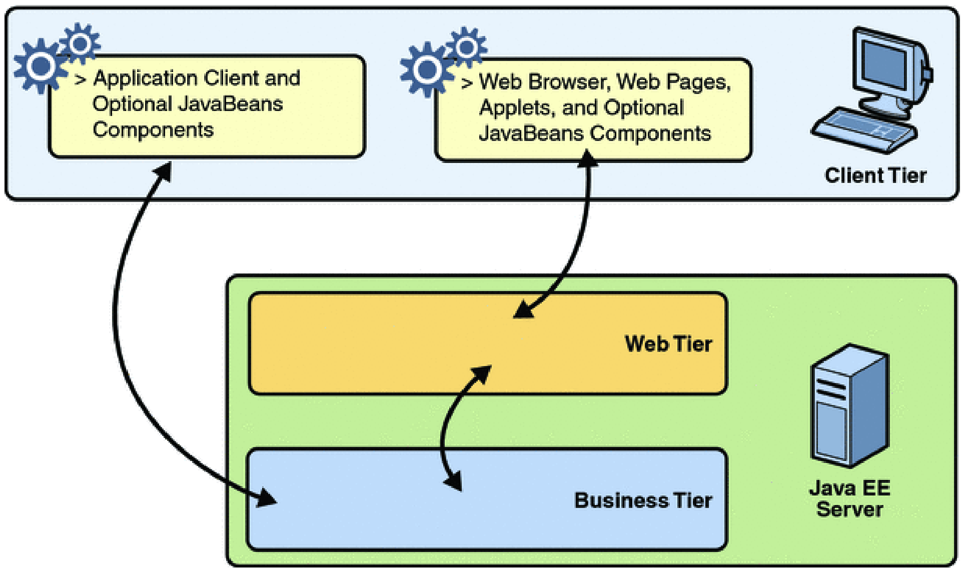
Java EE components are written in the Java programming language and are compiled in the same way as any program in the language. The difference between Java EE components and "standard" Java classes is that Java EE components are assembled into a Java EE application, are verified to be well formed and in compliance with the Java EE specification, and are deployed to production, where they are run and managed by the Java EE server.

See Also

- Distributed Multitiered Applications, The Java EE 5 Tutorial

Client Tier

A Java EE client can be a web client or an application client.


> Application Client and Optional JavaBeans Components
> Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
Web Tier
Business Tier
Java EE Server

Web Clients

A web client consists of two parts: (1) dynamic web pages containing various types of markup language (HTML, XML, and so on), which are generated by web components running in the web tier, and (2) a web browser, which renders the pages received from the server.

A web client is sometimes called a thin client. Thin clients usually do not query databases, execute complex business rules, or connect to legacy applications. When you use a thin client, such heavyweight operations are off-loaded to enterprise beans executing on the Java EE server, where they can leverage the security, speed, services, and reliability of Java EE server-side technologies.

Applets

A web page received from the web tier can include an embedded applet. An applet is a small client application written in the Java programming language that executes in the Java virtual machine installed in the web browser. However, client systems will likely need the Java Plug-in and possibly a security policy file for the applet to successfully execute in the web browser.

Web components are the preferred API for creating a web client program because no plug-ins or security policy files are needed on the client systems. Also, web components enable cleaner and more modular application design because they provide a way to separate applications programming from web page design. Personnel involved in web page design thus do not need to understand Java programming language syntax to do their jobs.

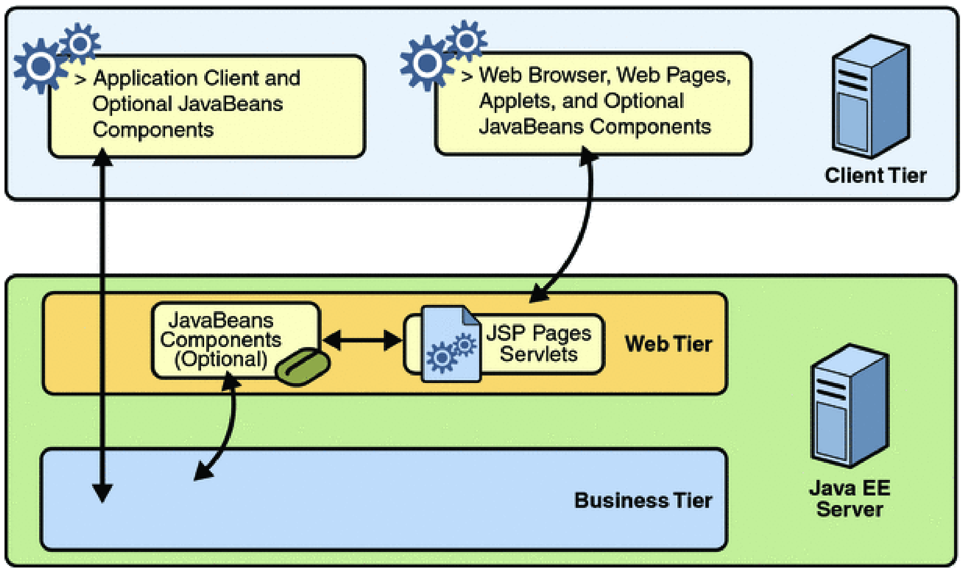
Application Clients

An application client runs on a client machine and provides a way for users to handle tasks that require a richer user interface than can be provided by a markup language. It typically has a graphical user interface (GUI) created from the Swing or the Abstract Window Toolkit (AWT) API, but a command-line interface is certainly possible.

Application clients directly access enterprise beans running in the business tier. However, if application requirements warrant it, an application client can open an HTTP connection to establish communication with a servlet running in the web tier. Application clients written in languages other than Java can interact with Java EE 5 servers, enabling the Java EE 5 platform to interoperate with legacy systems, clients, and non-Java languages.

Web Tier

Java EE application uses a thin browser-based client or thick application client. In deciding which one to use, you should be aware of the trade-offs between keeping functionality on the client and close to the user (thick client) and off-loading as much functionality as possible to the server (thin client). The more functionality you off-load to the server, the easier it is to distribute, deploy, and manage the application; however, keeping more functionality on the client can make for a better perceived user experience.


> Application Client and Optional JavaBeans Components
> Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
JavaBeans Components (Optional)
JSP Pages Servlets
Web Tier
Business Tier
Java EE Server

Web Components

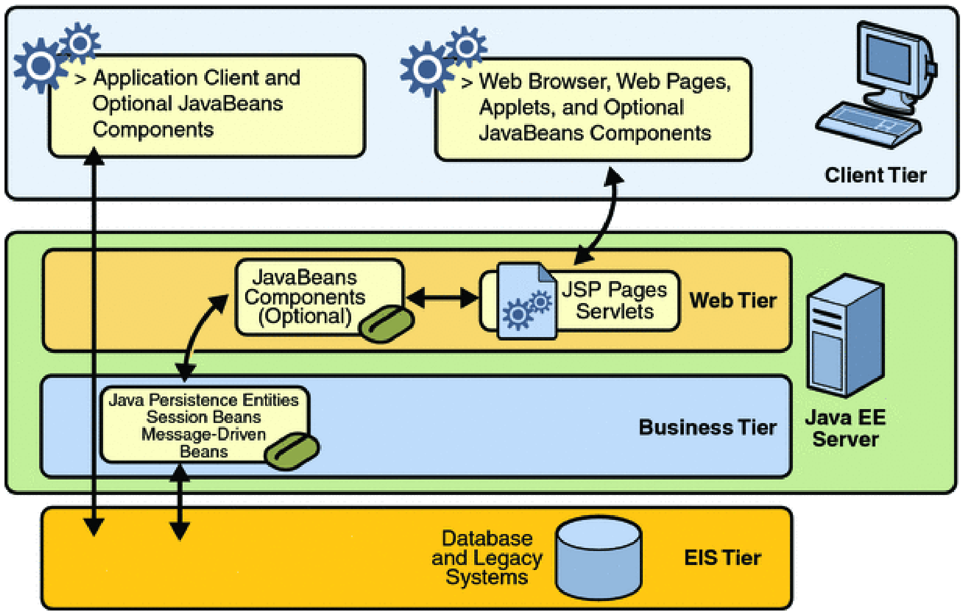
Java EE web components are either servlets or pages created using JSP technology (JSP pages) and/or ?JavaServer Faces technology. Servlets are Java programming language classes that dynamically process requests and construct responses. JSP pages are text-based documents that execute as servlets but allow a more natural approach to creating static content. ?JavaServer Faces technology builds on servlets and JSP technology and provides a user interface component framework for web applications.

Static HTML pages and applets are bundled with web components during application assembly but are not considered web components by the Java EE specification. Server-side utility classes can also be bundled with web components and, like HTML pages, are not considered web components.

Business Components and EIS

Enterprise JavaBeans^{TM} (EJBTM) components (enterprise beans) are business components that run on the server.

Business code, which is logic that solves or meets the needs of a particular business domain such as banking, retail, or finance, is handled by enterprise beans running in the business tier. The figure shows how an enterprise bean receives data from client programs, processes it (if necessary), and sends it to the enterprise information system tier for storage. An enterprise bean also retrieves data from storage, processes it (if necessary), and sends it back to the client program.

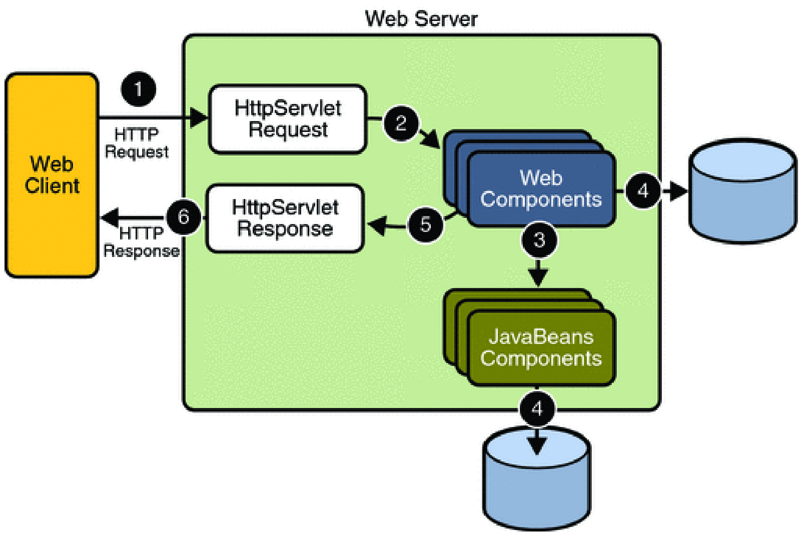

> Application Client and Optional JavaBeans Components
> Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
JavaBeans Components (Optional)
JSP Pages Servlets
Web Tier
Java Persistence Entities Session Beans Message-Driven Beans
Business Tier
Java EE Server
Database and Legacy Systems
EIS Tier

Enterprise Information System Tier

The enterprise information system tier handles EIS software and includes enterprise infrastructure systems such as enterprise resource planning (ERP), mainframe transaction processing, database systems, and other legacy information systems. For example, Java EE application components might need access to enterprise information systems for database connectivity.

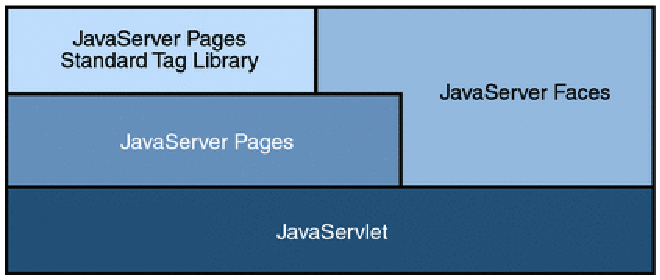
Web Applications

In the Java 2 platform, web components provide the dynamic extension capabilities for a web server. Web components are either Java servlets, JSP pages, or web service endpoints. The interaction between a web client and a web application is illustrated in Figure 3-1. The client sends an HTTP request to the web server. A web server that implements Java Servlet and JavaServer Pages technology converts the request into an `HTTPServletRequest` object. This object is delivered to a web component, which can interact with JavaBeans components or a database to generate dynamic content. The web component can then generate an `HTTPServletResponse` or it can pass the request to another web component. Eventually a web component generates a `HTTPServletResponse` object. The web server converts this object to an HTTP response and returns it to the client.



Servlets are Java programming language classes that dynamically process requests and construct responses. *JSP pages* are text-based documents that execute as servlets but allow a more natural approach to creating static content. Although servlets and JSP pages can be used interchangeably, each has its own strengths. Servlets are best suited for service-oriented applications (web service endpoints are implemented as servlets) and the control functions of a presentation-oriented application, such as dispatching requests and handling nontextual data. JSP pages are more appropriate for generating text-based markup such as HTML, Scalable Vector Graphics (SVG), Wireless Markup Language (WML), and XML.

Since the introduction of Java Servlet and JSP technology, additional Java technologies and frameworks for building interactive web applications have been developed.



Notice that Java Servlet technology is the foundation of all the web application technologies, so you should familiarize yourself with the material in Chapter 4, Java Servlet Technology even if you do not intend to write servlets. Each technology adds a level of abstraction that makes web application prototyping and development faster and the web applications themselves more maintainable, scalable, and robust.

Web components are supported by the services of a runtime platform called a *web container*. A web container provides services such as request dispatching, security, concurrency, and life-cycle management. It also gives web components access to APIs such as naming, transactions, and email.

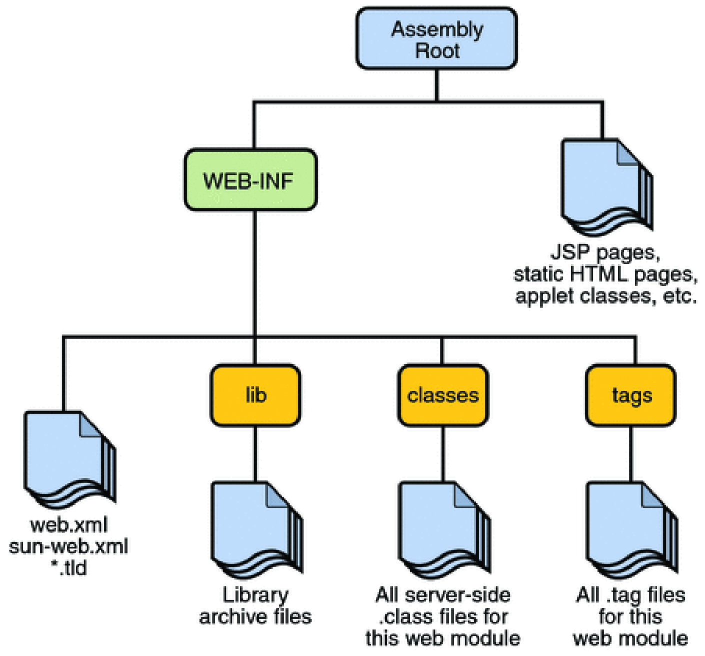
Certain aspects of web application behavior can be configured when the application is installed, or *deployed*, to the web container. The configuration information is maintained in a text file in XML format called a *web application deployment descriptor* (DD). A DD must conform to the schema described in the Java Servlet Specification.

See Also

- The Java EE 5 Tutorial

Web Modules

In the Java EE architecture, web components and static web content files such as images are called *web resources*. A *web module* is the smallest deployable and usable unit of web resources. A Java EE web module corresponds to a *web application* as defined in the Java Servlet specification.


Assembly Root
WEB-INF
JSP pages, static HTML pages, applet classes, etc.
lib
classes
tags
web.xml sun-web.xml *.tld
Library archive files
All server-side .class files for this web module
All .tag files for this web module

In addition to web components and web resources, a web module can contain other files:

Server-side utility classes (database beans, shopping carts, and so on). Often these classes conform to the ?JavaBeans component architecture.

Client-side classes (applets and utility classes).

A web module has a specific structure. The top-level directory of a web module is the *document root* of the application. The document root is where JSP pages, *client-side* classes and archives, and static web resources, such as images, are stored.

A web module can be deployed as an unpacked file structure or can be packaged in a JAR file known as a *web archive* (WAR) file. Because the contents and use of WAR files differ from those of JAR files, WAR file names use a .war extension. The web module just described is portable; you can deploy it into any web container that conforms to the Java Servlet Specification.

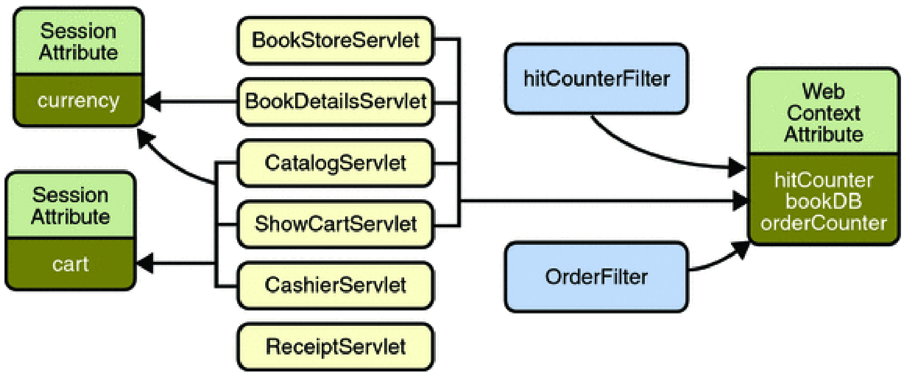
To deploy a WAR on the Application Server, the file must also contain a runtime *deployment descriptor* (DD). The runtime deployment descriptor is an XML file that contains information such as the context root of the web application and the mapping of the portable names of an application's resources to the Application Server's resources. The Application Server web application runtime DD is located in the WEB-INF directory along with the web application DD.

See Also

- Web Modules, The Java EE 5 Tutorial

Java Servlet

A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.



Sharing Information

Web components, like most objects, usually work with other objects to accomplish their tasks. There are several ways they can do this. They can use private helper objects (for example, JavaBeans components), they can share objects that are attributes of a public scope, they can use a database, and they can invoke other web resources. The Java Servlet technology mechanisms that allow a web component to invoke other web resources are described in Invoking Other Web Resources.

Using Scope Objects

Collaborating web components share information by means of objects that are maintained as attributes of four scope objects. You access these attributes using the `[get|set]Attribute` methods of the class representing the scope.

The `javax.servlet` and `javax.servlet.http` packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods. When implementing a generic service, you can use or extend the ? GenericServlet class provided with the Java Servlet API. The `HttpServlet` class provides methods, such as `doGet` and `doPost`, for handling HTTP-specific services.

See Also

- Java Servlet, The Java EE 5 Tutorial

JavaServer Pages

JavaServer Pages (JSP) technology allows you to easily create web content that has both static and dynamic components. JSP technology makes available all the dynamic capabilities of Java Servlet technology but provides a more natural approach to creating static content.

A JSP page is a text document that contains two types of text: static data, which can be expressed in any text-based format (such as HTML, SVG, WML, and XML), and JSP elements, which construct dynamic content.

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="/functions" prefix="f" %>
<html>
<head><title>Localized Dates</title></head>
<body bgcolor="white">
<jsp:useBean id="locales" scope="application"
    class="mypkg.MyLocales"/>

<form name="localeForm" action="index.jsp" method="post">
<c:set var="selectedLocaleString" value="${param.locale}" />
<c:set var="selectedFlag"
     value="${!empty selectedLocaleString}" />
<b>Locale:</b>
<c:if test="${selectedFlag}" >
    <jsp:setProperty name="locales"
        property="selectedLocaleString"
        value="${selectedLocaleString}" />
    <jsp:useBean id="date" class="mypkg.MyDate"/>
    <jsp:setProperty name="date" property="locale"
        value="${locales.selectedLocale}"/>
    <b>Date: </b>${date.date}</c:if>
</body></html>
```

The *JavaServer Pages Standard Tag Library* (JSTL) encapsulates core functionality common to many JSP applications. Instead of mixing tags from numerous vendors in your JSP applications, you employ a single, standard set of tags. This standardization allows you to deploy your applications on any JSP container that supports JSTL and makes it more likely that the implementation of the tags is optimized.

JSTL has iterator and conditional tags for handling flow control, tags for manipulating XML documents, internationalization tags, tags for accessing databases using SQL, and commonly used functions.

JavaServer Faces

The main components of JavaServer Faces technology are as follows:

A GUI component framework.

A flexible model for rendering components in different kinds of HTML or different markup languages and technologies. A Renderer object generates the markup to render the component and converts the data stored in a model object to types that can be represented in a view.

A standard RenderKit for generating HTML/4.01 markup.

The following features support the GUI components:

- Input validation
- Event handling
- Data conversion between model objects and components
- Managed model object creation
- Page navigation configuration

All this functionality is available using standard Java APIs and XML-based configuration files.

Enterprise JavaBeans

Written in the Java programming language, an enterprise bean is a server-side component that encapsulates the business logic of an application. The business logic is the code that fulfills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the business logic in methods called checkInventoryLevel and orderProduct. By invoking these methods, clients can access the inventory services provided by the application.

Benefits of Enterprise Beans

For several reasons, enterprise beans simplify the development of large, distributed applications. First, because the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems. The EJB container, rather than the bean developer, is responsible for system-level services such as transaction management and security authorization.

Second, because the beans rather than the clients contain the application’s business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases. As a result, the clients are thinner, a benefit that is particularly important for clients that run on small devices.

Third, because enterprise beans are portable components, the application assembler can build new applications from existing beans. These applications can run on any compliant Java EE server provided that they use the standard APIs.

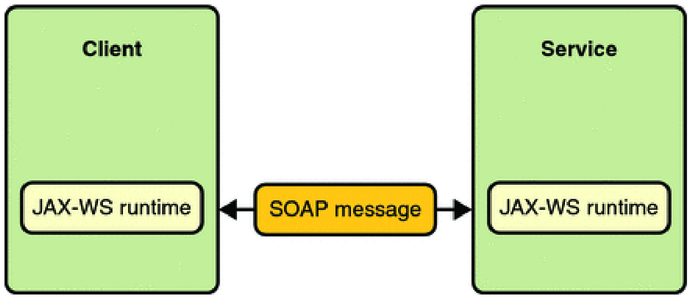
Types of Enterprise Beans

There are two kinds of enterprise beans: session beans and message-driven beans. A session bean represents a transient conversation with a client. When the client finishes executing, the session bean and its data are gone. A message-driven bean combines features of a session bean and a message listener, allowing a business component to receive messages asynchronously. Commonly, these are Java Message Service (JMS) messages.

In Java EE 5, entity beans have been replaced by Java persistence API entities. An entity represents persistent data stored in one row of a database table. If the client terminates, or if the server shuts down, the persistence manager ensures that the entity data is saved.

Java Web Services

Web services are web-based enterprise applications that use open, XML-based standards and transport protocols to exchange data with calling clients. The Java EE platform provides the XML APIs and tools you need to quickly design, develop, test, and deploy web services and clients that fully interoperate with other web services and clients running on Java-based or non-Java-based platforms.

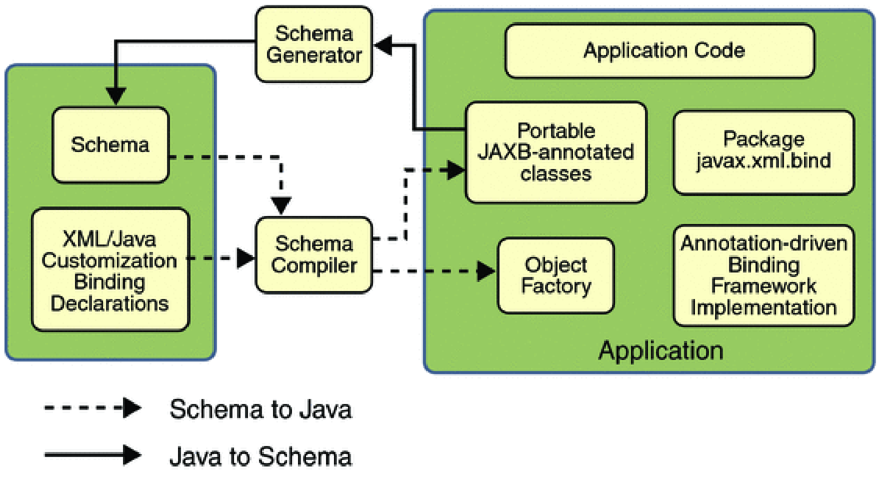


JAX-WS stands for Java API for XML Web Services. JAX-WS is a technology for building web services and clients that communicate using XML. JAX-WS allows developers to write message-oriented as well as RPC-oriented web services.

In JAX-WS, a web service operation invocation is represented by an XML-based protocol such as SOAP. The SOAP specification defines the envelope structure, encoding rules, and conventions for representing web service invocations and responses. These calls and responses are transmitted as SOAP messages (XML files) over HTTP.

JAXB Architecture

The Java^{TM} Architecture for XML Binding (JAXB) provides a fast and convenient way to bind between XML schemas and Java representations, making it easy for Java developers to incorporate XML data and processing functions in Java applications. As part of this process, JAXB provides methods for unmarshalling XML instance documents into Java content trees, and then marshalling Java content trees back into XML instance documents. JAXB also provides a way to generate XML schema from Java objects.



The general steps in the JAXB data binding process are:

Generate classes: An XML schema is used as input to the JAXB binding compiler to generate JAXB classes based on that schema.

Compile classes: All of the generated classes, source files, and application code must be compiled.

Unmarshal: XML documents written according to the constraints in the source schema are unmarshalled by the JAXB binding framework. Note that JAXB also supports unmarshalling XML data from sources other than files/documents, such as DOM nodes, string buffers, SAX Sources, and so forth.

Generate content tree: The unmarshalling process generates a content tree of data objects instantiated from the generated JAXB classes; this content tree represents the structure and content of the source XML documents.

Validate (optional): The unmarshalling process optionally involves validation of the source XML documents before generating the content tree. Note that if you modify the content tree in Step 6, below, you can also use the JAXB Validate operation to validate the changes before marshalling the content back to an XML document.

Process content: The client application can modify the XML data represented by the Java content tree by means of interfaces generated by the binding compiler.

Marshal: The processed content tree is marshalled out to one or more XML output documents. The content may be validated before marshalling.

Java Persistence

The Java Persistence API provides an object/relational mapping facility to Java developers for managing relational data in Java applications. Java Persistence consists of three areas:

- The Java Persistence API
- The query language
- Object/relational mapping metadata

An entity is a lightweight persistence domain object. Typically an entity represents a table in a relational database, and each entity instance corresponds to a row in that table. The primary programming artifact of an entity is the entity class, although entities can use helper classes.

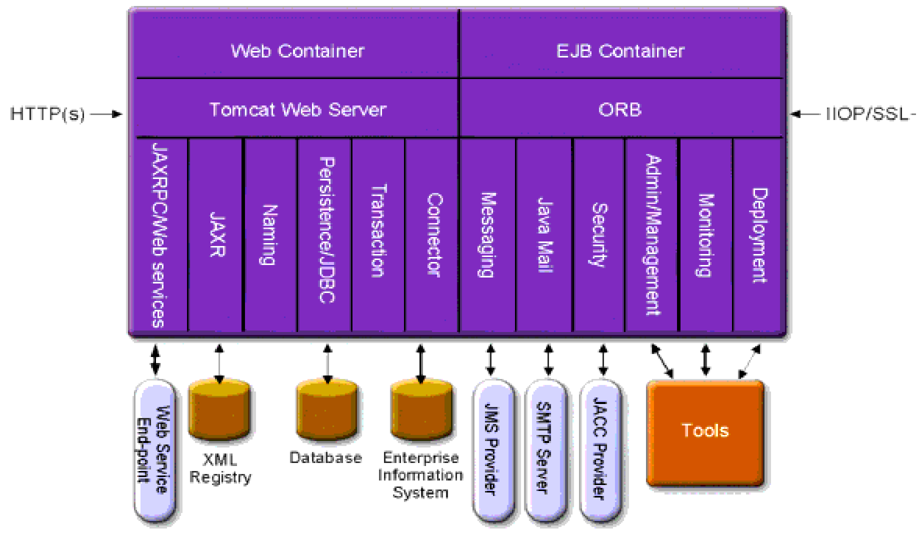
The persistent state of an entity is represented either through persistent fields or persistent properties. These fields or properties use object/relational mapping annotations to map the entities and entity relationships to the relational data in the underlying data store.

Application Server Architecture

The Application Serveris a platform that supports services from Web publishing to enterprise-scale transaction processing while enabling developers to build applications based on Java Server Pages (JSPTM), Java servlets, and Enterprise JavaBeans^{TM} (EJBTM) technology.

The Application Server 9.1 Clustering and Enterprise profiles provide advanced clustering and failover technologies. These features enable you to run scalable and highly available Java EE applications.

Application Server Architecture



Containers - A container is a runtime environment that provides services such as security and transaction management to Java EE components. Figure 1–1 shows the two types of Java EE containers: Web and EJB. Web components, such as JSP pages and servlets, run within the Web container. Enterprise beans, the components of EJB technology, run within the EJB container.

Client Access - At runtime, browser clients access Web applications by communicating with the Web server via HTTP, the protocol used throughout the internet. The HTTPS protocol is for applications that require secure communication. Enterprise bean clients communicate with the Object Request Broker (ORB) through the IIOP or IIOP/SSL (secure) protocols. The Application Server has separate listeners for the HTTP, HTTPS, IIOP, and IIOP/SSL protocols. Each listener has exclusive use of a specific port number.

Web Services - On the Java EE platform, it is possible to deploy a Web application that provides a Web service implemented by Java API for XML-Based RPC (JAX-RPC). A Java EE application or component can also be a client to other Web services. Applications access XML registries through the Java API for XML Registries (JAXR).

Services for Applications - The Java EE platform was designed so that the containers provide services for applications. Figure 1–1 shows the following services:

> *Naming* - A naming and directory service binds objects to names. A Java EE application locates an object by looking up its JNDI name. JNDI stands for the Java Naming and Directory Interface API.
> *Security* - The Java Authorization Contract for Containers (JACC) is a set of security contracts defined for the Java EE containers. Based on the client's identity, the containers restrict access to the container's resources and services.

Transaction management - A transaction is an indivisible unit of work. For example, transferring funds between bank accounts is a transaction. A transaction management service ensures that a transaction either completes fully or is rolled back.

Access to External Systems

The Java EE platform enables applications to access systems that are outside of the application server. Applications connect to these systems through objects called resources. One of the responsibilities of an administrator is resource configuration. The Java EE platform enables access to external systems through the following APIs and components:

JDBC - A database management system (DBMS) provides facilities for storing, organizing, and retrieving data. Most business applications store data in relational databases, which applications access via the JDBC API. The information in databases is often described as persistent because it is saved on disk and exists after the application ends. The Application Server bundle includes the Java DB database.

Messaging - Messaging is a method of communication between software components or applications. A messaging client sends messages to, and receives messages from, any other client. Applications access the messaging provider through the Java Messaging Service (JMS) API. The Application Server includes a JMS provider.

Connector - The Java EE Connector architecture enables integration between Java EE applications and existing Enterprise Information Systems (EIS). An application accesses an EIS through a portable Java EE component called a connector or resource adapter.

JavaMail - Through the JavaMail API, applications connect to an SMTP server in order to send and receive email.

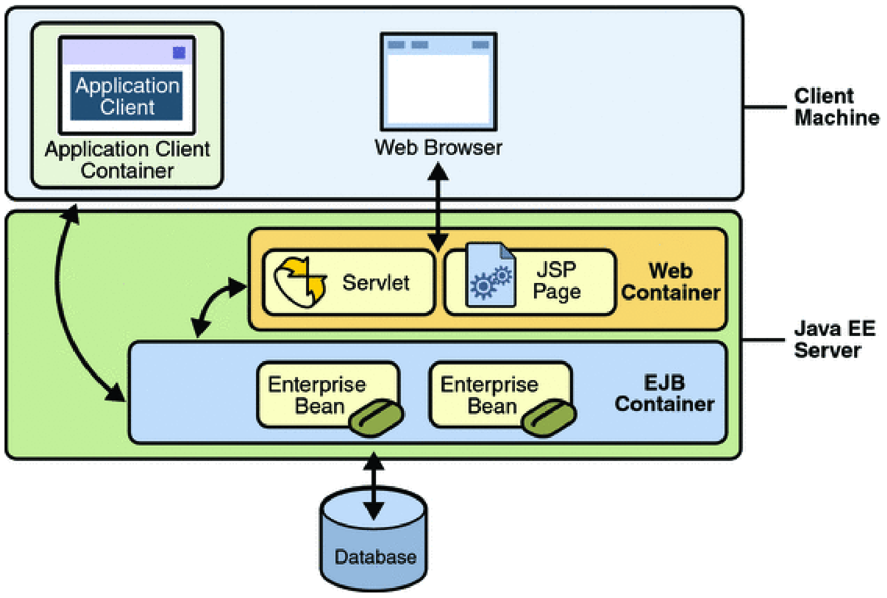
Server Administration - The lower right-hand corner of Figure 1-1 shows some of the tasks performed by the administrator of the Application Server. For example, an administrator deploys (installs) applications and monitors the server's performance. These tasks are performed with the administration tools provided by the Application Server.

See Also

- Sun Java System Application Server 9.1 Administration Guide

Java EE Containers

Containers are the interface between a component and the low-level platform-specific functionality that supports the component. Before a web, enterprise bean, or application client component can be executed, it must be assembled into a Java EE module and deployed into its container.



The assembly process involves specifying container settings for each component in the Java EE application and for the Java EE application itself. Container settings customize the underlying support provided by the Java EE server, including services such as security, transaction management, Java Naming and Directory InterfaceTM (JNDI) lookups, and remote connectivity.

Java EE server: The runtime portion of a Java EE product. A Java EE server provides EJB and web containers.

Enterprise JavaBeans (EJB) container: Manages the execution of enterprise beans for Java EE applications. Enterprise beans and their container run on the Java EE server.

Web container: Manages the execution of JSP page and servlet components for Java EE applications. Web components and their container run on the Java EE server.

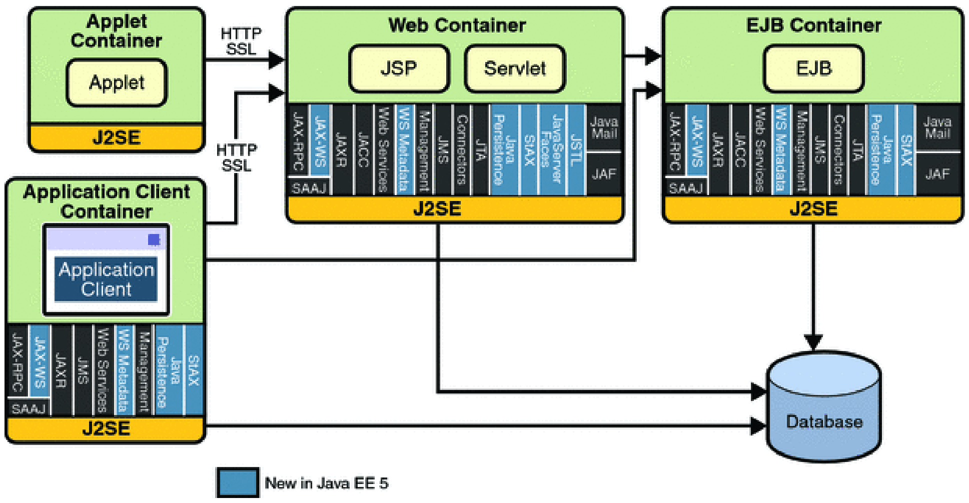
Application client container: Manages the execution of application client components. Application clients and their container run on the client.

Applet container: Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.

See Also

- Java EE Containers, The Java EE 5 Tutorial

Java EE APIs and Technologies



An Enterprise Java``Beans^{TM} (EJB) component, or enterprise bean, is a body of code having fields and methods to implement modules of business logic. You can think of an enterprise bean as a building block that can be used alone or with other enterprise beans to execute business logic on the Java EE server.

Java Servlet technology lets you define HTTP-specific servlet classes. A servlet class extends the capabilities of servers that host applications that are accessed by way of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.

JavaServer Pages^{TM} (JSP) technology lets you put snippets of servlet code directly into a text-based document. A JSP page is a text-based document that contains two types of text: static data (which can be expressed in any text-based format such as HTML, WML, and XML) and JSP elements, which determine how the page constructs dynamic content.

JavaServer Faces technology is a user interface framework for building web applications.

The Java Message Service (JMS) API is a messaging standard that allows Java EE application components to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous.

The Java Transaction API (JTA) provides a standard interface for demarcating transactions. The Java EE architecture provides a default auto commit to handle transaction commits and rollbacks. An auto commit means that any other applications that are viewing data will see the updated data after each database read or write operation. However, if your application performs two separate database access operations that depend on each other, you will want to use the JTA API to demarcate where the entire transaction, including both operations, begins, rolls back, and commits.

Java EE applications use the JavaMail^{TM} API to send email notifications. The JavaMail API has two parts: an application-level interface used by the application components to send mail, and a service provider interface. The Java EE platform includes JavaMail with a service provider that allows application components to send Internet mail.

The *JavaBeans Activation Framework* (JAF) is included because JavaMail uses it. JAF provides standard services to determine the type of an arbitrary piece of data, encapsulate access to it, discover the operations available on it, and create the appropriate JavaBeans component to perform those operations.

The *Java API for XML Processing* (JAXP), part of the Java SE platform, supports the processing of XML documents using Document Object Model (DOM), Simple API for XML (SAX), and Extensible Stylesheet Language Transformations (XSLT). JAXP enables applications to parse and transform XML documents independent of a particular XML processing implementation.

JAXP also provides namespace support, which lets you work with schemas that might otherwise have naming conflicts. Designed to be flexible, JAXP lets you use any XML-compliant parser or XSL processor from within your application and supports the W3C schema. You can find information on the W3C schema at this URL: http://www.w3.org/XML/Schema.

The *Java API for XML Web Services* (JAX-WS) specification provides support for web services that use the JAXB API for binding XML data to Java objects. The JAX-WS specification defines client APIs for accessing web services as well as techniques for implementing web service endpoints. The Web Services for J2EE specification describes the deployment of JAX-WS-based services and clients. The EJB and servlet specifications also describe aspects of such deployment. It must be possible to deploy JAX-WS-based applications using any of these deployment models.

The JAX-WS specification describes the support for message handlers that can process message requests and responses. In general, these message handlers execute in the same container and with the same privileges and execution context as the JAX-WS client or endpoint component with which they are associated. These message handlers have access to the same JNDI java:comp/env namespace as their associated component. Custom serializers and deserializers, if supported, are treated in the same way as message handlers.

The *Java Architecture for XML Binding* (JAXB) provides a convenient way to bind an XML schema to a representation in Java language programs. JAXB can be used independently or in combination with JAX-WS, where it provides a standard data binding for web service messages. All Java EE application client containers, web containers, and EJB containers support the JAXB API.

The *SOAP with Attachments API for Java* (SAAJ) is a low-level API on which JAX-WS and JAXR depend. SAAJ enables the production and consumption of messages that conform to the SOAP 1.1 specification and SOAP with Attachments note. Most developers do not use the SAAJ API, instead using the higher-level JAX-WS API.

The *Java API for XML Registries* (JAXR) lets you access business and general-purpose registries over the web. JAXR supports the ebXML Registry and Repository standards and the emerging UDDI specifications. By using JAXR, developers can learn a single API and gain access to both of these important registry technologies.

Additionally, businesses can submit material to be shared and search for material that others have submitted. Standards groups have developed schemas for particular kinds of XML documents; two businesses might, for example, agree to use the schema for their industry's standard purchase order form. Because the schema is stored in a standard business registry, both parties can use JAXR to access it.

The *J2EE Connector architecture* is used by tools vendors and system integrators to create resource adapters that support access to enterprise information systems that can be plugged in to any Java EE product. A resource adapter is a software component that allows Java EE application components to access and interact with the underlying resource manager of the EIS. Because a resource adapter is specific to its resource manager, typically there is a different resource adapter for each type of database or enterprise information system.

The J2EE Connector architecture also provides a performance-oriented, secure, scalable, and message-based transactional integration of Java EE-based web services with existing EISs that can be either synchronous or asynchronous. Existing applications and EISs integrated through the J2EE Connector architecture into the Java EE platform can be exposed as XML-based web services by using JAX-WS and Java EE component models. Thus JAX-WS and the J2EE Connector architecture are complementary technologies for enterprise application integration (EAI) and end-to-end business integration.

The *Java Database Connectivity* (JDBC) API lets you invoke SQL commands from Java programming language methods. You use the JDBC API in an enterprise bean when you have a session bean access the database. You can also use the JDBC API from a servlet or a JSP page to access the database directly without going through an enterprise bean.

The JDBC API has two parts: an application-level interface used by the application components to access a database, and a service provider interface to attach a JDBC driver to the Java EE platform.

The *Java Persistence* API is a Java standards-based solution for persistence. Persistence uses an object-relational mapping approach to bridge the gap between an object oriented model and a relational database. Java Persistence consists of three areas:

- The Java Persistence API
- The query language
- Object/relational mapping metadata

The *Java Naming and Directory Interface*™ (JNDI) provides naming and directory functionality, enabling applications to access multiple naming and directory services, including existing naming and directory services such as LDAP, NDS, DNS, and NIS. It provides applications with methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes. Using JNDI, a Java EE application can store and retrieve any type of named Java object, allowing Java EE applications to coexist with many legacy applications and systems.

The *Java Authentication and Authorization Service* (JAAS) provides a way for a Java EE application to authenticate and authorize a specific user or group of users to run it.

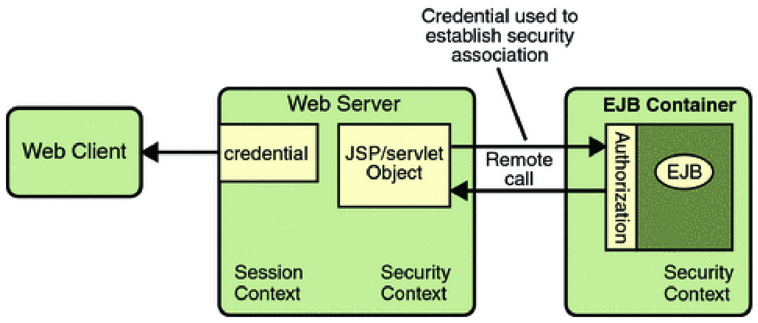
JAAS is a Java programming language version of the standard Pluggable Authentication Module (PAM) framework, which extends the Java Platform security architecture to support user-based authorization.

Simplified Systems Integration. The Java EE platform is a platform-independent, full systems integration solution that creates an open marketplace in which every vendor can sell to every customer. Such a marketplace encourages vendors to compete, not by trying to lock customers into their technologies but instead by trying to outdo each other in providing products and services that benefit customers, such as better performance, better tools, or better customer support.

Java Security

While other enterprise application models require platform-specific security measures in each application, the Java EE security environment enables security constraints to be defined at deployment time. The Java EE platform makes applications portable to a wide variety of security implementations by shielding application developers from the complexity of implementing security features.

The Java EE platform provides standard declarative access control rules that are defined by the developer and interpreted when the application is deployed on the server. Java EE also provides standard login mechanisms so application developers do not have to implement these mechanisms in their applications. The same application works in a variety of different security environments without changing the source code.



Characteristics of Application Security

Java EE applications consist of components that can contain both protected and unprotected resources. Often, you need to protect resources to ensure that only authorized users have access. Authorization provides controlled access to protected resources. Authorization is based on identification and authentication. Identification is a process that enables recognition of an entity by a system, and authentication is a process that verifies the identity of a user, device, or other entity in a computer system, usually as a prerequisite to allowing access to resources in a system.

Authorization and authentication are not required for an entity to access unprotected resources. Accessing a resource without authentication is referred to as unauthenticated or anonymous access.

These and several other well-defined characteristics of application security that, when properly addressed, help to minimize the security threats faced by an enterprise, include the following:

Authentication: The means by which communicating entities (for example, client and server) prove to one another that they are acting on behalf of specific identities that are authorized for access. This ensures that users are who they say they are.

Authorization, or Access Control: The means by which interactions with resources are limited to collections of users or programs for the purpose of enforcing integrity, confidentiality, or availability constraints. This ensures that users have permission to perform operations or access data.

Data integrity: The means used to prove that information has not been modified by a third party (some entity other than the source of the information). For example, a recipient of data sent over an open network must be able to detect and discard messages that were modified after they were sent. This ensures that only authorized users can modify data.

Confidentiality or Data Privacy: The means used to ensure that information is made available only to users who are authorized to access it. This ensures that only authorized users can view sensitive data.

Non-repudiation: The means used to prove that a user performed some action such that the user cannot reasonably deny having done so. This ensures that transactions can be proven to have happened.

Quality of Service (QoS): The means used to provide better service to selected network traffic over various technologies.

Auditing: The means used to capture a tamper-resistant record of securityrelated events for the purpose of being able to evaluate the effectiveness of security policies and mechanisms. To enable this, the system maintains a record of transactions and security information.

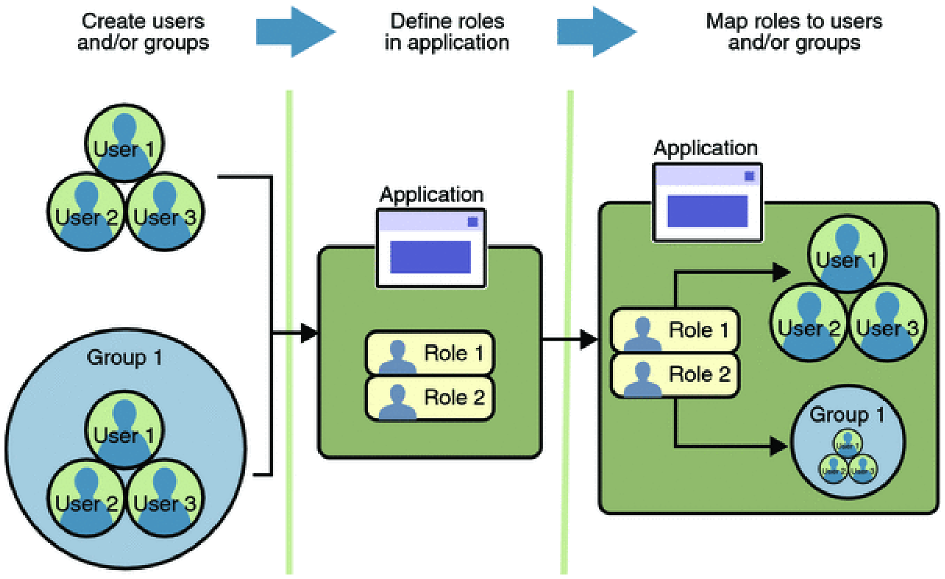
Realms, Users, Groups, and Roles

For a web application, a *realm* is a complete database of users and groups that identify valid users of a web application (or a set of web applications) and are controlled by the same authentication policy.

A *user* is an individual (or application program) identity that has been defined in the Application Server. In a web application, a user can have a set of roles associated with that identity, which entitles them to access all resources protected by those roles. Users can be associated with a group.

A *group* is a set of authenticated users, classified by common traits, defined in the Application Server.

A *role* is an abstract name for the permission to access a particular set of resources in an application. A role can be compared to a key that can open a lock. Many people might have a copy of the key. The lock doesn’t care who you are, only that you have the right key.



Principal: A principal is an entity that can be authenticated by an authentication protocol in a security service that is deployed in an enterprise. A principal is identified using a principal name and authenticated using authentication data.

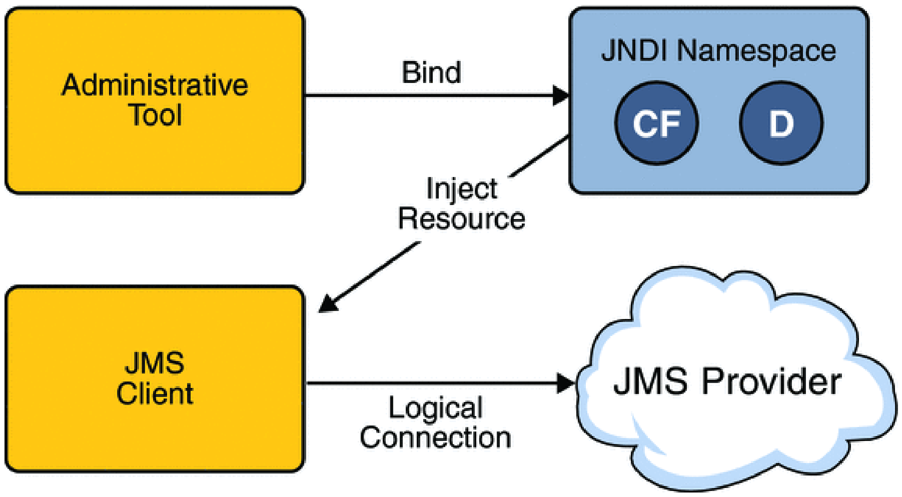
Security policy domain (also known as security domain or realm): A security policy domain is a scope over which a common security policy is defined and enforced by the security administrator of the security service.

Security attributes: A set of security attributes is associated with every principal. The security attributes have many uses, for example, access to protected resources and auditing of users. Security attributes can be associated with a principal by an authentication protocol.

Credential: A credential contains or references information (security attributes) used to authenticate a principal for Java EE product services. A principal acquires a credential upon authentication, or from another principal that allows its credential to be used.

Java Message Services

Messaging is a method of communication between software components or applications. A messaging system is a peer-to-peer facility: A messaging client can send messages to, and receive messages from, any other client. Each client connects to a messaging agent that provides facilities for creating, sending, receiving, and reading messages.

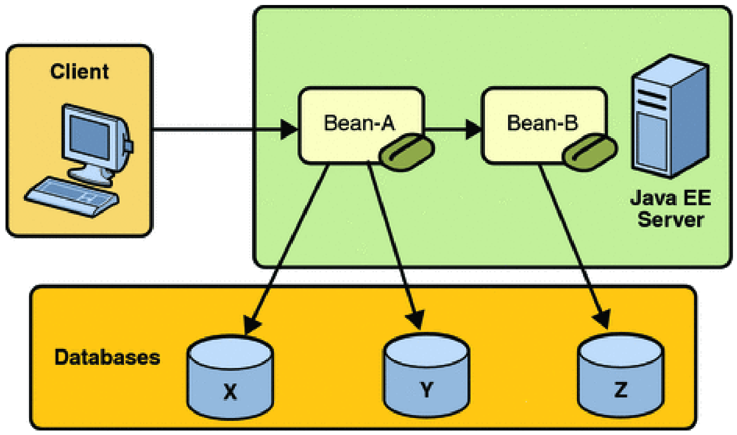


Messaging enables distributed communication that is loosely coupled. A component sends a message to a destination, and the recipient can retrieve the message from the destination. However, the sender and the receiver do not have to be available at the same time in order to communicate. In fact, the sender does not need to know anything about the receiver; nor does the receiver need to know anything about the sender. The sender and the receiver need to know only which message format and which destination to use. In this respect, messaging differs from tightly coupled technologies, such as Remote Method Invocation (RMI), which require an application to know a remote application’s methods.

Messaging also differs from electronic mail (email), which is a method of communication between people or between software applications and people. Messaging is used for communication between software applications or software components.

Transactions

A typical enterprise application accesses and stores information in one or more databases. Because this information is critical for business operations, it must be accurate, current, and reliable. Data integrity would be lost if multiple programs were allowed to update the same information simultaneously. It would also be lost if a system that failed while processing a business transaction were to leave the affected data only partially updated. By preventing both of these scenarios, software transactions ensure data integrity. Transactions control the concurrent access of data by multiple programs. In the event of a system failure, transactions make sure that after recovery the data will be in a consistent state.

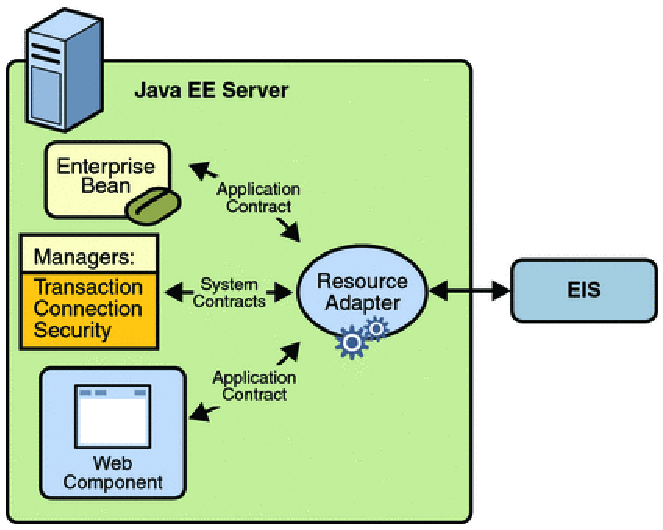


In an enterprise bean with *container-managed transaction demarcation*, the EJB container sets the boundaries of the transactions. You can use container-managed transactions with any type of enterprise bean: session, or message-driven. Container-managed transactions simplify development because the enterprise bean code does not explicitly mark the transaction’s boundaries. The code does not include statements that begin and end the transaction.

In *bean-managed transaction demarcation*, the code in the session or message-driven bean explicitly marks the boundaries of the transaction. Although beans with container-managed transactions require less coding, they have one limitation: When a method is executing, it can be associated with either a single transaction or no transaction at all. If this limitation will make coding your bean difficult, you should consider using bean-managed transactions.

Connector Architecture

The Connector architecture enables Java EE components to interact with enterprise information systems (EISs) and EISs to interact with Java EE components. EIS software includes various types of systems: enterprise resource planning (ERP), mainframe transaction processing, and nonrelational databases, among others. Connector architecture simplifies the integration of diverse EISs. Each EIS requires only one implementation of the Connector architecture. Because an implementation adheres to the Connector specification, it is portable across all compliant Java EE servers.



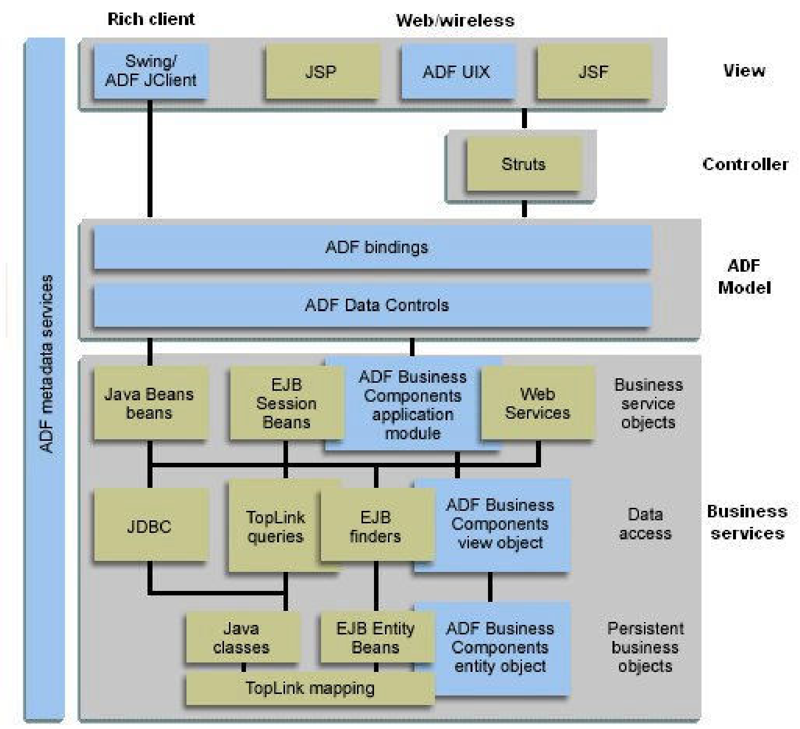
Stored in a *Resource Adapter Archive* (RAR) file, a resource adapter can be deployed on any Java EE server, much like the EAR file of a Java EE application. An RAR file may be contained in an EAR file, or it may exist as a separate file. See Figure 35-2 for the structure of a resource adapter module.

A resource adapter is analogous to a JDBC driver. Both provide a standard API through which an application can access a resource that is outside the Java EE server. For a resource adapter, the outside resource is an EIS; for a JDBC driver, it is a DBMS. Resource adapters and JDBC drivers are rarely created by application developers. In most cases, both types of software are built by vendors that sell products such as tools, servers, or integration software.

Application Development Framework

- ADF Architecture
- Visual Declarative Development
- Data Binding
- ADF Business Layer
  - Entity Objects
  - Polymorphic Entities
- ADF Views
  - ADF Faces
  - ADF Facets
- ADF Controller
- ADF Validation and Conversion
- ADF Design Patterns

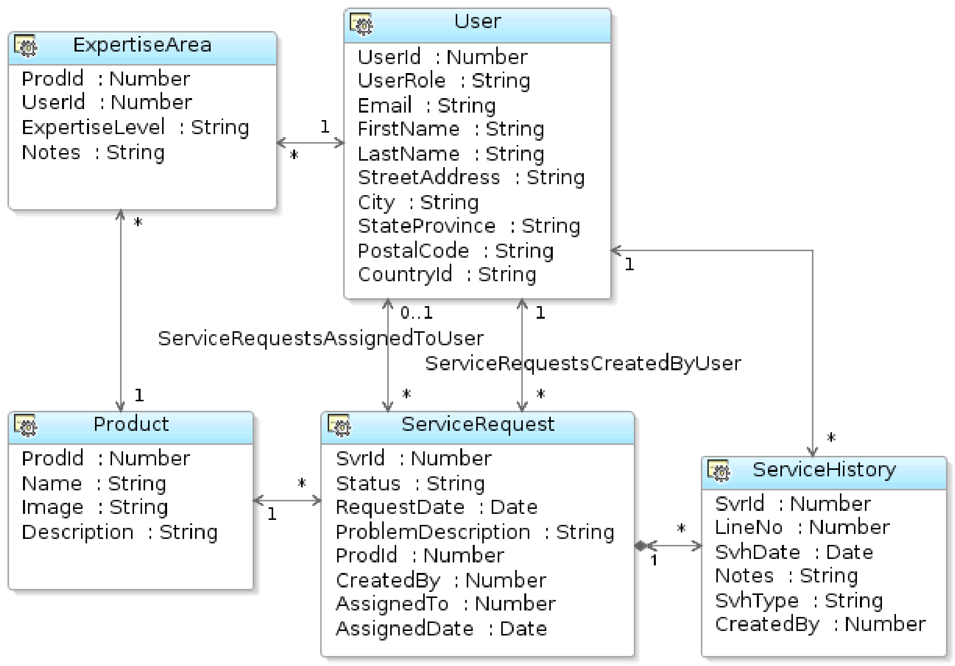
ADF Architecture



Oracle ADF provides out-of-the-box support for data controls based on Java classes, EJB Session Beans, Web Services and ADF Application Modules. Like other ADF components you may already be familiar with, both ADF Bindings and ADF Data Controls are configured using XML metadata so a small set of framework base classes can handle most of your application development needs without coding. When needs dictate, you can either customize the base framework classes or provide completely new data control and binding implementations.

Visual Declarative Development

Entity Diagram

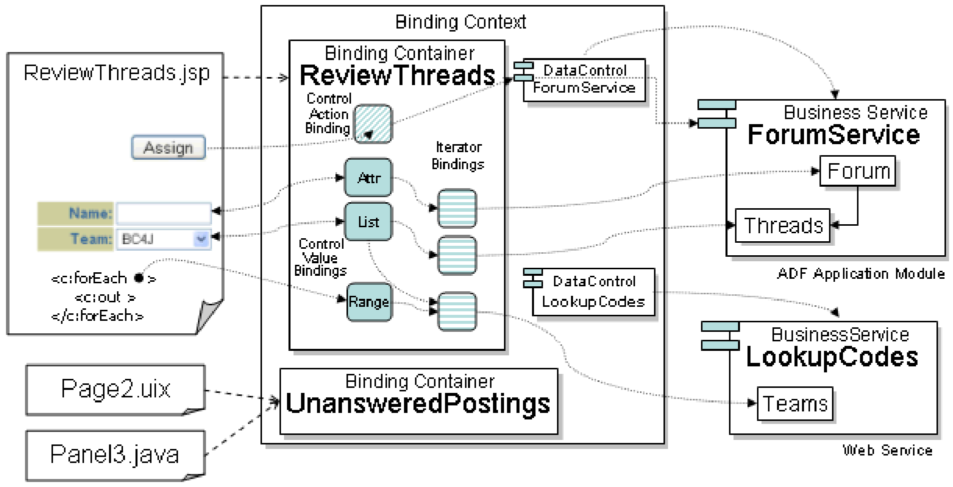


Since your layer of business domain objects represents a key reusable asset to your team, it is often convenient to visualize it using a UML model. JDeveloper supports easily creating a diagram for your business domain layer that you and your colleagues can use for reference.

See Also

- Creating an Entity Diagram for Your Business Layer

Data Binding



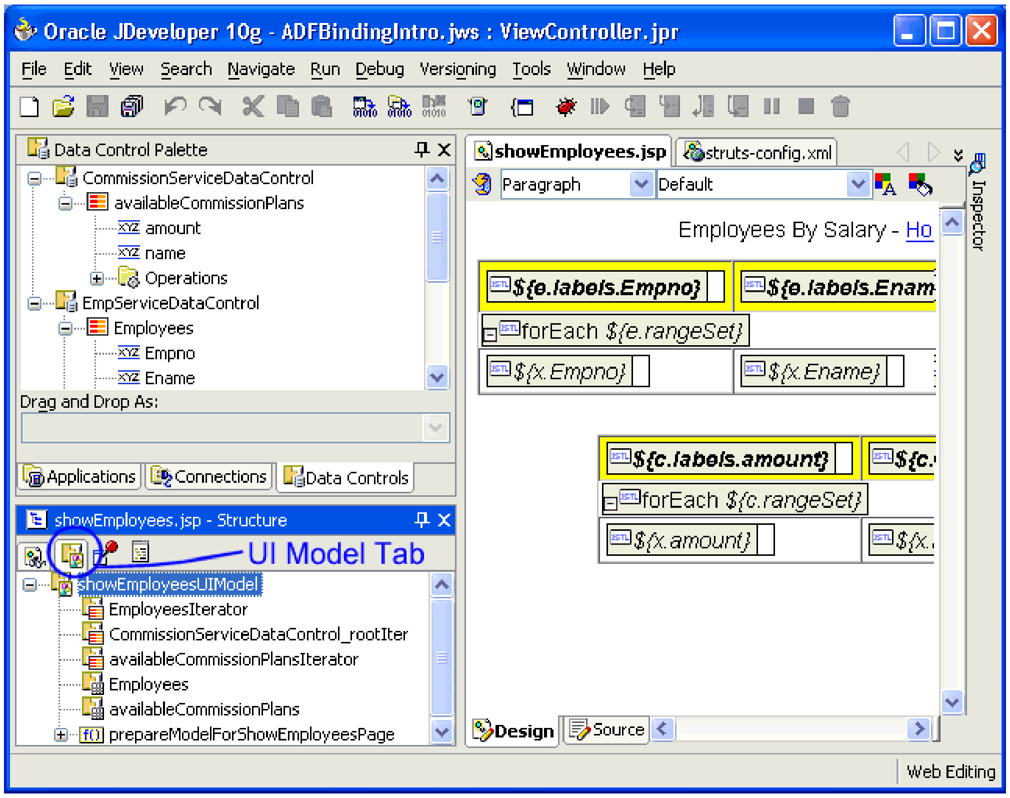
The key data binding concepts in Oracle ADF are the following:

Data Controls
: A data control abstracts the implementation of a business service, allowing the binding layer to access the data from all services in a consistent way.

Iterator Bindings and Control Bindings
: Bindings are lightweight objects that decouple back-end data and front-end UI display. An iterator binding provides a consistent way to work with a collection of data objects supplied by a data control. Control bindings provide a standard interface for UI components to interact with an iterator's data or to invoke "action" methods for preparing model data and handling events. Bindings also expose key metadata to simplify building dynamic, multi-lingual user interfaces.

Binding Containers
: A binding container is a named group of of related iterator and control bindings that you use together for a particular page (or panel) of your application. A binding container is also known as a "UI Model" since it provides the appropriate subset of model data for a specific UI.

Binding Context
: The binding context provides the data environment for your application. It contains all of the data controls and binding containers that your application can access.



When you drag and drop data-bound UI elements from the Data Control Palette to either the Visual Editor, Structure Pane, or the Code Editor, JDeveloper 10g will implicitly create bindings and a binding container to hold them if necessary.

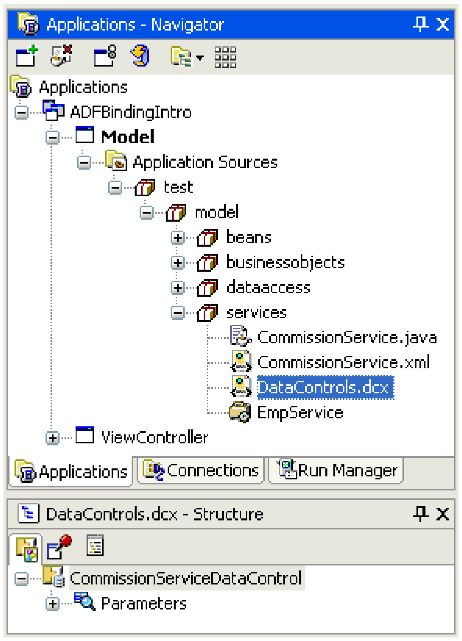
See Also

- Oracle ADF Data Binding Primer and ADF/Struts Overview

ADF Business Layer

By default, any ADF Business Components application modules that you create are accessible as data controls. You need to explicitly create other kinds of data controls by selecting the Java Class, EJB Session Bean, or Web Service in the Application Navigator and either:

Choosing Create Data Control from the context menu, or Dragging and dropping the item onto the Data Control Palette The ? DataControls.dcx file appears automatically in any project where you've created some data controls for business services that are not implemented using ADF application modules. The file stores a few pieces of basic metadata needed by the runtime framework about your data controls.



Based on Standard Java and XML

Like the rest of Oracle ADF, the ADF Business Components technology is implemented in Java. The base components implement a large amount of generic, metadata-driven functionality to save you development time by standing on the shoulders of a rich layer of working, tested code. The metadata for ADF Business Components follow J2EE community best practice of using cleanly-separated XML files to hold the metadata that configures each component's runtime behavior.

Works with Any Application Server or Database

Because your business components are implemented using plain Java classes and XML files, you can use them in any runtime environment where a Java Virtual Machine is present. This means that services built using ADF Business Components are easy to use both inside a J2EE server — known as the "container" of your application at runtime — as well as outside. Customers routinely use application modules in such diverse configurations as command-line batch programs, web services, custom servlets, JSP pages, desktop-fidelity clients built using Swing, and others.

Implements All of the J2EE Design Patterns You Need

The ADF Business Components layer implements all of the popular J2EE design patterns that you would normally need to understand, implement, and debug yourself to create a real-world enterprise J2EE application

Components are Organized into Packages

Since ADF Business Components is implemented in Java, it is implemented in Java classes and interfaces that are organized into packages. Java packages are identified by dot-separated names that developers use to arrange code into a hierarchical naming structure. To ensure your code won't clash with reusable code from other organizations, best practice dictates choosing package names that begin with your organization's name or web domain name.

Architecture of the Base ADF Business Components Layer

The classes and interfaces that comprise the pre-built code provided by the ADF Business Components layer live in the oracle.jbo package and numerous subpackages, however in your day to day work with ADF Business Components you'll mostly be working with classes and interfaces in the two key packages oracle.jbo and oracle.jbo.server. The oracle.jbo package contains all of the interfaces that are designed for the business service client to work with, while the oracle.jbo.server package contains the classes that implement these interfaces.

Components Are Metadata-Driven With Optional Custom Java Code

Each kind of component in ADF Business Components comes with built-in runtime functionality that you control through declarative settings. These settings are stored in an XML component definition file with the same name as the component that it represents. When you need to write custom code for a component, you can enable an optional custom Java class for the component in question.

Configuring ADF Business Components Design Time Preferences

You can configure whether JDeveloper generates custom Java files by default for each component type that supports it, as well as whether JDeveloper maintains a list of Oracle ADF business components in each package using a package XML file. This section describes Oracle's recommendations to developers getting started with ADF Business Components on how to configure these options.

Basic Datatypes

The Java language provides a number of built-in data types for working with strings, dates, numbers, and other data. When working with ADF Business Components, you can use these types, but by default you'll use an optimized set of types in the oracle.jbo.domain and oracle.ord.im packages.

Generic Versus Strongly-Typed APIs

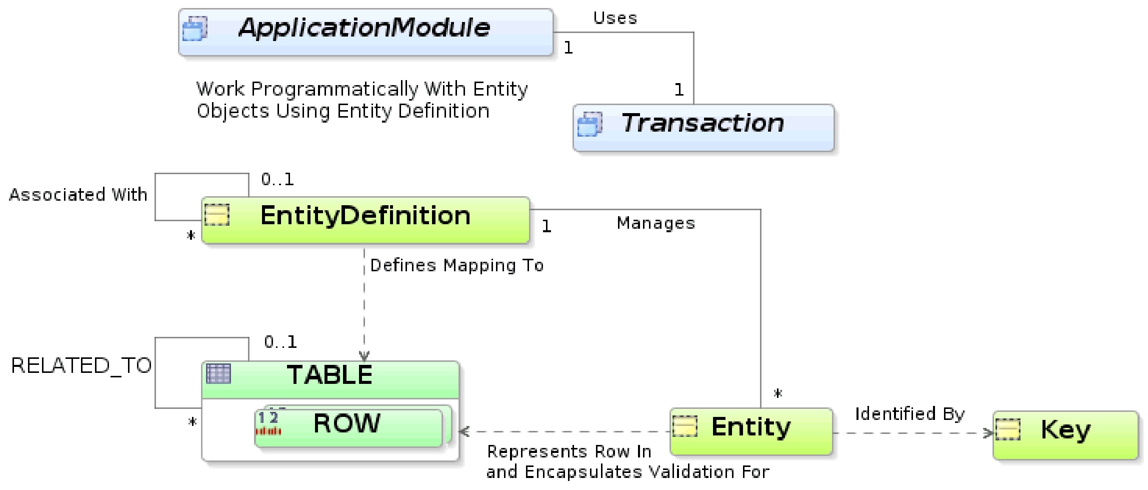
When working with application modules, view objects, and entity objects, you can choose to use a set of generic APIs or can have JDeveloper generate code into a custom Java class to enable a strongly-typed API for that component.

Client-Accessible Components Can Have Custom Interfaces

By design, the entity objects in the business domain layer of business service implementation are not designed to be referenced directly by clients. Instead, clients work with the data queried by view objects as part of an application module's data model.

Entity Objects

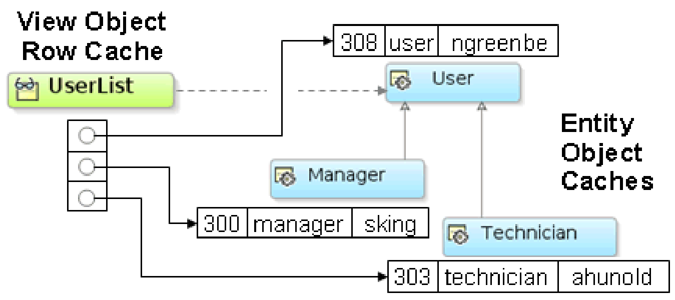
An entity object is the ADF Business Components component that represents a row in a database table and simplifies modifying its data. Importantly, it allows you to encapsulate domain business logic for those rows to ensure that your business policies and rules are consistently validated. By the end of this chapter, you'll understand the concepts shown here:



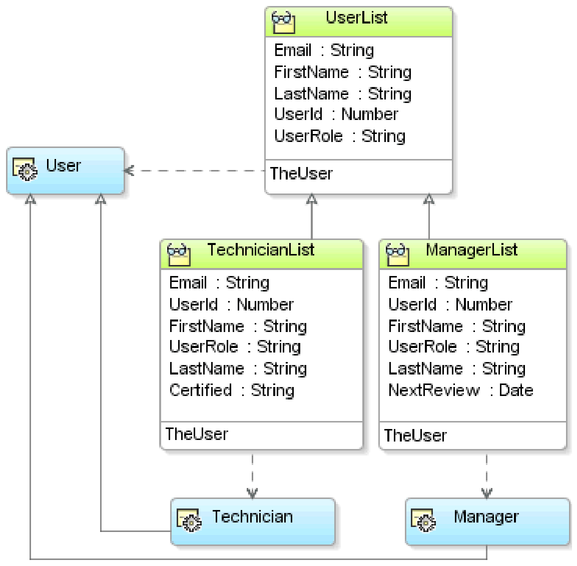
- You define an entity object by specifying the database table whose rows it will represent.
- You can associate an entity object with others to reflect relationships between underlying database tables.
- At runtime, entity rows are managed by a related entity definition object.
- Each entity rows is identified by a related row key.
- You retrieve and modify entity rows in the context of an application module that provides the database transaction.

Polymorphic Entities

A *polymorphic entity* usage is one that references a base entity object in an inheritance hierarchy and is configured to handle subtypes of that entity as well. The figure shows the results of using a view object with a polymorphic entity usage. The entity-based ?UserList view object has the User entity object as its primary entity usage. The view object partitions each row retrieved from the database into an entity row part of the appropriate entity object subtype of User. It creates the appropriate entity row subtype based on consulting the value of the discriminator attribute. For example, if the ?UserList query retrieves one row for user ngreenbe, one row for manager sking, and one row for technician ahunold, the underlying entity row parts would be as shown in the figure.



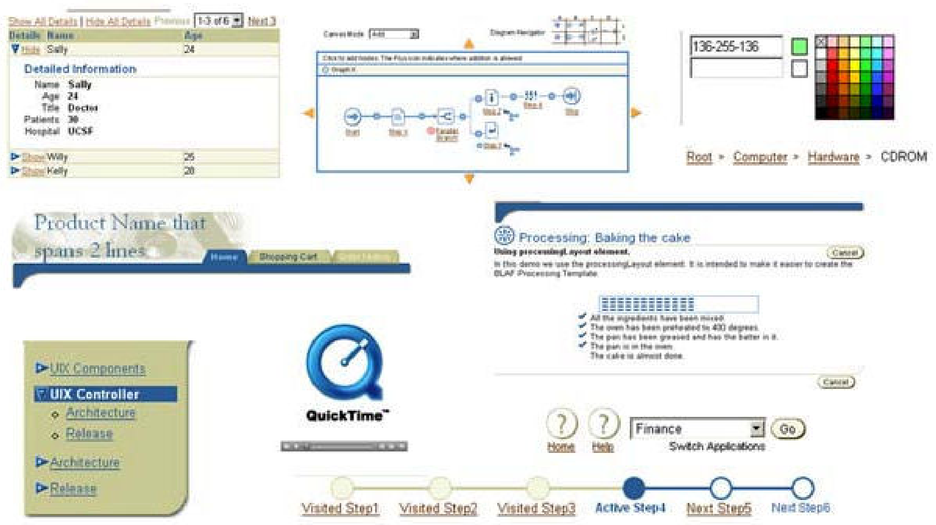
Inheritance is a powerful feature of object-oriented development that can simplify development and maintenance when used appropriately. As you've seen in Section "Creating Extended Components Using Inheritance", ADF Business Components supports using inheritance to create new components that extend existing ones in order to add additional properties or behavior or modify the behavior of the parent component. This section helps you understand when inheritance can be useful in modeling the different kinds of entities in your reusable business domain layer.



See Also

- Using View Objects to Work with Multiple Row Types, Oracle® Application Development Framework Developer's Guide For Forms/4GL Developers

ADF Views



See Also

- Introduction to ADF Faces

ADF Faces

Oracle ADF Faces is a 100% ?JavaServer Faces (JSF) compliant component library that offers a broad set of enhanced UI components for JSF application development. Based on the JSF JSR 127 specification, ADF Faces components can be used in any IDE that supports JSF. More specifically, ADF Faces works with Sun's JSF Reference Implementation 1.1_01 (or later) and Apache ?MyFaces 1.0.8 (or later).

ADF Faces ensures a consistent look and feel for your application, allowing you to focus more on user interface interaction than look and feel compliance. The component library supports multi–language and translation implementations, and accessibility features. ADF Faces also supports multiple render kits for HTML, mobile, and telnet users—this means you can build web pages with the same components, regardless of the device that will be used to display the pages.

Using the partial-page rendering features of ADF Faces components, you can build interactive web pages that update the display without requiring a complete page refresh. In the future, Oracle plans to provide render kits that make even more sophisticated use of AJAX technologies—?JavaScript, XML, and the Document Object Model (DOM)—to deliver more Rich Internet Applications with interactivity nearing that of desktop-style applications.

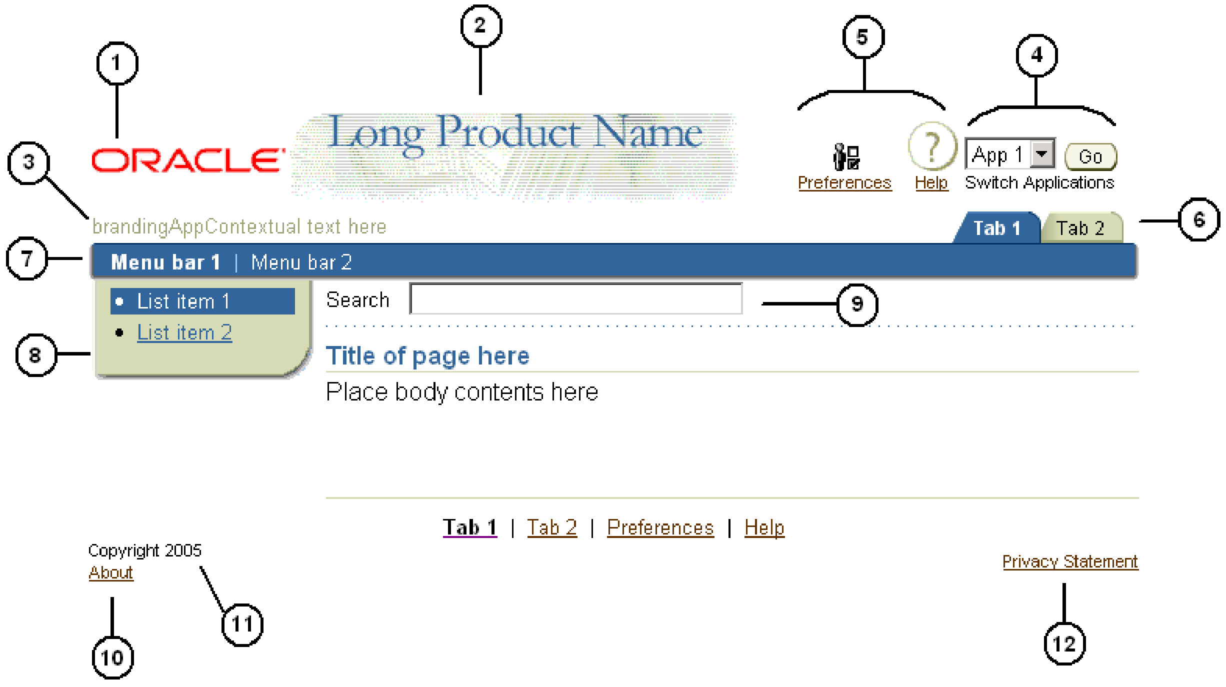
ADF Faces UI components include advanced tables with column sorting and row selection capability, tree components for displaying data hierarchically, color and date pickers, and a host of other components such as menus, command buttons, shuttle choosers, and progress meters.

ADF Faces out-of-the-box components simplify user interaction, such as the input file component for uploading files, and the select input components with built-in dialog support for navigating to secondary windows and returning to the originating page with the selected values.

See Also

- Introduction to ADF Faces

ADF Facets

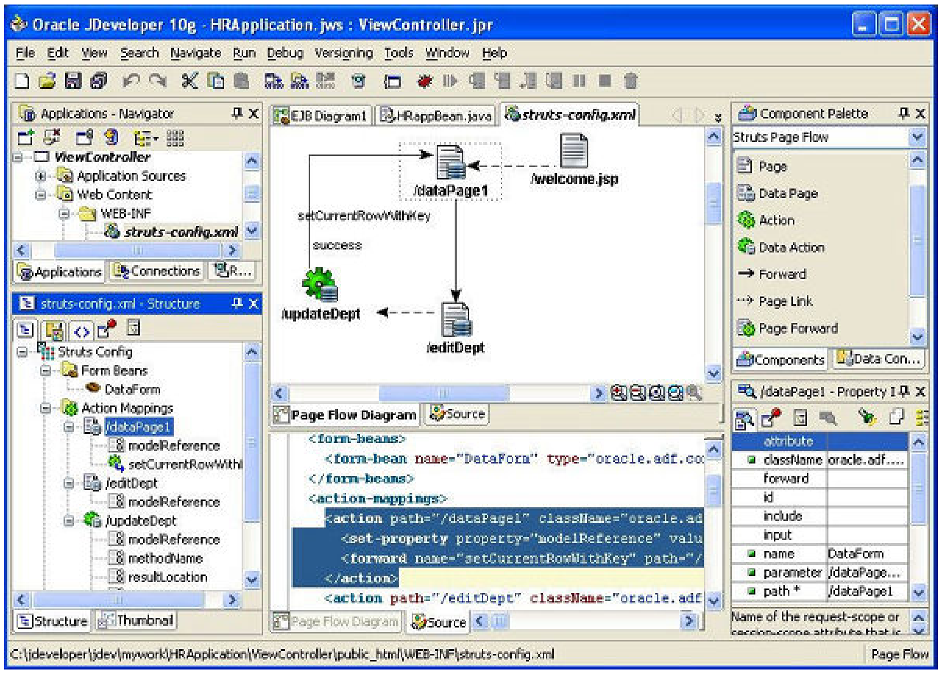


The panelPage component uses facets (or JSF f:facet tags) to render children components in specific, predefined locations on the page. Consider a facet as a placeholder for one child component. Each facet has a name and a purpose, which determines where the child component is to be rendered relative to the parent component. The child component is often a container component for other child components.

The panelPage component uses menu1, menu2, and menu3 facets for creating hierarchical, navigation menus that enable users to go quickly to related pages in the application.

ADF Controller

Navigation through a JSF application is defined by navigation rules. These rules determine, based on outcomes specified by UI components, which page is displayed next when the UI component is clicked.



```
  <form-beans>
    <form-bean name="DataForm" type="oracle.adf.co
  </form-beans>
  <action-mappings>
    <action path="/dataPage1" className="oracle.ad
      <set-property property="modelReference" valu
      <forward name="setCurrentRowWithKey" path="/
    </action>
    <action path="/editDept" className="oracle.adf
```

Defining page navigation for an application is a two-step process:

- First, you create navigation rules for all the pages in your application. In most cases, you define one rule for each page in your application. However, you can also define pattern-based rules that affect groups of pages or global rules that affect all pages.
- Next, in each navigation component on the pages, such as a command button or link, you specify either a static or dynamic outcome value in the action attribute.

  Static outcome values are an explicit reference to a specific outcome defined in a navigation rule. *Dynamic* outcome values are derived from a binding on a backing bean method that returns an outcome value. In either case, the outcome value specified in the action attribute must match an outcome defined in the navigation rules or be handled by a default navigation rule for navigation to occur.

See Also

- Adding Page Navigation, Application Development Framework Developer's Guide For Forms/4GL Developers

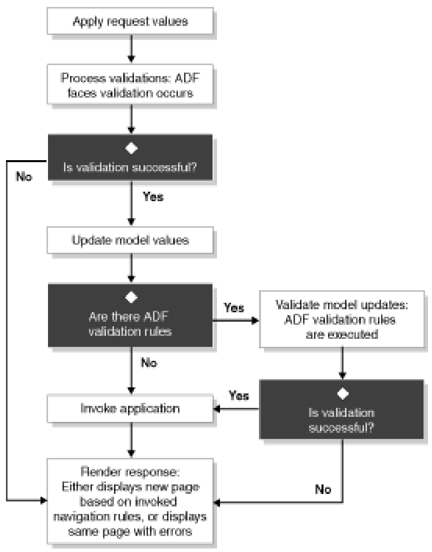
ADF Validation and Conversion

In an ADF Business Components application, virtually all validation code is defined in the shared, reusable business domain layer of entity objects. This ensures that your business information is validated consistently in every page where end users are allowed to make changes, and it simplifies maintenance by centralizing validation.

In the model layer, ADF Model validation rules can be set for the attributes of a collection. In an ADF Business Components application, unless you use data controls other than the application module data controls, you don't need to add ADF Model validation rules.

the view layer, ADF Faces input components have built-in validation capabilities. You set validation on a UI component either by setting the required attribute or by using one of the built-in ADF Faces validators. ADF applications also have validation capabilities at the model layer, allowing you to set validation on a binding to an attribute. In addition, you can create your own ADF Faces validators to suit your business needs.

Validation and Conversion in the Lifecycle


Apply request values
Process validations: ADF faces validation occurs
Is validation successful?
No
Yes
Update model values
Are there ADF validation rules
Yes
Validate model updates: ADF validation rules are executed
No
Invoke application
Yes
Is validation successful?
Render response: Either displays new page based on invoked navigation rules, or displays same page with errors
No

When a form with data is submitted, the browser sends a request value to the server for each UI component whose value attribute is bound. The request value is first stored in an instance of the component in the JSF Apply Request Values phase. If the value requires conversion (for example, if it is displayed as a String but stored as a ?DateTime object), the data is converted to the correct type. Then, if you set ADF Faces validation for any of the components that hold the data, the value is validated against the defined rules during the Process Validations phase, before the value is applied to the model.

If validation or conversion fails, the lifecycle proceeds to the Render Response phase and a corresponding error message is displayed on the page. If validation and conversion are successful, then the ?UpdateModel phase starts and the validated and converted values are used to update the model.

ADF Design Patterns

By using the Oracle Application Development Framework's business components building-blocks and related design-time extensions to JDeveloper, you get a prescriptive architecture for building richly-functional and cleanly layered J2EE business services with great performance. Below is a brief overview of the numerous design patterns that the ADF Business Components layer implements for you. Some are the familiar patterns from Sun's J2EE Blueprints, and some are design patterns that ADF Business Components adds to the list.

- Model/View/Controller
- Interface / Implementation Separation
- Service Locator
- Inversion of Control
- Dependency Injection
- Active Record
- Data Access Objects
- Session Facade
- Value Object
- Page-by-Page Iterator
- Fast-Lane Reader
- (Bean) Factory
- Entity Facade
- Value Messenger
- Continuations

See Also

- J2EE Design Patterns Implemented by ADF Business Components

Enterprise Application Integration

✔ Full printable view

- ?Enterprise Server Bus

- ?BPEL
- ?Portals

Customizable Solutions

✔ Full printable view

- ?Customizable UI
- ?Enterprise Scripting

Fusion (Full) (last edited 2008-01-29 21:49:29 by OlegKobchenko)