

MODULE 3: J2EE ARCHITECTURE OVERVIEW

WHAT IS J2EE?

- Standard architecture specification to give control in maximizing capabilities described in module 2 (i.e. Availability, Security, Reliability, Scalability, ...)
- Delivered as the following elements:
 - **J2EE Platform** - A standard platform for hosting J2EE applications, specified as a set of required APIs and policies.
 - **J2EE Compatibility Test Suite** - A suite of compatibility tests for verifying that a J2EE platform product is compatible with the J2EE platform standard.
 - **J2EE Reference Implementation** - A reference implementation for demonstrating the capabilities of J2EE and for providing an operational definition of the J2EE platform.
 - **Sun Blueprints™ Design Guidelines for J2EE** - Guidelines that describe a standard programming model for developing multi-tier, thin-client applications.

J2EE OVERVIEW

The Java 2 platform, Enterprise Edition reduces the cost and complexity of developing multi-tier services, resulting in services that can be rapidly deployed and easily enhanced as the enterprise responds to competitive pressures.

The Java 2 platform, Enterprise Edition (J2EE) achieves these benefits by defining a standard architecture that is delivered as the following elements:

- J2EE Platform - A standard platform for hosting J2EE applications, specified as a set of required APIs and policies.
- J2EE Compatibility Test Suite - A suite of compatibility tests for verifying that a J2EE platform product is compatible with the J2EE platform standard.
- J2EE Reference Implementation - A reference implementation for demonstrating the capabilities of J2EE and for providing an operational definition of the J2EE platform.
- Sun Blueprints™ Design Guidelines for J2EE - Guidelines that describe a standard programming model for developing multi-tier, thin-client applications.

J2EE APPLICATION MODEL

The J2EE application model is being provided as part of the Sun BluePrints™ best practices program:

- Java Technology Foundation
 - J2SE and JVM portability, security, and developer productivity.
 - JavaBeans component model.
- Platform-Independent Security
 - Security constraints (roles access control rules etc.)are defined at deployment time.
 - The same program works in a variety of different security environments without change to the source code.
 - In most cases, neither the service nor its clients require developer-written security logic.
- Emphasis on business logic tier
 - BL is implemented as Enterprise JavaBean components.
 - JSP technology and servlets present BL to the client tier.

J2EE APPLICATION ARCHITECTURES V. 1.1 © RESOLUTION OY, MAY 2002

J2EE APPLICATION MODEL

J2EE 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 MIDDLE TIER

The major benefit of the J2EE application model is in the middle tiers of multi-tier applications. In the J2EE platform, middle-tier business functions are implemented as Enterprise JavaBean components. These enterprise beans allow service developers to concentrate on the business logic and let the EJB server handle the complexities of delivering a reliable, scalable service.

PLATFORM-INDEPENDENT SECURITY

While other enterprise application models require platform-specific security measures in each application, the J2EE platform's security environment enables security constraints to be defined at deployment time. By shielding applications from the complexity of implementing security, the J2EE platform makes them portable to a wide variety of security implementations.

The J2EE platform defines standard declarative access control rules to be defined by the application programmer/ assembler and interpreted when the application is deployed on the enterprise platform. J2EE also requires platform vendors to supply standard login mechanisms so applications do not have to incorporate these mechanisms into their logic. The same program works in a variety of different security environments without change to the source code.

As an example, a J2EE application developer can specify several levels of security (say user, super-user, and administrator), then write code to check the current user's permission level when accessing secure operations. At deployment time, the Application Deployer assigns groups of users to the appropriate security levels, enabling the application to easily verify permission level before performing the restricted operations.

THE ENTERPRISE INFORMATION SYSTEMS

The business functions must access and update the information in the EIS-tier. The following standard Java service APIs provide basic access to these systems:

JDBC - the standard API for accessing relational data from Java.

Java Naming and Directory Interface (JNDI) - the standard API for accessing information in enterprise name and directory services.

Java Message Service (JMS) - the standard API for sending and receiving messages via enterprise messaging systems like IBM MQ Series and TIBCO Rendezvous.

JavaMail - the standard API for sending E-mail.

JavaIDL - the standard API for calling CORBA services.

SERVICE TECHNOLOGIES

Alternative view to the EIS integration (as well seen in Sun documentation) is to look at J2EE platform service technologies. They allow applications to access a wide range of services in a uniform manner. This note page describes technologies that provide access to databases, transactions, XML processing, naming and directory services, and enterprise information systems.

JDBC API

The JDBC API provides database-independent connectivity between the J2EE platform and a wide range of tabular data sources. JDBC technology allows an application component provider to:

- Perform connection and authentication to a database server
- Manage transactions
- Move SQL statements to a database engine for preprocessing and execution
- Execute stored procedures
- Inspect and modify the results from Select statements.

The J2EE platform requires both the JDBC 2.0 Core API (included in the J2SE platform), and the JDBC 2.0 Extension API, which provides row sets, connection naming via JNDI, connection pooling, and distributed transaction support. The connection pooling and distributed transaction features are intended for use by JDBC drivers to coordinate with a J2EE server.

JAVA TRANSACTION API AND SERVICE

The Java Transaction API (JTA) allows applications to access transactions in a manner that is independent of specific implementations. JTA specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the transactional application, the J2EE server, and the manager that controls access to the shared resources affected by the transactions. The Java Transaction Service (JTS) specifies the implementation of a transaction manager that supports JTA and implements the Java mapping of the Object Management Group Object Transaction Service 1.1 specification.

JAVA NAMING AND DIRECTORY INTERFACE

The Java Naming and Directory Interface (JNDI) API provides naming and directory functionality. It provides applications with methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes. Applications can use JNDI to access multiple naming and directory services, including existing naming and directory services such as LDAP, NDS, DNS, and NIS. This allows applications to coexist with legacy applications and systems.

J2EE CONNECTOR ARCHITECTURE

The J2EE Connector architecture is a standard API for connecting the J2EE platform to enterprise information systems, such as enterprise resource planning, mainframe transaction processing, and database systems. The architecture addresses the issues involved when integrating existing enterprise information systems (EIS), such as SAP, CICS, legacy applications, and non-relational databases, with an EJB server and enterprise applications. The J2EE Connector architecture defines a set of scalable, secure, and transactional mechanisms for integrating an EIS with a J2EE platform. The J2EE Connector architecture:

- Defines system contracts between J2EE-compliant application servers and resource adapters providing connectivity between J2EE application components and an EIS.
- Defines a common set of APIs so that Java applications and tools vendors can connect to and use an EIS through its resource adapters.
- Defines a standard packaging and deployment facility for resource adapters to facilitate their deployment in a J2EE environment.

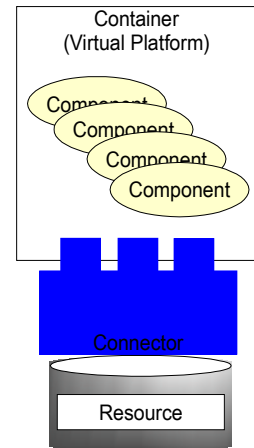
JAVA API FOR XML PROCESSING TECHNOLOGY

The Java API for XML Processing (JAXP) technology supports the processing of XML documents using DOM, SAX, and XSLT. JAXP enables applications to parse and transform XML documents independent of a particular XML processing implementation. Depending on the needs of the application, developers have the flexibility to swap between XML processors, such as between high-performance or memory-conservative parsers, with no application code changes.

CONNECTOR ARCHITECTURE

The J2EE application model divides enterprise applications into three fundamental parts:

- **Components**
 - The key focus of application developers
- **Containers**
 - Implemented by system vendors
 - Provide vertical services to components
 - Allows component behaviors to be specified at deployment time.
- **Connectors**
 - Sit beneath the J2EE platform
 - Define a portable service API to plug into vendor products.
 - Promote flexibility by enabling a variety of implementations of specific services.



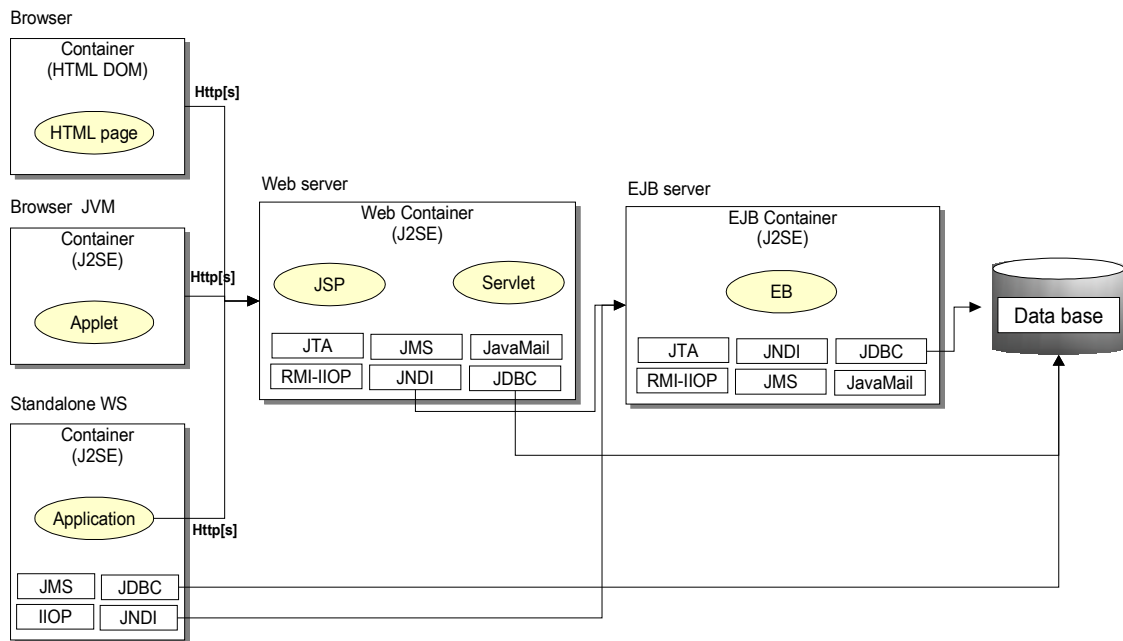
J2EE APPLICATION ARCHITECTURES V. 1.1 © RESOLUTION OY, MAY 2002

CONTAINER-BASED COMPONENT

Management Central to the J2EE component-based development model is the notion of containers. Containers are standardized runtime environments that provide specific services to components. Components can expect these services to be available on any J2EE platform from any vendor. For example, all J2EE Web containers provide runtime support for responding to client requests, performing request-time processing (such as invoking JSP pages or servlet behavior), and returning results to the client. In addition, they provide APIs to support user session management. All EJB containers provide automated support for transaction and life cycle management of EJB components, as well as bean lookup and other services. Containers also provide standardized access to enterprise information systems; for example, providing access to relational data through the JDBC API. In addition, containers provide a mechanism for selecting application behaviors at assembly or deployment time. Through the use of deployment descriptors (XML files that specify component and container behavior), components can be configured to a specific container's environment when deployed, rather than in component code. Features that can be configured at deployment time include security checks, transaction control, and other management responsibilities.

While the J2EE specification defines the component containers that a platform implementation must support, it doesn't specify or restrict the containers' configurations. Thus, both container types can run on a single platform, Web containers can live on one platform and EJB containers on another, or a J2EE platform can be made up of multiple containers on multiple platforms.

J2EE CONTAINERS AND API's



J2EE APPLICATION ARCHITECTURES V. 1.1 © RESOLUTION OY, MAY 2002

SUPPORT FOR CLIENT COMPONENTS

The J2EE client tier provides support for a variety of client types, both within the enterprise firewall and outside. Clients can be offered through Web browsers by using plain HTML pages, HTML generated dynamically by JavaServer Pages™ (JSP™) technology, or Java applets. Clients can also be offered as standalone Java language applications. J2EE clients are assumed to access the middle tier primarily using Web standards, namely HTTP, HTML, and XML. Because of its flexible programming model, the J2EE platform can support a number of simple application models implemented primarily on the strengths of its Web tier component technologies.

SUPPORT FOR BUSINESS LOGIC COMPONENTS

While simple J2EE applications may be built largely in the client tier, business logic is often implemented on the J2EE platform in the middle tier as Enterprise JavaBeans components (also known as enterprise beans). Enterprise beans allow the component or application developer to concentrate on the business logic while the complexities of delivering a reliable, scalable service are handled by the EJB container. In many ways, the J2EE platform and EJB architecture have complementary goals. The EJB component model is the backbone of industrial-strength application architectures in the J2EE programming model. The J2EE platform complements the EJB specification by:

- Fully specifying the APIs that an enterprise bean developer can use to implement enterprise beans, and by
- Defining the larger, distributed programming environment in which enterprise beans are used as business logic components.

J2EE API's

- Java Naming and Directory Interface (JNDI) API
 - Mechanism for J2EE components to look up other objects they require.
- Java Data Base Connectivity (JNDI) API
 - Management of relational databases
- JavaMail API
 - Provides the ability to send and receive e-mail.
- CORBA Compliance (JavaIDL and RMI-IIOP).
 - JavaIDL enables interaction with any CORBA-compliant system.
 - RMI- IIOP combines RMI API with CORBA's IIOP protocol
- Java Transaction API
 - Provides a way to manage transactions in own code.
- XML Deployment Descriptors
 - Helps implementing customizable components and custom tools.
- Java Message Service
 - Standard mechanism for components to send and receive messages asynchronously, for fault-tolerant interaction.

STANDARD PATTERNS USED BY J2EE

To support architectural capabilities J2EE architecture uses a set of standard design patterns.

- Knowing these pattern help extending J2EE to real life projects better
- J2EE itself is a good case example of a good architecture

Patterns used:

- Proxy
 - A separate implementation of interface for location independence
- Decorator
 - The same contract with added services and functionality
- Factory Method
 - A contract for creating objects without specifying implementation
- Abstract Factory
 - A contact for creating a family of objects without specifying implementation

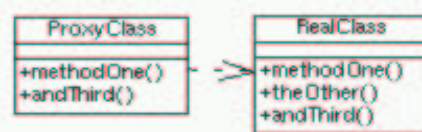


PROXY PATTERN

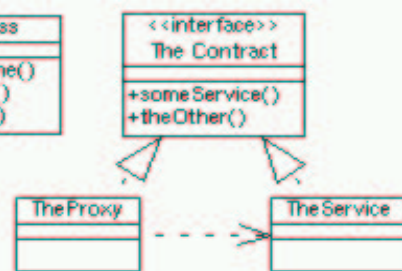
Decouples interface and implementation

- Implementation may change, proxy remains the same
- A common java technology Interface NOT required
- EJBObject is a proxy for EB component
 - Bean instant can be changed at run time
 - Container has freedom to manage life cycle
 - Bean instant can be deployed to desired container

Example 1:

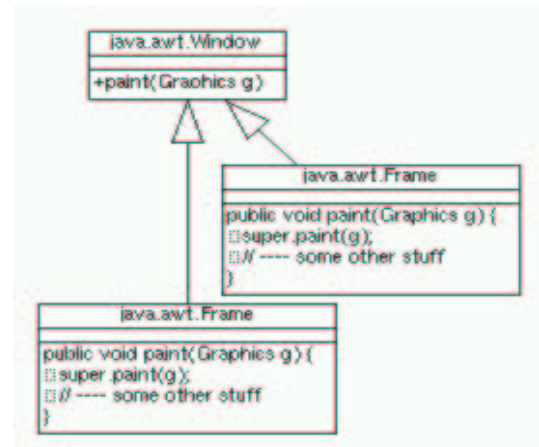
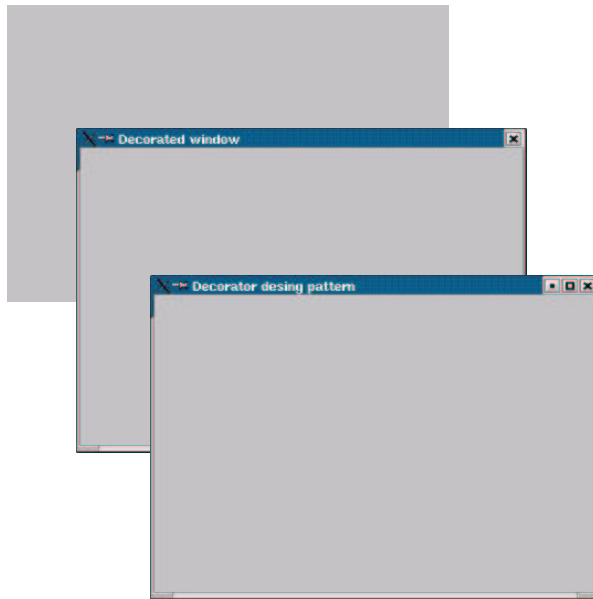


Example 2:



DECORATOR PATTERN

The same interface with extended functionality



J2EE APPLICATION ARCHITECTURES V. 1.1 © RESOLUTION OY, MAY 2002

```
import java.awt.*;

public class Decorator {

    private Window[] wins;

    public static void main(String[] args) {
        Decorator jimi = new Decorator();
        jimi.isAlive();
    }

    public void isAlive() {

        // ----- create the windows
        wins = new Window[3];
        wins[0] = new Frame("Decorator desing pattern");
        wins[1] = new Window((Frame)wins[0]);
        wins[2] = new Dialog((Frame)wins[0], "Decorated window", false);

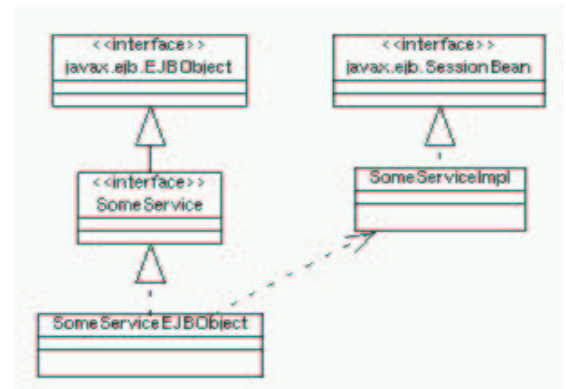
        // ----- notice the same interfaces
        for (int i = 0; i < wins.length; i++) {
            showMe(wins[i], i);
        }
    }

    public void showMe(Window w, int loc) {
        w.setSize(400, 300);
        w.setLocation(40 * loc, 40 * loc);
        w.setVisible(true);
    }
}
```

DECORATOR PATTERN (CONT.)

EJBOject is a decorator for the EB component:

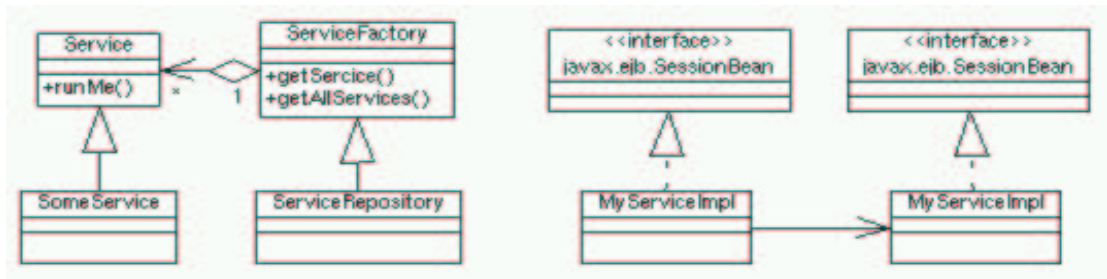
- Remote invocation services added
- Concurrency control added
- Bean developer can concentrate on business logic



FACTORY METHOD PATTERN

Definition of an interface for object creation without the actual implementation.

- Improves flexibility by providing "a hook" for location independent service repositories
- EJBHome interface acts as a factory enabling EB component creation



ABSTRACT FACTORY PATTERNS

Interface for creating a family of related objects without specifying the actual implementation

- Reinforces pluggable behavior and dynamic content selection
- Used by EJBHome to control EB component creation

