

Architect Enterprise Applications with Java EE

Volume 2 – Student Guide

D68136GC10
Edition 1.0
July 2012
D71726

ORACLE®

Author

Joe Greenwald

**Technical Contributors
and Reviewers**

Linda deMichiel

Ian Evans

Roberto Chinnici

William Markos

Shreechar Ganapaphy

Vishal Mehray

Publisher

Jayanthy Keshavamurthy

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

1 Introducing Enterprise Architecture

- Lesson Objectives 1-2
- Lesson Agenda 1-3
- Course Objectives 1-4
- Target Audience 1-6
- Introductions 1-7
- Course Agenda 1-8
- Classroom Guidelines 1-10
- Appendices Used in the Course 1-11
- For More Information 1-12
- Lesson Agenda 1-13
- Discussion Questions 1-14
- Challenges of Enterprise Applications 1-15
- What Is Software Architecture? 1-16
- Justifying the Need for Software Architecture 1-18
- Client-Server System 1-20
- Highly Distributed System 1-21
- Quality of Service 1-22
- Risk Evaluation and Control 1-24
- The Goal of Architecture 1-26
- What Is Enterprise Architecture? 1-27
- Enterprise Architects and Enterprise Application Architects 1-28
- Lesson Agenda 1-30
- The Architect Role 1-31
- The Architect's Involvement 1-32
- Responsibilities of an Architect 1-33
- Architect and Other Team Members 1-36
- Analyze – Evaluate – Prescribe 1-38
- Influencing Factors 1-39
- What Do You Think? 1-40
- Additional Resources 1-43
- Quiz 1-44
- Summary 1-47
- Practice 1 Overview: Review Questions: Introducing Enterprise Architecture 1-48

2 Introducing Fundamental Architectural Concepts

Objectives 2-2

Discussion Questions 2-3

Lesson Agenda 2-4

Which Is More Important, Architecture or Design? 2-5

Distinguishing Between Architecture and Design 2-6

Common Principles Between Architecture and Design 2-7

Architectural Principles 2-9

Lesson Agenda 2-11

Architectural Patterns and Design Patterns 2-12

Architectural Patterns 2-14

The Layers Pattern 2-15

The MVC Pattern 2-16

The Tiers Pattern 2-17

Model 1 Architecture 2-19

Model 2 Architecture 2-20

PAC Architectural Pattern 2-21

Lesson Agenda 2-22

Typical Software Deliverable Artifacts 2-23

Architectural Blueprint 2-25

The 4+1 View Model 2-26

The Architectural Prototype 2-28

Lesson Agenda 2-29

Architecture Modeling Using UML 2-30

Class Diagrams 2-31

Interaction Diagrams 2-32

Communication Diagrams 2-33

Sequence Diagrams 2-34

Component Diagrams 2-35

Types of Components 2-36

Deployment Diagrams 2-38

Types of Deployment Diagrams 2-40

Package Diagrams 2-41

Lesson Agenda 2-42

Architecture Workflow Steps 2-43

Select the Architecture Type 2-45

Type of Software Architectures 2-46

Standalone Applications 2-47

Client/Server (2-Tier) Applications 2-48

Application-centric N-Tier Applications 2-49

Web-centric N-Tier Applications	2-50
Enterprise-centric N-Tier Applications	2-51
Creating the Detailed Deployment Diagram	2-52
Lesson Agenda	2-53
What Is an Enterprise Architecture Framework?	2-54
Popular Enterprise Architecture Frameworks	2-55
Comparison of EA Frameworks	2-56
Oracle Enterprise Architecture Framework	2-57
The Oracle Architectural Development Process	2-58
TOGAF to OEA Mapping	2-59
OADP to TOGAF ADM Mapping	2-60
Additional Resources	2-61
Quiz	2-62
Summary	2-67
Practice 2 Overview: Model Initial Architecture	2-68

3 Developing a Security Architecture

Objectives	3-2
Discussion Questions	3-3
Lesson Agenda	3-4
Types of Access Control	3-5
Role-based Access Control (RBAC)	3-7
Invasions	3-9
Regulatory Constraints	3-11
Lesson Agenda	3-12
Impact of Security	3-13
Securing the Network Services	3-14
Securing the Hosting Environment	3-15
Securing Applications	3-16
Common Security Principles	3-17
Maintaining Corporate Security Policies	3-18
Self-Preservation	3-20
Defense in Depth	3-21
Least Privilege	3-22
Compartmentalization	3-23
Proportionality	3-24
Costs and Benefits of Security Features	3-25
Resolving Trade-Off Items in Security	3-26
How Do You Know Your Users?	3-27
Lesson Agenda	3-28
Examining Security in the Java EE Technology	3-29

Defining Java EE Security Terminology	3-30
Authentication	3-31
Defining Protection Domains	3-32
Authorization	3-33
Data Confidentiality and Integrity	3-34
Selecting Cipher Suites	3-35
Declarative and Programmatic Security	3-37
Declarative Security	3-38
Programmatic Security	3-39
Comparing Declarative Security and Programmatic Security	3-40
Lesson Agenda	3-42
Understanding Web Services Security	3-43
XML Signature	3-45
XML Encryption	3-48
Web Service Security (WS-Security)	3-50
Addressing Web Services Security Requirements	3-51
Message Structure in WS-Security	3-52
Web Services Security and Java EE	3-54
Web Service Security in Oracle WebLogic	3-55
Web Service Interoperability Technology (WSIT)	3-56
Security Assertion Markup Language	3-57
SAML Terminology	3-59
SAML WS-Security Architecture	3-60
SAML WS-Security Profiles	3-61
Additional Resources	3-62
Quiz	3-63
Summary	3-68
Practice 3 Overview: Identifying Security Risks	3-69

4 Understanding Nonfunctional Requirements

Objectives	4-2
Discussion Questions	4-3
Lesson Agenda	4-4
Functional and Nonfunctional Requirements	4-5
Nonfunctional Requirements	4-6
Nonfunctional Requirements Categories	4-7
Impact of Dimensions on Nonfunctional Requirements	4-11
Making Trade-Off Decisions on System Dimensions	4-15
Capture and Examine NFRs	4-16
Lesson Agenda	4-17
Introducing Redundancy to the System Architecture	4-18

Load Balancing	4-19
Failover	4-21
Effects of Failover	4-22
Clusters	4-23
Common Cluster Configurations	4-24
Two Node Cluster: Asymmetric	4-25
Two Node Cluster: Symmetric	4-26
Clustered Pairs Topology	4-27
N+1 Topology	4-28
N*N Topology	4-29
Evaluating Replication Strategies	4-30
Machine Equivalence	4-31
Network Design	4-32
Cost Consideration	4-33
Improving Performance and Throughput	4-34
Improving Availability	4-35
Improving Extensibility and Flexibility	4-36
Improving Scalability	4-37
Virtualization	4-38
JVMs	4-41
Packaging and Deployment Considerations	4-43
Testing	4-44
Lesson Agenda	4-46
Prioritizing Quality-of-Service (QoS) Requirements	4-47
System Design Considerations	4-49
Ranking QoS Requirements	4-50
Prioritization of QoS Requirements	4-52
Reviewing Quality Estimation	4-53
Revising QoS Values	4-54
Lesson Agenda	4-55
Inspecting QoS Requirements for Trade-Off Opportunities	4-56
Additional Resources	4-58
Quiz	4-59
Summary	4-64
Practice 4 Overview: Consider the impact of Nonfunctional Requirements	4-65

5 Defining Common Problems and Solutions

Objectives	5-2
Discussion Questions	5-3
Lesson Agenda	5-4
Identifying Key Risk Factors	5-5

What Do You Think?	5-6
System Flexibility	5-7
Network Communication and Layout	5-8
Transaction Model	5-9
Security Model	5-10
System Sizing and Planning	5-11
Estimation Best Practices	5-12
Lesson Agenda	5-13
Designing a Flexible Object Model	5-14
Using Abstractions	5-15
Applying Object-Oriented Principles	5-16
Open-Closed Principle	5-17
Dependency Inversion Principle	5-19
Interface Segregation Principle	5-20
Composite Reuse Principle and Separation of Concerns Principle	5-21
Common Closure Principle and Common Reuse Principle	5-22
Applying Patterns	5-23
Common Pattern Catalogs	5-24
Gang of Four (GoF) Design Patterns	5-25
Java EE Patterns	5-26
Architecture Patterns by Buschmann, et al.	5-29
The Layers Pattern	5-30
Enterprise Integration Patterns	5-31
Enterprise Application Architecture Patterns	5-32
Pattern Integration	5-33
Using Patterns Across a Network	5-34
Object-Based Patterns	5-36
Using Reliable Frameworks	5-37
Open Source Issues	5-38
Service-Based Architecture	5-39
Developing Service-Based Architectures	5-40
Types of Services	5-42
Granularity of Services	5-43
Versioning Services	5-45
What Do You Think?	5-47
Additional Resources	5-48
Quiz	5-49
Summary	5-53
Practice 5 Overview: Create a Flexible Object Model	5-54

6 Defining Common Problems and Solutions

- Objectives 6-2
- Discussion Questions 6-3
- Lesson Agenda 6-4
- Network Communication Guidelines 6-5
- Network Performance Guidelines 6-6
- Distributed Computing Fallacies 6-7
- What Do You Think? 6-8
- Creating a Network Model 6-9
- Estimating Network Latency 6-10
- Constructing Efficient Data Models 6-12
- Increased Operation Granularity 6-13
- Compression 6-14
- Lesson Agenda 6-15
- Justifying the Use of Transactions 6-16
- When Are Transactions Required 6-17
- CAP Conjecture 6-19
- ACID versus BASE 6-20
- Difficulties of Transaction Models 6-21
- Types of Transactions 6-22
- Impact of Transactions on Latency and Request Frequency? 6-23
- Write-Write Conflict 6-25
- Deadlock 6-26
- What Do You Think 6-27
- Isolation Levels 6-28
- Transaction (Txn) Isolation Phenomena 6-29
- Transactions Already Committed 6-31
- Creating the Transaction Model 6-32
- Analyzing Application Transaction Requirements 6-34
- What Do You Think? 6-36
- Creating an Application's Transaction Model 6-37
- Transaction Performance Metrics 6-39
- Lesson Agenda 6-41
- Planning System Capacity 6-42
- Two Extremes for System Capacity 6-43
- System Load Characteristics 6-44
- Estimating Transaction Load 6-45
- Estimating Transaction Rate 6-46
- Estimating Client Load 6-47
- Sizing the System 6-49
- What Do You Think? 6-51

Planning Scalability	6-52
Machine Performance Profiles	6-53
Cloud Computing	6-54
Additional Resources	6-56
Quiz	6-57
Summary	6-61
Practice 6 Overview: Consider Network, Transaction and Capacity Planning for the Architecture	6-62

7 Java EE 6 Overview

Objectives	7-2
Discussion Questions	7-3
Java EE 6 Goals	7-4
Distributed Multitiered Applications	7-6
Java EE Server Communications	7-8
Web Tier and Java EE Applications	7-9
Business and EIS Tiers	7-10
Java EE Containers	7-11
New Features: EJBs	7-13
New Features: Java Servlet Technology	7-15
New Features: JSF 2.0	7-17
New Features: Java API for RESTful Web Services	7-19
New Features: Managed Beans	7-20
New Features: Context and Dependency Injection	7-21
Overview of CDI	7-23
Beans as Injectable Objects	7-25
New Features: Bean Validation	7-26
New Features: Java Persistence API	7-28
“Classical” J2EE / Java EE 5 Architecture	7-30
Impact of Java EE 6 on Architecture	7-31
Quiz	7-32
Summary	7-35
Practice 7 Overview	7-36

8 Developing an Architecture for the Client Tier

Objectives	8-2
Discussion Questions	8-3
Lesson Agenda	8-4
Overview of the Client Tier	8-5
Client Tier Roles	8-6
Lesson Agenda	8-8

Information Architecture Client Tier Concerns	8-9
User Analysis	8-10
Usage Analysis	8-11
Robustness Analysis	8-12
User Interface Design Principles	8-13
Two Laws of User Interface Design	8-14
Usability, User Acceptance, and Prototyping	8-15
Data Density	8-16
Accessibility and Section 508	8-17
Internationalization	8-19
Lesson Agenda	8-20
Selecting User Interface Devices	8-21
Resource Limitations	8-22
Robustness of Human Input Devices	8-23
Basic Compared to Complex User Interaction	8-25
Selecting User Interface Technologies	8-26
Desktop Graphical UI Toolkits	8-27
Desktop Graphical User Interface Technologies	8-28
The Abstract Window Toolkit	8-29
The Swing Toolkit	8-30
The Standard Widget Toolkit	8-31
JavaFX	8-32
Desktop Web Browser Technologies	8-33
Browser Compatibility	8-34
HTML and Cascading Style Sheets	8-35
Rich Internet Applications	8-36
Rich Internet Application Frameworks	8-37
RIA Technologies	8-38
Java Applets	8-40
Mobile Graphical User Interface Technologies	8-41
JTWI Architecture	8-43
SMS Asynchronous Messaging	8-45
Java ME Web Service API	8-46
Microbrowsers	8-47
XHTML MP Example	8-48
XHTML Mobile Profile	8-49
MIDlets or Microbrowsers	8-50
Client Side Adaptation	8-51
Multiserving	8-52
Web Browser Technology Best Practices	8-53
Out-of-band JavaScript Callbacks	8-54

Out-of-Band JavaScript Callbacks	8-55
The XMLHttpRequest Messages	8-56
Comet	8-57
WebSockets	8-59
Web 2.0	8-61
Search Engine Optimization	8-63
What Do You Think?	8-65
Lesson Agenda	8-66
Discovering Reusability in the Client Tier	8-67
User Interface Design Patterns	8-68
Third Party Web Service APIs	8-69
Lesson Agenda	8-70
Deployment Strategies for the User Interface	8-71
Using Installation Utilities	8-72
Installation Utilities	8-73
Java Web Start	8-74
Application Client Container	8-75
Lesson Agenda	8-76
Security Concerns in the Client Tier	8-77
Input Data Validation	8-78
Code Injection	8-79
Cross Site Scripting (XSS)	8-80
Lesson Agenda	8-82
Testing	8-83
Additional Resources	8-84
Quiz	8-85
Summary	8-90
Practice 8 Overview: Choosing Client Technologies	8-91

9 Developing an Architecture for the Web Tier

Objectives	9-2
Discussion Questions	9-4
Lesson Agenda	9-5
Responsibilities of the Web Tier	9-6
What Is a Servlet?	9-7
What Is a JavaServer Page?	9-8
What Is JavaServer Faces?	9-9
Oracle ADF Faces	9-10
Java Naming and Directory Interface	9-11
Web Tier Development Roles	9-13
Lesson Agenda	9-14

Separation of Concerns (SoC)	9-15
Presentation Concerns	9-16
Templating the Layout View	9-17
Inclusion Templating	9-18
TransclusionTemplating	9-19
View Compositing	9-20
View Generation Frameworks	9-21
Previewability	9-22
Scripting Languages	9-23
Session State Management	9-24
Input Data Validation	9-25
Internationalization and Localization	9-27
Languages Spoken by Internet Users	9-28
Content Management	9-30
Searchability	9-31
Control and Logic Concerns	9-32
Model View Controller	9-33
Service-to-Worker Pattern	9-34
Filtering	9-36
Web Flow	9-37
Lesson Agenda	9-38
Popular Web Tier Frameworks	9-39
Criteria for Framework Selection	9-40
Request-Oriented Frameworks	9-41
Component Oriented Frameworks	9-42
Oracle Application Development Framework (ADF)	9-43
Technology Choices for ADF BC Applications	9-44
Competing Frameworks	9-45
Lesson Agenda	9-47
Providing Security in the Web Tier	9-48
JAAS	9-50
Java Authorization Contract for Containers	9-51
Oracle Identity Management Oracle + Sun Combination	9-52
Oracle Access Management Suite Plus	9-55
Salient Features	9-57
OAM 11g Architecture	9-59
Lesson Agenda	9-60
Web Server Clustering	9-61
Oracle Coherence	9-62
Singletons in the Web Tier	9-63
Accommodating Load Spikes	9-64

Session Persistence	9-65
Load Balancing	9-66
Single System Images	9-67
Additional Resources	9-68
Quiz	9-69
Summary	9-73
Practice 9 Overview: Architecting for the Web Tier	9-75

10 Developing an Architecture for the Business Tier

Objectives	10-2
Lesson Agenda	10-3
What is an Enterprise Bean?	10-4
Accessing Enterprise Beans	10-5
Deciding on Remote or Local Access	10-7
What Is a Session Bean?	10-9
Stateless Versus Stateful Session Beans	10-10
Singleton Session Beans	10-12
Message-Driven Beans	10-14
Timer Service	10-15
What Is JAX-WS?	10-16
Representational State Transfer (REST)	10-17
Enterprise Application Container Services	10-18
Lesson Agenda	10-21
Architecting Domain Model Services	10-22
JMS Asynchronous Communication	10-24
Message Driven Beans	10-26
JNDI Naming Server	10-28
Rules Engines	10-29
Basic Oracle Business Rule Concepts	10-31
Oracle Business Rule Components	10-32
Workflow Engines	10-33
Architecting Domain Model Entities	10-34
Java EE 6 Business State	10-35
Mapping Domain Value Objects to XML	10-36
Distributing Domain Model Components	10-37
The Component Distribution Golden Hammer	10-38
Creating Coarse Grained Façades	10-39
Lesson Agenda	10-40
Development Best Practices	10-41
Exception Handling	10-42
Using Runtime Exceptions	10-44

Logging	10-45
Additional Resources	10-47
Quiz	10-48
Summary	10-53
Practice 10 Overview: Refining the Business Architecture	10-54

11 Developing an Architecture for the Integration and Resource Tiers

Objectives	11-2
Discussion Questions	11-3
Lesson Agenda	11-4
Challenges of Integration	11-5
The Integration Tier	11-7
The EIS Resource Tier	11-9
Relational Databases	11-11
Nonrelational Data Sources	11-12
Alternatives to Relational Databases for High Performance	11-13
Operational Resources	11-15
Resource Servers	11-16
LDAP Servers	11-17
Security Servers	11-18
Extract, Transform and Load (ETL) Tools	11-19
Data Mining	11-20
Enterprise Data Model	11-21
What Do You Think?	11-22
Lesson Agenda	11-23
Java Integration Technologies	11-24
JDBC and Object Relational Mapping (ORM)	11-25
ORM Frameworks	11-27
What Do You Think?	11-29
Overview of Java Persistence API (JPA)	11-30
What Are JPA Entities?	11-31
Messaging Systems	11-32
Message-Oriented Middleware	11-33
Java Message Service	11-34
JMS Application Architecture	11-35
Point-to-Point Queue	11-36
Publish-Subscribe Topics	11-37
WebLogic Server JMS Features	11-38
JMS Architecture: Connecting	11-39
JMS Architecture: Sending Messages	11-40
Transacted Messaging	11-41

WebLogic Server JMS Server	11-42
Connection Factory	11-43
JMS Destination	11-44
Message Driven Beans (MDB)	11-45
Java EE Connector Architecture (JCA)	11-46
Integration Technologies	11-48
Web Services	11-49
Java Web Service Technologies	11-51
Java API for XML-based Web Services (JAX-WS)	11-52
Comparing Integration Technologies	11-53
Comparing JMS, JCA, and Web Services	11-54
Lesson Agenda	11-56
Applying Integration Tier Patterns	11-57
Integration Tier Patterns	11-58
Aspect Oriented Programming	11-59
XML Services	11-60
Lesson Agenda	11-62
Service-Oriented Architecture (SOA)	11-63
Why Is an SOA Approach Required?	11-64
Ways to Implement Services	11-65
What Are Services?	11-66
Adopting Standards for an SOA Approach	11-67
Standards That Enable SOA	11-68
Designing with an SOA Approach	11-69
Creating Service Portfolios	11-71
SOA, Web Services and Java EE	11-72
Business Process Execution Language (BPEL)	11-73
Introducing Business Process Execution Language (BPEL)	11-74
Service Component Architecture (SCA) and Service Data Objects (SDO)	11-76
ESB Features and Functions	11-78
Canonical Model	11-80
SOA Best Practices	11-81
Moving to SOA	11-83
SOA Governance	11-84
Define SOA Governance	11-85
Identifying the Need of SOA Governance	11-86
What Do You Think	11-88
Additional Resources	11-89
Quiz	11-90
Summary	11-94
Practice 11 Overview: Refining the Integration Tier Architecture	11-95

12 Evaluating the Software Architecture

- Objectives 12-2
- Discussion Questions 12-3
- Lesson Agenda 12-4
- Evaluating Software Architectures 12-5
- Characteristics of a Good Architecture 12-6
- Architecture Evaluation Guidelines 12-8
- Lesson Agenda 12-9
- Java EE Technologies and Architectural Objectives 12-10
- Designing for Long-Term Application State 12-11
- Managing Client Session State 12-14
- Enabling Business Process and Workflow Control 12-17
- Enabling Presentation Process and Workflow Control 12-19
- Managing Presentation Layout 12-21
- Designing for Asynchronous Communication 12-23
- Java EE Technologies and Architectural Objectives 12-24
- Lesson Agenda 12-25
- Creating System Prototypes 12-26
- Prototypes Based on Patterns 12-27
- Prototype Validation for Standards Conformance 12-28
- Prototype Testing 12-29
- Measuring QoS Capabilities of System Prototypes 12-30
- Saturation Points 12-31
- Judging the Prototypes Against Architectural Goals 12-33
- Extrapolations or Trend Curves 12-34
- What Do You Think? 12-35
- Lesson Agenda 12-36
- Defining Application Server Selection Criteria 12-37
- Evaluating Application Server Selection Criteria 12-39
- Selecting Frameworks & Libraries 12-41
- Additional Resources 12-43
- Quiz 12-44
- Summary 12-48
- Practice 12 Overview 12-49

Appendix A: UML 2: Quick Reference

Appendix B: Acronyms

Appendix C: Glossary

Appendix D: UMLet Tips

Iveth Tello (iveth.tello@eppetroecuador.ec) has a non-transferable
license to use this Student Guide.

Developing an Architecture for the Client Tier

8

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the roles involved in client tier development
- Define Information Architecture client tier concerns
- Describe how to select a user interface device that will fit your application requirements
- Describe how reuse can apply to the client tier
- Describe strategies for deploying Java desktop-based applications
- Be familiar with the security concerns of the client tier



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Discussion Questions

- What kind of applications do end users use with their cell phones and PDAs?
- How does the level of interactivity affect the usability of cell phone or PDA-based GUI clients?
- What sorts of features qualify a client as being “rich”? What sorts of applications call for the features of a Rich Internet Application?
- How does the desire for a productive online marketing campaign affect the architecture of a browser-based client?
- How does the availability of toolkits for out-of-band web service callbacks in JavaScript language affect the design of browser-based clients?

 ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Client tier development roles
- Information Architecture Client tier concerns
- Selecting user interface devices and technologies
- Discovering reusability in the Client tier
- Deployment strategies for the user interface
- Security concerns in the Client tier
- Testing

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Overview of the Client Tier

- The client tier is unlike the other tiers of the enterprise architecture.
- The other tiers tend to focus on aspects of a software architecture that directly, or indirectly, support the implementation of a business process.
- The client tier, however, must give consideration to the nature of the technology hosting the user interface.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The architectural decisions, given the ubiquity of the Java platform, can be considered with limited regard to the enterprise class technology hosting the application components.

The goal of a well architected user interface is to provide the best possible user experience for interacting with the application, and the user experience depends upon the capabilities of the platform technology the end user is using to access the application interface.

Client Tier Roles

- **Graphic Artist:** Uses a combination of raster, vector, and/or 3D modeling software tools to generate the interface artwork.
- **Interface Designer:** Provides the view component layout design and implementation.
- **Information Architect:** Creates the organization, structure, user interaction, and overall layout of a user interface.

 ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The development roles for the client tier can be broken into three areas of concern. These areas of concern encapsulate the creation of visually appealing elements, the proper placement of these elements on an interface, and the manner in which users interact with an interface. These concerns directly map to the job descriptions that would identify a graphic artist, an interface designer, and an information architect, respectively. Each role involves specialized knowledge and skill sets that can be difficult for a single person to fulfill.

A graphic artist is responsible for creating the visual elements to be used in the interface. They might use any combination of raster, vector, or 3D modeling software tools to generate the artwork to be incorporated into the interface. The visual elements that a graphic artist might create include:

- Backgrounds
- Buttons, tabs, and icons
- Borders
- Decorative labels
- Logos
- Advertisement banners
- Look and feel and “branding” elements

The interface designer is an expert at the chosen interface technology. This job provides the view component layout design and implementation under the guidance of the information architect. The interface designer establishes the look-n-feel of the interface and often works closely with the graphic artist to incorporate the artwork elements. An interface designer might be particularly adept at:

- Browser markup languages
- Java graphical user interface (GUI) toolkit APIs
- Mobile device markup languages

An information architect is concerned with the organization, structure, user interaction, and overall layout of a user interface to create the greatest degree of user acceptance. The interface architect specializes in User Centered Design (UCD) and can identify the patterns relevant to interaction design, usability, findability, and general information management strategies. A qualified information architect should lead the user acceptance testing of prototypes and interface production releases.

Lesson Agenda

- Client tier development roles
- **Information Architecture Client tier concerns**
- Selecting user interface devices and technologies
- Discovering reusability in the Client tier
- Deployment strategies for the user interface
- Security concerns in the Client tier
- Testing



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Information Architecture Client Tier Concerns

- The information architect is concerned with creating a usable interface that the end users will find acceptable.
- The following topics describe the concerns that help the information architect achieve this objective:
 - User Analysis
 - Usage Analysis
 - User Interface Design
 - Usability, User Acceptance, and Prototyping
 - Data Density
 - Accessibility



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

User Analysis

- As part of a user-centered design strategy, you should understand the defining characteristics of the potential users of an application and document these characteristics.
- Such an analysis will have a significant impact on interface design in an attempt to achieve a high degree of user acceptance.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Employees and more educated users might appreciate a more data dense interface with less artistic appeal than you would typically provide in a customer-facing interface targeting the general public.

For example, if you found that a vast majority of your audience were computer gamers, that could open up the possibility of user acceptance of an application interface in an online virtual reality environment. However, the expectation of user acceptance of that same interface to a demographic primarily composed of soccer moms would probably not be high.

Usage Analysis

- How a user is going to use an application interface, and what the functional requirements are, also have an impact on the design and selection of devices and technologies for such an interface.
- Starts with a set of use cases and a requirements list.
- Should address how the user will fulfill the functional requirements with a user interface.
- Should identify the view components that will fulfill those functions.
- Robustness analysis boundary components should help identify the view components.

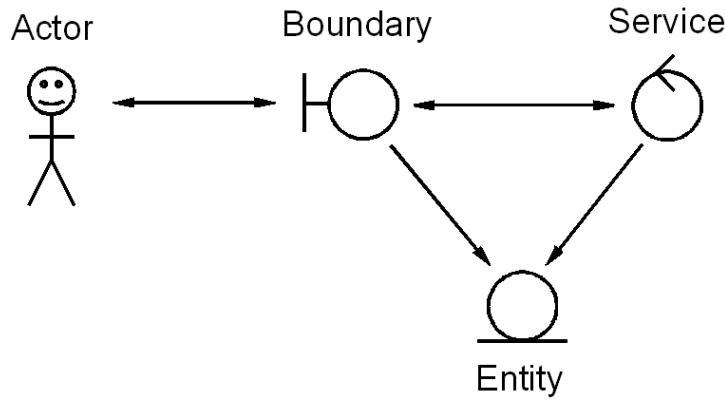
ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

How a user is going to use an application interface, and what the functional requirements are, also have an impact on the design and selection of devices and technologies for such an interface.

The use cases and requirements list that comes out of a architecture methodology approach to application architecture during the Inception Phase should form the basis for a usage analysis. These artifacts should tell you what the user needs to do. However, they do not tell you how the user is to accomplish these use cases using the interface.

Robustness Analysis



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

To provide some direction with the interface definition, and also the domain model, you can use a Robustness Analysis to take a preliminary pass at a set of interfaces, entities, and services that could implement the use cases.

A Robustness Analysis model applies the Separation of Concerns (SoC) paradigm to your use cases. Each use case should yield a set of boundary (presentation), entity (data), and control/service (logic) components.

The Robustness Analysis modeling process is beyond the scope of this class. Students interested in the details of this process are encouraged to take the Object Oriented Analysis and Design class.

Refining your Robustness Analysis models involves classifying, grouping, consolidating, and refactoring. What you will find is that some boundary components, as well as entity and control components, are shared between use cases. As it applies to information architecture and interface design, it is the boundary components that are of the greatest interest. They show you how the user's requests need to be fulfilled, and give an indication for the user interface components that need to be developed.

User Interface Design Principles

Principle	Description
Visibility	Clarity of the visual goal
Feedback	User sent information following their action
Affordance	Element usage obvious to user
Simplicity	Keep it simple
Structure	Visual elements laid out in a meaningful way
Consistency	Visual patterns and navigation make usage easy
Tolerance	Interface is forgiving of user errors



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Any interface should follow a generally accepted set of design principles.

Two Laws of User Interface Design

- **First Law:** A computer shall not harm your work, or, through inactivity, allow your work to come to harm.
- **Second Law:** A computer shall not waste your time or require you to do more work than is strictly necessary.
- **Plus:**
 - Do not make the user remember anything.
 - Do not make the user type any more than they have to.
 - Do not let the user make a mistake.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

As a bit of a spin on Isaac Asimov's Two Laws of Robotics, from his science fiction books, the author of *The Humane Interface: New Directions for Designing Interactive Systems*, JefRaskin, gives us the Two Laws of User Interface Design.

JefRaskin, *The Humane Interface: New Directions for Designing Interactive Systems*. Addison-Wesley Professional; 1st edition (March 29, 2000)

- **First Law:** A computer shall not harm your work, or, through inactivity, allow your work to come to harm.
- **Second Law:** A computer shall not waste your time or require you to do more work than is strictly necessary.

As you can see, the driving principle here is that good user interface design affords a high level of productivity for the task at hand. However, while you might have created an interface that increases the productivity for a given task, you will not have user acceptance until you abide by the core principles for good interface design

Usability, User Acceptance, and Prototyping

- Skilled experts in Human-Computer Interaction (HCI) can have a high degree of success reusing the design methodologies and patterns that have been shown to work in the past.
- However, interface designing is an extremely creative, and thus subjective, process.
- While methodologies and process patterns can go a long way toward creating good design, there is something to be said for the intangible feel of an interface that comes from experience in the interface design industry.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Usability Engineering (UE) extends the subjective notion of design principles, with a pragmatic approach. UE focuses on an iterative metric gathering process to determine the level of user acceptance of a given interface.

UE encourages regular interface prototyping, beginning at the early stages of the development cycle. Such prototypes are often simple mock-ups with no real back-end functionality. However, they go a long way towards gauging user acceptance and the ease of use for the end user.

The process of presenting prototypes, or the real interface, to end users for feedback is called usability testing. It is less formal than a full Quality Assurance (QA) or System Test process, and is simply meant to get a feel for user acceptance. Information architects and interface designers should look for opportunities to improve the interface design during such sessions.

It is not uncommon for usability testing of prototypes early in the development cycle to have profound effects on the application requirements at a broader level. Quite often, end users, and even Subject Matter Experts (SMEs), cannot fully quantify and qualify the behavior and features of the application they want until they begin usability testing. This is true of interfaces without well-known archetypes. All the key players of the development process, including architects, designers, business analysts, and project managers, should attend the initial usability tests. There is value in catching significant functional changes early in the development process.

Data Density

- Measure of the amount of information on an interface.
- Number of different contextual elements represented on a single screen.
- May drive output device selection (screen size).



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Certainly, the impact of data density is obvious with respect to the size of the screen used by the end user. A smaller mobile device screen cannot contain nearly as much information as a large desktop monitor.

The term data density is often used as a mechanism to describe the number of elements on an interface, irrespective of the actual size of the display screen. Therefore, *density in this context should not imply a certain number of screen elements per a given amount of screen real estate*.

Information architects are often conflicted with issues of data density and progressive disclosure. Overly dense interfaces can be intimidating and unattractive. However, users can also be frustrated by having to perform too many interactions to get to their desired point in the interface. The proper balance is a discovery process tied to the usability testing. However, preliminary density and disclosure design factors can come from the user analysis.

Accessibility and Section 508

- Accessibility issues are mostly applicable to web interface development.
- Many countries around the globe have laws and policies relating to the ability of publicly accessible interfaces, including websites, to be usable by people with visual, auditory, physical, speech, cognitive, or neurological disabilities.
- The Web Accessibility Initiative (WAI) at the World Wide Web Consortium (W3C) maintains a publication called the Web Content Accessibility Guidelines (WCAG).
 - The WCAG provides explanations, strategies, and practical markup examples for the effective use of HTML and JavaScript.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Associated with this initiative are the following tips:

W3C WAI, Quick Tips to Make Accessible Web Sites [<http://www.w3.org/WAI/quicktips/>] accessed MAR 2007

Images and animations: Use the alt attribute to describe the function of each visual.

Image maps: Use the client-side map and text for hotspots.

Multimedia: Provide captioning and transcripts of audio, and descriptions of video.

Hypertext links: Use text that makes sense when read out of context. For example, avoid "click here."

Page organization: Use headings, lists, and consistent structure. Use Cascading Style Sheets (CSS) for layout and style where possible.

Graphs and charts: Summarize or use the longdesc attribute.

Scripts, applets, and plug-ins: Provide alternative content in case active features are inaccessible or unsupported.

Frames: Use the noframes element and meaningful titles.

Tables: Make line-by-line reading sensible. Summarize.

Validate: Use tools, checklist, and guidelines at <http://www.w3.org/TR/WCAG>

- W3C WAI, Quick Tips to Make Accessible Web Sites [<http://www.w3.org/WAI/quicktips/>] accessed MAR 2007
- What is Section 508?
- “In 1998, Congress amended the Rehabilitation Act of 1973 to require Federal agencies to make their electronic and information technology (EIT) accessible to people with disabilities. Inaccessible technology interferes with an ability to obtain and use information quickly and easily. Section 508 was enacted to eliminate barriers in information technology, open new opportunities for people with disabilities, and encourage development of technologies that will help achieve these goals. The law applies to all Federal agencies when they develop, procure, maintain, or use electronic and information technology. Under Section 508 (29 U.S.C. ‘794 d), agencies must give disabled employees and members of the public access to information that is comparable to access available to others.” Source: Section508.gov

Internationalization

- Internationalization is the support for multiple locales.
- Localization is the process in which support for a specific locale is added.
- Oracle ADF Faces components provide automatic translation (into 28 languages).



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Internationalization is the process of designing software so that it can be adapted (localized) to various languages and regions easily, cost-effectively, and in particular without engineering changes to the software. This generally involves isolating the parts of a program that are dependent on language and culture. For example, the text of error messages must be kept separate from program source code because they must be translated during localization.

What is localization?

Localization is the process of adapting a program for use in a specific locale. A locale is a geographic or political region that shares the same language and customs. Localization includes the translation of text such as user interface labels, error messages, and online help. It also includes the culture-specific formatting of data items such as monetary values, times, dates, and numbers.

Lesson Agenda

- Client tier development roles
- Information Architecture Client tier concerns
- **Selecting user interface devices and technologies**
- Discovering reusability in the Client tier
- Deployment strategies for the user interface
- Security concerns in the Client tier
- Testing

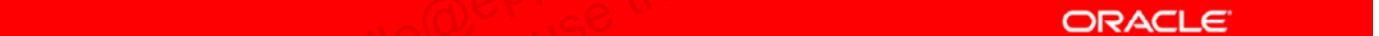


ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Selecting User Interface Devices

- When it comes to the software architecture of the client tier, consideration for the end user hardware resources is unavoidable.
- What you would like the user to use, or what the user is likely to use, for input and output of information from your application tends to drive many factors relating to information architecture and interface development.



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Resource Limitations

Such questions as

- How resource-intensive is the client application expected to be?
- Will it be CPU or memory intensive?

might immediately rule out the possibility of using smaller devices, such as mobile phones, as platforms for hosting client interfaces.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Interaction Requirements



Device Input Robustness

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Human Computer Interaction studies are quite extensive with regard to the mechanisms that humans use to interact with computer systems. In general, the smaller the device, the more limited are the possibilities for complex user interactions. Larger devices, with extensive resources available, might have the ability to accommodate advanced user input mechanisms.

The slide shows how the functional requirements for complex user interactions should be directly proportional to the robustness of an interface device's input capabilities.

- Robustness of Human Input Devices

A device with shallow input robustness should not be considered a potential platform candidate for the hosting of an application with functional requirements that suggest the need for complex user interactions.

PC: A Personal Computer includes desktops, laptops, tablets and some handhelds (also called a palmtop). iPads are beginning to blur the line between laptops and handhelds. The Apple Newton was one of the first devices to venture into this space. While it was not successful at the time, it has influenced the devices that came after.

Smart Terminal: A terminal has a CPU and therefore has some processing capabilities, separate from the host computer. Smart terminals have built-in logic for carrying out simple display operations, such as blinking and boldface. In contrast, a dumb terminal is simply a display device with no processing capabilities.

Thin Client: The obvious example of a thin client is a web browser. Thin clients rely in another machine, the host, to perform computation and generate the contents of the interface. JavaScript blurs this line as more processing is performed on the client. Since the application is not downloaded to the client machine in the traditionally accepted sense of the word, it is still considered a thin client. Client applications written using Swing are considered fat or thick clients. Java Web Start blurs this line in that it enables thick clients to be accessed and downloaded automatically from the Web server. Updates are pushed down automatically as well, relieving some of the traditional burden of thick client distribution.

Basic Compared to Complex User Interaction

BASIC	COMPLEX
<ul style="list-style-type: none">• streaming audio• browsing images• reading news• watching stock values 	<ul style="list-style-type: none">• insurance underwriting• bookkeeping• 3D modeling• text writing 

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Basic user interactions tend to be noninteractive: reading content, streaming video or audio, viewing static information like stock values or sports scores or browsing for images and files.

Complex user interactions are truly more interactive, can involve simple to exceptionally complex processing, and support sophisticated graphical user interfaces.

Selecting User Interface Technologies

- User interface technologies provide the means for the capture of input and display of visual components on an interface device.
- The architectural appropriateness of the following technologies should be given scrutiny before committing to the development of an interface.



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Desktop Graphical UI Toolkits

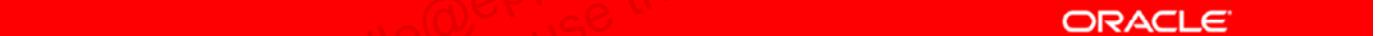
- Abstract Window Toolkit (AWT)
- Swing Toolkit
- Standard Widget Toolkit (SWT)
- JavaFX

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Desktop Graphical User Interface Technologies

- Desktop-oriented Graphical User Interfaces (GUIs) have set the standard for what is referred to as a Rich Client Interface (RCI).
- An RCI assumes the use of a keyboard, mouse, and desktop display for human-computer interaction.
- From a software perspective, it has been traditionally the domain of toolkit APIs with an extensive set of graphical elements for display on the screen called widgets.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The Abstract Window Toolkit

- The Abstract Window Toolkit (AWT) was the first GUI toolkit for Java technology developed by Sun.
- It works with the native libraries of various platforms.
- So, an AWT-based application has the look and feel of the native environment in which it is running.
- However, with respect to other GUI toolkits, AWT has a limited set of widgets with which to work.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

AWT applies the least common denominator (LCD) principle, which means AWT only has the common set of components which exist on all platforms. So you cannot find advanced components such as tables or trees in AWT, because these components are not supported on some other platforms. With regards to feature set per component, AWT applies LCD here, too. It can only support those features available from all platforms. For example, AWT buttons can not be attached with an icon, because on Motif platform, a button is not supposed to have an icon. Because of its poor component and feature support, AWT did not attract many developers. It is deprecated by Sun and is only here to ensure backward compatibility and support Swing.

The Swing Toolkit

- Swing, also developed by Sun, has a more extensive set of widgets, including drag and drop support.
- It uses a Java technology-based window manager, allowing all Swing GUIs to have a consistent look and feel, if desired.
- Recent versions of Swing have few performance-related marks against it.
- Swing-based applications also allow you to have the look and feel of the native platform.

 ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

With respect to component types, Swing applies the greatest common denominator (GCD) principle. Because Swing controls all of the GUI system and is very extensible and flexible, Swing can create almost any component you could imagine. The only limitation of Swing is its AWT containers. In Swing you still can not implement real transparent or irregular-shaped windows, because Swing depends on those AWT top containers including Applet, Window, Frame, and Dialog. Except for these niches, Swing has implemented almost all the standard components on every platform.

The Standard Widget Toolkit

- The open-source Eclipse IDE uses its own GUI toolkit.
- This decision was motivated by the performance problems of Swing, during the early stages, and a desire for users to have a familiar platform specific look and feel.
- Eclipse calls their toolkit the Standard Widget Toolkit (SWT).
- It has a similar set of widgets as Swing, but works with the native platform libraries in a manner similar to AWT.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

SWT implementation also utilize JNI methodology. But the detail is different from that of AWT. SWT evangelists often became furious when they hear people describing SWT as another AWT. In SWT, the only identical part on every platform is the component interface. That is the class and method definition signature. All the underlying code is different from platform to platform. SWT provides an OS class for every platform. This class encapsulates many native APIs by JNI methods. The SWT component class glues these JNI methods together to provide a meaningful functionality. For example, on Windows, text field selection can be conducted by only one system call. This system call is implemented in the Windows OS class as an native method. So there is only one JNI call in the setSelection method of Text on Windows. However, on the Motif platform, text selection involves two native calls. Again, SWT implements these two calls in the Motif OS class. So the component class on Motif needs to call these two calls to achieve text selection.

JavaFX

- JavaFX is a rich client platform for building expressive RIA for multiple screens such as the desktop, browser, mobile and television.
- Whereas, Swing is a set of extensible GUI components that enable developers to more rapidly develop powerful Java front ends for desktop-based commercial applications.
- JavaFX applications that are designed for desktop environments can take advantage of the powerful Swing widget toolkit to build RIA that are optimized for the desktop.

 ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Oracle will introduce a new set of Java APIs that will open JavaFX capabilities to all Java developers, without the need for them to learn a new scripting language. The JavaFX APIs will be a variation on typical JavaBeans properties and listeners, and will be designed to work well with the lambda expressions coming in future releases of Java.

The new Java APIs will:

- Allow the use of powerful Java features such as generics, annotations, and multithreading
- Make it easier for Web developers to use JavaFX from other popular dynamic languages such as JRuby, Groovy, and JavaScript
- Allow Java developers to use other system languages such as Scala, which run on the JVM for writing large or complex JavaFX applications

Desktop Web Browser Technologies

- The most commonly used technology for the display of enterprise application user interfaces is the desktop web browser.
- Inherently, all popular browsers support recent versions of the Hypertext Markup Language (HTML) specification and some sort of scripting language for dynamically manipulating the visual components within a page displayed on a web browser.
- However, supporting multiple user browsers simultaneously tends to be problematic with respect to the scripting languages.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Most browsers have adopted some flavor of the original JavaScript™ browser scripting language developed by Brendan Eich while at Netscape in 1995. The syntax for advanced features, however, tends to vary between browsers.

Consider using JSON (fast) over XML (dense) for architectural reasons.

Browser Compatibility

- Internet Explorer has its own scripting language (that is similar to the JavaScript language) called JScript.
- All the other browsers implement their own flavor of JavaScript technology.
- At a minimum, most JavaScript implementations abide by the European Computer Manufacturer's Association (ECMA) ECMAScript specification, for uniformity.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Unfortunately, the ECMA standards body has not progressively maintained the ECMAScript specification, relative to its usage in browsers. Even the most recent version of ECMAScript, version 4, is considered to have little application to the advanced feature set of most JavaScript implementations that provide interesting visual effects and capabilities.

As any experienced JavaScript programmer will tell you, imagining what your browser application can do, and actually getting it to function across multiple browsers, are two completely different tasks. The differences in how every browser implements JavaScript technology are significant enough that duplicating interesting behavior invariably requires browser-specific code in each browser.

You must decide whether supporting multiple browsers is worth the impact on the development cycle. Many corporate intranet web applications mandate a specific web browser by policy, to cut down on development costs.

Consider the following browser market share as a guide for consideration in the quest to support the appropriate degree of market share for your audience. Many Business-to-Consumer (B2C) Web applications just support Internet Explorer and Firefox. That strategy addresses a vast majority of the potential audience, and as luck would have it, a majority of the JavaScript API for Firefox is supported by Opera and Safari.

HTML and Cascading Style Sheets

- Cascading Style Sheets (CSS) were created as a mechanism to separate the stylistic aspects of how a webpage gets displayed from the HTML source code that contained the content.
- CSS allows you to define those aspects of your web application that create a look and feel into a common location.
- Items, such as fonts, colors, margins, alignments, and general positioning can be more easily addressed in CSS, without the need for HTML tricks.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

HTML development used to involve a great many “tricks” to create interesting stylistic effects. These tricks, such as deeply nested tables for the placement of images and transparent images just 1 pixel in size were commonplace. They also tended to make the HTML source code difficult to maintain. Just finding the actual data content within the HTML could be a challenging task.

Rich Internet Applications

- Have the look and feel of desktop GUI applications
- Makes use of advanced JavaScript features such as:
 - Directory tree expansions
 - Sliders
 - Layering
 - Tabbing
 - Transition effects
 - Drag and drop



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Modern day web browsers can manipulate their browser Document Object Models (DOMs) with their own flavor of JavaScript technology to such an extent that you can have what is referred to as a Rich Internet Application (RIA).

An RIA is a Rich Client Interface (RCI), with the ability to implement visual components, such as directory tree expansions, sliders, layering, fancy visual effects, and drag and drop without having to initiate a page request to the HTTP server.

From a deployment standpoint, the user navigates the browser to the application's website to see the application interface.

Rich Internet Application Frameworks

RIA frameworks ease the pain of development for multiple browsers by providing:

- Browser specific JavaScript source code generation
- Browser specific JavaScript libraries with ECMAScript interfaces
- A combined approach



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

After enough repetition, the average software engineer starts to look for patterns and potential frameworks to shortcut the development process. Fortunately, there are some mechanisms to help in the development of multibrowser JavaScript technology.

In the JavaScript source code generation approach, the interface is constructed using an independent user interface toolkit. The semantics of that user interface are parsed and used as the basis for the source code generation of JavaScript technology that allows the user interface to operate in most user browsers.

The JavaScript library approach is similar to a generic JavaScript API. The framework makes JavaScript functions available for the definition of graphical elements and the registering of event handlers. The developer interaction with the JavaScript library never goes beyond basic ECMAScript. So, you can be sure that your JavaScript code is portable. Inside the JavaScript library are the browser-specific JavaScript widgets.

Some RIA frameworks, like Flex, are designed to run inside the Flash browser plugin. Others, like OpenLaszlo, can be configured to either run inside Flash, or use the JavaScript language.

RIA Technologies

- HTML 5
- Adobe Flash
- Microsoft Silverlight
- Applets
- Java FX
- Adobe Air



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

There are also RIAs that are designed to run inside browser plugins like Flash.

There are no plugins that need to be installed, and no significant security threats. This is one of the most exciting evolutions in client-side application.

- **HTML 5:** HTML5 supports developers to create more exciting, interactive websites and applications. It is seen as an alternative to Adobe Flash, in part thanks to the Apple-Adobe arguments. Apple refuses to support Flash on its devices and this has caused a major ripple in the industry. HTML5 includes new tags and support for drawing, video and sound; geolocation, client-side databases offline application caches, thread-like operations and smarter forms.
- **Adobe Flash:** Adobe Flash (formerly Macromedia Flash) is a multimedia platform that adds animation, video, and interactivity to webpages. Most recently it has come into conflict with HTML5, mostly due to Steve Jobs' stance. Performance tests show Flash to be performant if allowed access to the underlying hardware acceleration. Since Apple does not allow this, performance of Flash is poorer on Apple devices. With the increasing presence of Apple's iOS platform for iPhone and iPad, some have seen this as signaling the end of Flash's dominance. Recent tests have also shown Flash performance on the Android, a strong competitor with the iOS platform, to be significantly slower.

- **Microsoft Silverlight:** It is a web application framework that provides functionalities similar to those in Adobe Flash, integrating multimedia, graphics, animations and interactivity into a single runtime environment. Benefits include the ability to target a single consistent runtime, execute .Net code without deploying the .Net runtime and third-party support is increasing. Issues include no support for Silverlight on iOS, the design tools are nonstandard, no support for popular codecs like h.264, it's a proprietary technology and developers are forced to develop on Windows.
- **JavaFX:** JavaFX is an expressive rich client platform for creating and delivering rich Internet experiences across different screens including mobile devices, desktops, televisions, and other consumer devices. The JavaFX platform contains an essential set of tools and technologies that enable developers and designers to collaborate, create, and deploy applications with expressive content to browsers and desktops. Benefits include: one-stop-shop for expressive content design and development platform for all screen, broad market reach, broad mobile device capabilities, lower implementation costs and a powerful runtime. Issues include maturity, non-Java syntax, fewer design tools, only supports Latin character sets (at this time) and possibly performance.
- **Adobe Integrated Runtime (AIR):** Also known as Adobe AIR, it is a cross-platform runtime environment developed by Adobe Systems for building rich Internet applications using Adobe Flash, Adobe Flex, HTML, or Ajax, that can be deployed as desktop applications. Benefits include fast execution, cross-platform capabilities, easy conversion of Flex or HTML applications, easy installation, support for ActionScript 3.0 and access to a fast, local database. Issues include: limited extensibility, limited database access is limited to SQLite or Web services, proprietary technology, not threading support, challenging development model, security concerns and lack of UI standards.

Java Applets

- Java Applets offer the ability to run client applications within a confined JavaTM Virtual Machine (JVM) Security Manager, often referred to as a sandbox.
- Applets run inside Web browsers and usually have Rich Client Interfaces, because they make use of Java GUI toolkits.
- Those who desire the RCI capabilities of Java Applets, tend to implement RIAs with JavaScript technology instead.
- Provides Rich Client Interfaces to the browser.
- Considered a security threat by many firewalls.

 ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

This capability created a great deal of excitement when the Java platform first came out. However, within most organizations, applets are now generally considered to be both problematic from a support perspective, and a bit of a security threat. The security threat stems from the naivety of the average computer user, and the propensity of that user to unknowingly permit Trojan activity, even when prompted for permission to allow it. For example, a digitally signed Java Applet, or browser plug-in, might ask a user for permission to accept a self-signed certificate. By allowing the certificate, the user might be granting the executable code to have enough permission to format the local hard drive. Given the likelihood of such an occurrence, or one similar, some corporate firewalls block Java Applets from downloading to user's desktop browsers.

Mobile Graphical User Interface Technologies

- The advances in mobile devices have made these devices viable interfaces for some applications that have limited interaction requirements.
- The most viable technologies for the presentation of the view components are Java Platform, Micro Edition (Java ME) MIDlets and microbrowsers, iOS and Android.
- New Devices:
 - Smartphones: iPhone, Android, and Palm (runs Java)
 - iPad



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A smartphone is the name given to the class of mobile phones that offer more advanced computing ability and connectivity than a contemporary basic “feature phone.” While both smartphones and features phones can be thought of as handheld computers, feature phones run applications based on platforms like Java ME or BREW. Smartphones enable the user to install and run more advanced applications based on a specific, different platform. Also smartphones run complete operating systems (albeit modified or scaled down versions of desktop OSs in some cases) thus providing a rich development environment for application developers.

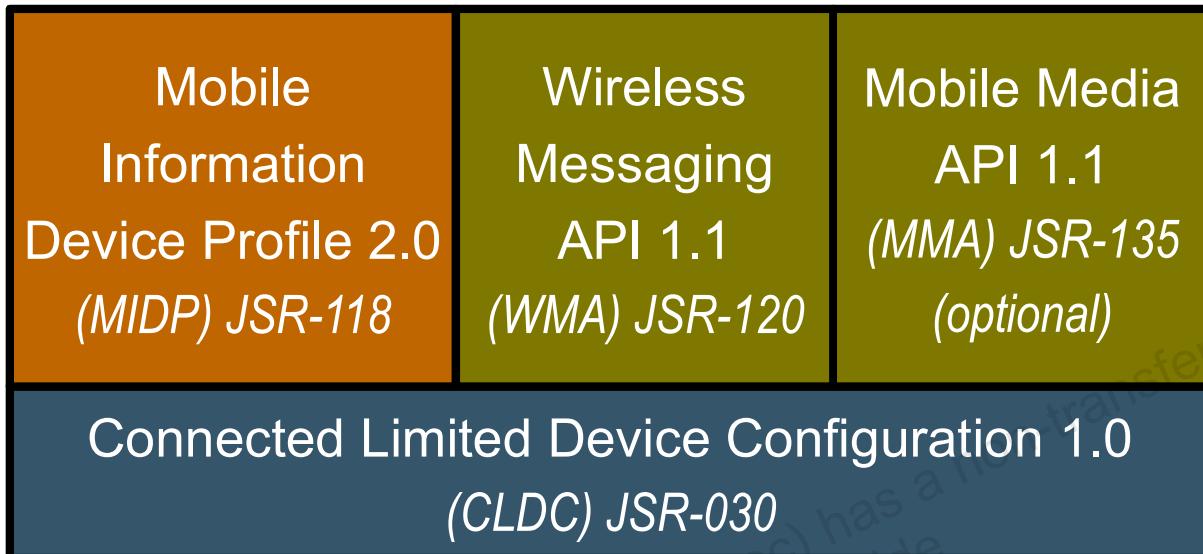
Operating systems that can be found on smartphones include Symbian OS (including S60 series), iOS for Apple mobile devines, Palm WebOS, BlackBerry OS, Samsung bada phones running Linux, Binary Runtime Environment for Wireless, Windows Mobile, Android and Maemo. WebOS, Android and Maemo are built on top of Linux. Apple’s iOS is derived from Mac OS X, which is Unix-based.

- **iPhone versus Android:** iPhone is a “closed” platform, based on Apple’s iOS operating systems which is derived from Mac OS X, with the hardware and software controlled by Apple. This is a double-edged sword. The user experience and integration across the Apple platform, as well as Enterprise support, are impressive.

Android is an open architecture, but is experiencing OS fragmentation due to different models and carriers. This puts a lot of strain on application developers to support different versions of the software and it can impede forward progress on application features. It also frustrates users when they have to wait indefinitely for their wireless carriers to roll out OS updates. Android applications do not have to go through the rigor, some might say over-controlled, application process like the iPhone, so there are more applications available. However, without an approval process, quality, usefulness and tastefulness of Android apps can be questionable. Also, Android does not sync to the desktop as easily or cleanly as the Apple devices.

- **The iPad:** It is a tablet computer designed and developed by Apple. It is particularly marketed as a platform for audio and visual media such as books, periodicals, movies, music, and games, as well as Web content. At about 700 grams, its size and weight are between those of most contemporary smartphones and laptop computers. Apple released the iPad in April 2010, and sold 3 million of the devices in 80 days.
- **Netbooks:** They (sometimes also called mini notebooks or ultraportables) are a branch of subnotebooks, a rapidly evolving category of small, lightweight, and inexpensive laptop computers suited for general computing and accessing Web-based applications; they are often marketed as "companion devices", that is, to augment a user's other computer access.
- **A Tablet PC:** It is a laptop PC, equipped with a stylus or a touchscreen. This form factor is intended to offer a more mobile PC; Tablet PCs may be used where notebooks are impractical or unwieldy, or do not provide the needed functionality.
- **Windows Phone 7 Series:** Announced in February of 2010, Microsoft is offering a new, fresh OS for their phones that will appear in 2010. Some have called it a Zune and XBox phone (alluding to Microsoft's media player and game console). It remains to be seen if the impressive feature set will be sufficient to overcome the iPhone and Android's lead.

JTWI Architecture



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The Java Technology for the Wireless Industry (JTWI) specification defines the requirements for a configuration, profile, and set of optional packages that need to be provided in a Java runtime environment on a mobile device to be considered JTWI-compliant. The slide illustrates the JTWI architecture.

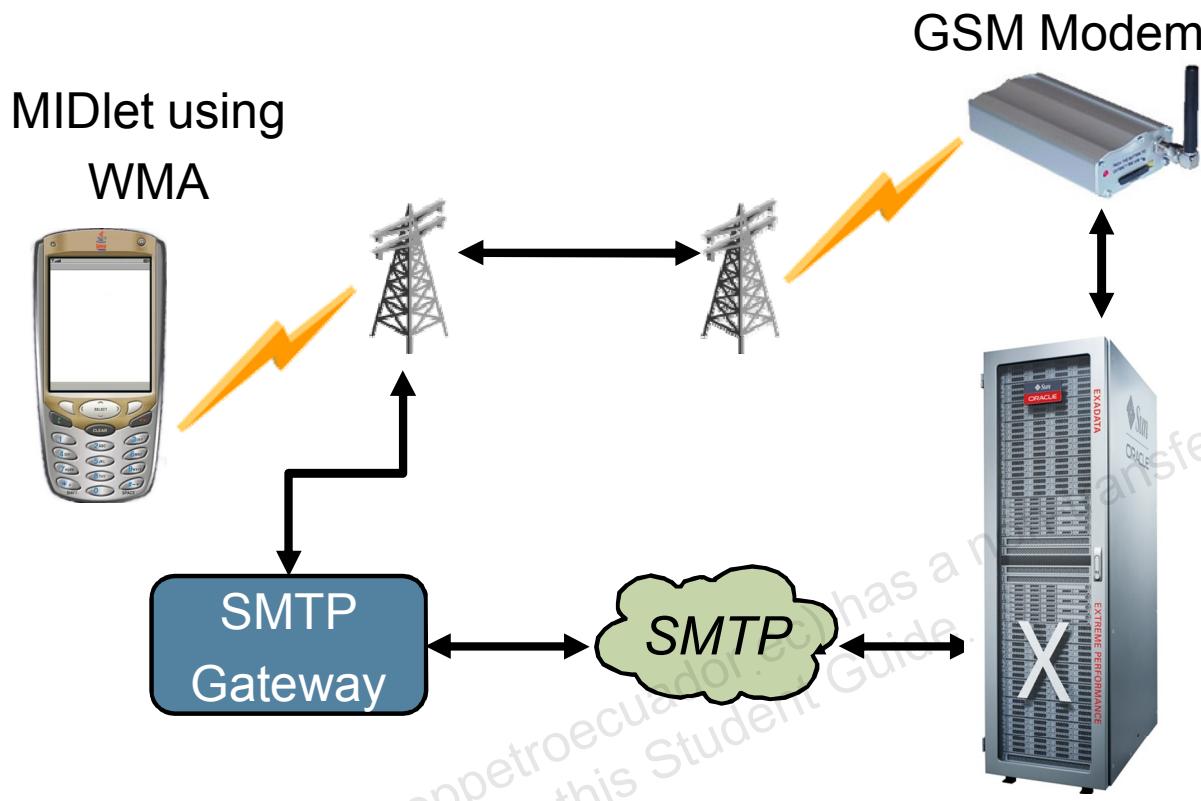
A Java application built to execute within the context and life cycle management facilities of the Mobile Information Device Profile (MIDP) is referred to as a MIDlet. To assist in the development of MIDlets, Sun provides the Sun JavaTM Wireless Toolkit for CLDC (WTK). This environment supports the coding, building, JTWI device emulation, and monitoring tools for the development of MIDlets.

The developers of the popular Netbeans Java Integrated Development Environment (IDE) have created an add-on called the Mobility Pack for the development of MIDlets on JTWI-compliant devices. It has visual tools for MIDlet development, a good emulator, integration with WTK, and much more.

The Wireless Messaging API (WMA) provides for the handling of remote messaging using one of two communication protocols, Short Message System (SMS) and Cell Broadcast Short (CBS). SMS support in WMA is bidirectional. Messages can be created, sent, and received. With respect to WMA, CBS is a receive-only messaging system.

WMA 2.0, as described in JSR-205, adds support for the Multimedia Messaging System (MMS) protocol.

SMS Asynchronous Messaging



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

SMS is a good candidate for an asynchronous messaging model for mobile devices. Assuming the MIDlet is a remote client of a distributed enterprise application component, the component can be adapted for SMS communication. The strategies for this adaptation vary, but can include using:

A platform-specific SMS Java API to communicate with a Global System for Mobile Communication (GSM) modem

A cellular provider's SMTP gateway

Java ME Web Service API

MIDlet using

WSA



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

You can get synchronous request/response style communication from a MIDlet with the Java ME Web Services API (WSA) defined in JSR-172. This optional package provides for the usage of a subset of the JAX-RPC API for communicating with Simple Object Access Protocol (SOAP)-based Web services over HTTP. The slide illustrates the Java ME Web services API.

The core MIDP API contains an `HttpConnection` class that you can use for communication with Representational State Transfer (REST)-oriented web services.

Microbrowsers

- A microbrowser is a browser designed to run in a small embedded device, such as a mobile phone or PDA.
- These microbrowsers are usually capable of rendering multiple markup languages that have been optimized for low bandwidth communication and display on small screens.



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The only significant difference between the two is that the PDA might have a slightly larger screen, a more robust operating system, and a touch sensitive screen or miniature keyboard for more efficient character data entry than is typically found on a mobile phone.

XHTML MP Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN"
"http://www.wapforum.org/DTD/xhtml-mobile10.dtd">
<html>
<head>
<title>My Hello World Page</title>
</head>
<body>
<imgsrc="images/happyface.gif"/><br/>
<p>
    Hello World. Click
<a href="http://www.sample.com">here</a>.
</p>
</body>
</html>
```



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

WAP 2.0 stipulates end-to-end HTTP protocol communication. It retracts the protocol that was mandated as part of the original WAP specification. Therefore, WAP gateways are not required for XHTML MP content.

XHTML MP 1.0 does not support any form of ECMAScript (JavaScript) at this point. Future versions might support some subset of the ECMAScript specification. In the meantime, some microbrowsers offer customized mechanisms to associate JavaScript event handlers with document elements to offer the features of Dynamic XHTML MP and Asynchronous JavaScript and XML (Ajax) capabilities. The Opera Platform is one such development environment. The Opera Mobile and Mini microbrowsers support the JavaScript semantics defined by developers using their development environment.

XHTML Mobile Profile

- XHTML Mobile Profile (XHTML MP) is a specification maintained by the Open Mobile Alliance, a standards body for the mobile industry.
- XHTML Mobile Profile 1.0 is the official markup language of WAP 2.0, thus supplanting the Wireless Markup Language (WML).
- XHTML MP will probably become the dominant markup language for mobile devices in the near future, and already has the largest degree of support in microbrowsers.

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The actual XHTML MP 1.0 specification, is little more than the addition of a few tags on top of the XHTML Basic W3C specification. These extra tags provide for the referencing of WAP Cascading Style Sheets (WCSS), another specification maintained by the Open Mobile Alliance, which describes a subset of the W3C Cascading Style Sheets 2.0 (CSS2) specification.

XHTML Basic limits you to a simple set of document types:

- Basic text (including headings, paragraphs, and lists)
- Hyperlinks and links to related documents
- Basic forms
- Basic tables
- Images
- Meta information

MIDlets or Microbrowsers

- Factors that influence the decision include:
 - The desire for a richer interface (MIDlets)
 - The desire to support robust asynchronous messaging (MIDlets)
 - The desire to support non-HTML Web services (MIDlets)
 - The desire to simplify the development process (XHTML MP)
 - The desire to support other more robust device platforms at the same time (XHTML MP)
- iPhone does not run Java VM



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Client Side Adaptation

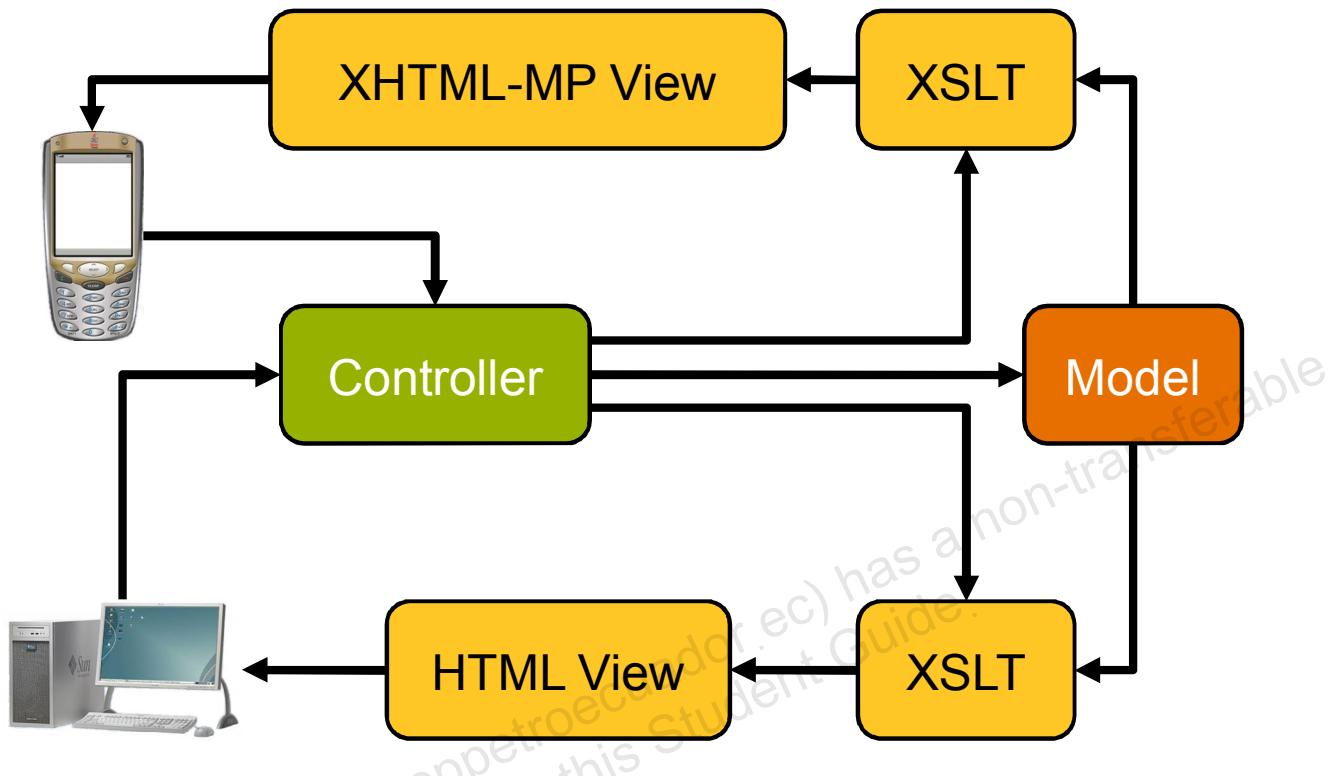
- Most microbrowsers can parse and display an XHTML page that is designed for display on desktop computer screens.
- Web sites that were never intended for display on mobile devices are represented in a manner that is visually displeasing and has low user acceptance.
- mobileOK is designed to provide additional metadata context to Web content, relative to its appropriateness for display on mobile devices.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Microbrowsers can parse and display an XHTML page that is designed for display on desktop computer screens. This is referred to as client-side adaptation. All microbrowsers have varying degrees of success in this regard. Often, client-side adaptation of a Web site that was never intended for display on mobile devices is represented in a manner that is visually displeasing and has low user acceptance.

Multiserving



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Multiserving is the process of presenting an interface to the user that is appropriate to the technology making the request for the interface.

As an example, imagine that you want to serve up a blog from your web server process to both computer browser users and wireless PDA microbrowser users. You could implement a framework that examines the user agent of the browser making the request of the blog. If the user agent is a full computer browser, then you could make some assumptions about the size of the monitor the user is using, and display HTML with images and a banner advertisement within the context of a graphic intensive look and feel. If the user agent is a microbrowser, you could respond with XHTML-MP containing just the text of the blog. Multiserving implementations should have a solid Separation of Concerns (SoC). This means that logic, style, and data should exist as separate aspects of the application. As it relates to the Model-View-Controller (MVC) architecture, the model component encompasses the logic and data aspects of the SoC paradigm. Multiserving applications with multiple view components should provide a presentation style to the same data element. To do otherwise would make Web flow and data integrity management problematic.

To alleviate the need of multiserving and assist microbrowsers in client-side adaptation, the W3C has started a Mobile Web Initiative. The Mobile Web Initiative has set out to define a Mobile Web Best Practices recommendation and a schema for the tagging of XHTML. That schema, called mobileOK, is designed to provide additional metadata context to Web content, relative to its appropriateness for display on mobile devices.

Web Browser Technology Best Practices

The following topics discuss best practices for selecting technologies that effect the architectural concerns of a web browser interface client.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Out-of-band JavaScript Callbacks

- Use JavaScript to make an HTTP request to a web service without reloading the Web page.
- Use XML or JavaScript Object Notion (JSON) for messages.
- Parsed responses usually result in a view change
- Are often referred to as Asynchronous JavaScript and XML (AJAX), even when JSON is used instead of XML



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

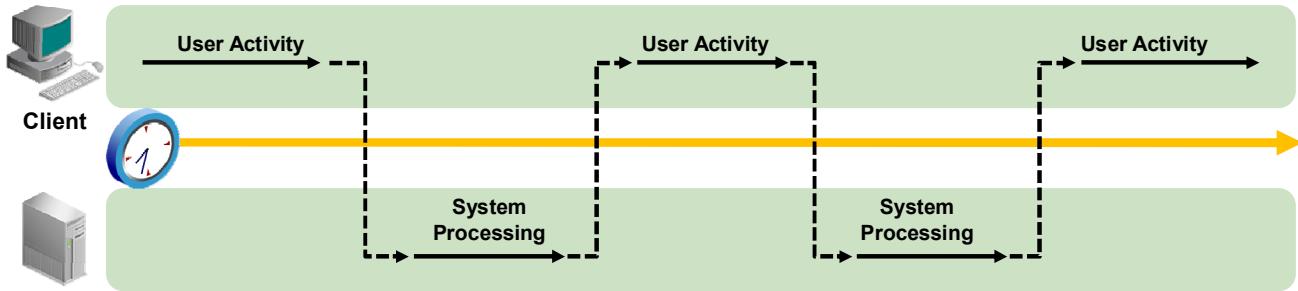
The idea of sending and receiving XML messages in out-of-band JavaScript server callbacks is better known as Ajax. The term was coined in an article by Jesse James Garrett to be short for Asynchronous JavaScript and XML, and is often used in such a way as to refer to all out-of-band JavaScript callbacks, regardless of whether the messages are XML.

The first out-of-band callback mechanism was written by Alex Hopmann from Microsoft. He wrote XMLHttpRequest as an ActiveX control for the Outlook Web Access (OWA) for Exchange 2000 developers to create a better user experience for users of OWA. JavaScript developers quickly realized the potential of this mechanism and encouraged the developers of the Open Source Mozilla project to develop something similar in their flavor of the JavaScript language. They responded with XMLHttpRequest (XHR).

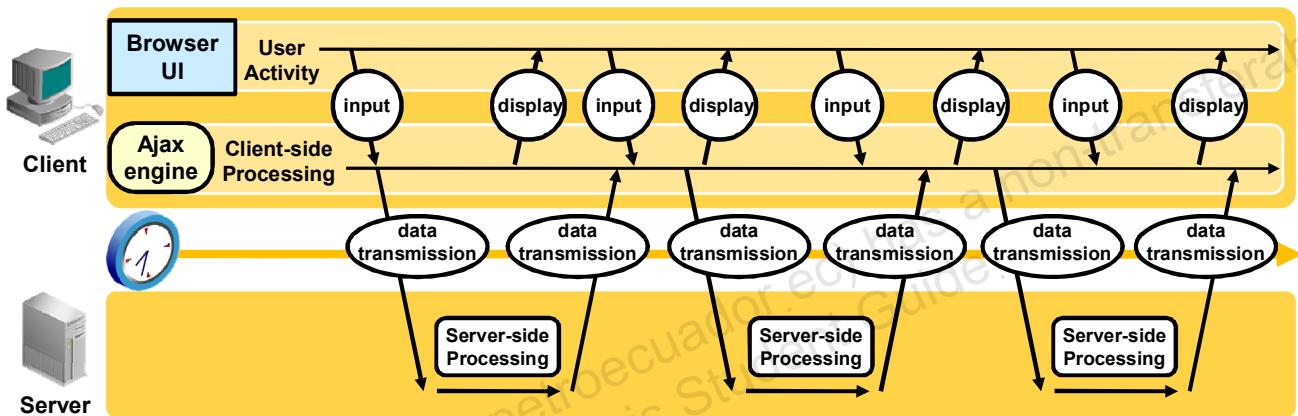
The XMLHttpRequest ECMAScript Object API specification is now on track to become a recommendation of the W3C.

Out-of-Band JavaScript Callbacks

Classic Web Application Model (synchronous)



Ajax Web Application Model (asynchronous)



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

There are various reasons why a web developer might create a series of wizard style forms in a Web flow for user interaction. Sometimes, however, those reasons are by necessity, given the requirement to communicate with a server process to establish additional data gathering requirements. For example, the input form might need to change based on a selection from a drop-down input element.

Depending on the nature of the application, this requirement for a round-trip to the server, and subsequent page redraw, can be an annoying experience for the user. While this might be tolerable for an e-commerce checkout scenario, it might not be tolerable for an underwriting application. All the dynamic HTML JavaScript effects will be of little value, if the user experience is disjointed.

Fortunately, there are now functions in most flavors of the JavaScript language that allow for an out-of-band callback to the server. This callback is a simple HTTP request/response that is transparent to the user. It is, however, up to the JavaScript code to formulate the request and make sense of the response. Often, these request and response messages are represented in XML, for easy processing.

The XMLHttpRequest Messages

- User interfaces that make use of out-of-band JavaScript callbacks, significantly reduce their bandwidth requirements on a per user basis.
- This is due to the fact that view-oriented data only needs to be sent to the browser on the initial page load.
- After that, the size of the messages is the only concern.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The size of the request and response messages is generally of concern because of the latency issues that a user can encounter with the user interface. The more concise the messages are, the less data needs to be pushed across the wire.

While they might seem attractive at first, in anything other than third-party services, SOAP messages are considered to be heavy messages for JavaScript callbacks. They contain a great deal of metadata. Even customized XML, in strict Ajax fashion, can be a bit metadata heavy, depending on the human readability of the element names.

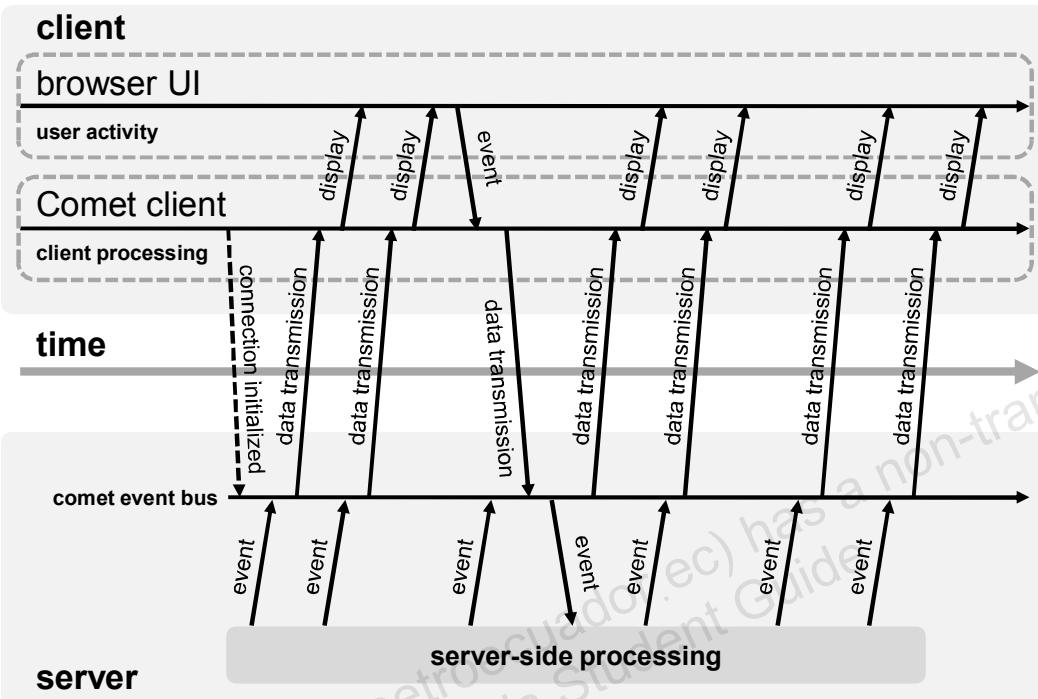
The JavaScript Object Notion (JSON) is an excellent alternative to XML for a lightweight data-interchange format. Its reach extends far beyond JavaScript technology, and can be used in dozens of other languages. Parsers and scriptifiers are available to deserialize and serialize JSON text strings.

JSON is specified by RFC-4627 with the Internet Engineering Task Force (IETF).

JSONP (JSON with Padding) was created as a workaround to the cross domain problem. The cross domain problem refers to the fact that Ajax code run from website A can't access data from website B. By wrapping your JSON data dynamically in a function call, you could call your function with the dynamic data, which is a technique called dynamic JavaScript insertion.

Comet

Comet Web application model



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The term Ajax was invented to describe the technologies that provide background request-response between a web server and client browser. With Ajax, web applications can retrieve data from the server in the background, essentially preloading information that is revealed in response to a user gesture and giving the impression of a more responsive web page. Although the data appears to be downloaded asynchronously, the server responds to client requests—typically through the use of the XMLHttpRequest object.

Web protocols require that servers respond to browser requests. Ajax falls short for multi-user, interactive pages because one user won't see changes other users make until the browser sends a request. In order for a server to push data to a browser asynchronously, some design workarounds are required. These design strategies are broadly described by the term Reverse Ajax.

One strategy is to use browser polling, in which the browser makes a request of the server every few seconds. Another strategy, known as piggybacking, delays a server's page update until the next request is received from the browser, at which time the server sends the pending update along with the response.

A third strategy, and the one used in the example application presented in this article, is to design with long-lived HTTP connections between the browser and server. With this strategy, a browser does not poll the server for updates; instead, the server has an open line of communication it can use to send data to the browser asynchronously. Such connections reduce the latency with which messages are passed to the server. This technique is known as Comet—a play on words that can be appreciated by users of household cleansers.

WebSockets

"Reducing kilobytes of data to 2 bytes...and reducing latency from 150ms to 50ms is far more than marginal. In fact, these two factors alone are enough to make Web Sockets seriously interesting to Google."

- Ian Hickson, HTML 5 Specification Lead

- A Web socket is a socket connection that can be established between a (modern) Web browser and a Web container.
- It provides a low-latency, bi-directional communication "parallel" to the HTTP channel.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

In contrast to Server-Sent Events, the WebSocket protocol is not built on top of HTTP. However, the WebSocket protocol defines the HTTP handshake behavior to switch an existing HTTP connection to a lower level WebSocket connection. WebSockets does not try to simulate a server push channel over HTTP. It just defines a framing protocol on top of TCP. In this way WebSockets enables two-way communication natively.

Like the Server-Sent Events specification, WebSockets specifies an API as well as a wire protocol. The WebSockets API specification includes a new HTML element, `WebSocket`.

Example JavaScript using the WebSocket interface

```
<html>
  <head>
    <script type='text/javascript'>
      var ws = new WebSocket('ws://localhost:8876/Channel',
'mySubprotocol.example.org');
      ws.onmessage = function (message) {
        var messages = document.getElementById('messages');
        messages.innerHTML += "<br>[in] " + message.data;
      };

      sendmsg = function() {
        var message = document.getElementById('message_to_send').value
        document.getElementById('message_to_send').value = ''
        ws.send(message);
        var messages = document.getElementById('messages');
        messages.innerHTML += "<br>[out] " + message;
      };
    </script>
  </head>
  <body>
    <form>
      <input type="text" id="message_to_send" name="msg"/>
      <input type="button" name="btn" id="sendMsg" value="Send"
onclick="javascript:sendmsg();">
      <div id="messages"></div>
    </form>
  </body>
</html>
```

Web 2.0

- A paradigm of web applications branded by the O'Reilly Group
- The user interaction, whether indirectly or directly, adds the greatest amount of value to a website.
- Features some aspect of:
 - Social networking
 - Social bookmarking and tagging
 - RIA-style interfaces
 - Mashups with third-party service providers



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

So, what is this Web 2.0 business anyway? The idea for branding a new paradigm of web applications came from the O'Reilly Group. It is not a specification, but rather a classification of web applications where the user interaction, whether indirectly or directly, adds the greatest amount of value to the website. These sites often also feature some aspect of social networking, social bookmarking, RIA-style interfaces, and mashups with third-party service providers.

Years ago, when Ward Cunningham originated the first Wiki, with his *Portland Pattern Repository* [<http://c2.com>], the idea of allowing average users to author the content of a website was considered reckless and dangerous, especially for the commercial sector. It was not until Amazon.com allowed their users to rate and comment on the products that the company sold, that other began to take notice.

Web 2.0 example websites like Facebook, Tweeter, Youtube, Wikipedia, and Flickr have proven that the users themselves can provide the greatest degree of success to any marketing effort.

Web 3.0

Imagine this: You go online to book the perfect vacation. Instead of having to search through two hundred flights, forty hotel rooms, and eight car rental companies, an Internet application will your personal ontology, consisting of contextual descriptors you've decided to share, and give you a

short, customized list of options. No more having to manually enter in everything from the price of the flight to the type of car – you'll get your very own view of products and information on the Internet, based on your preferences. This is Web 3.0.

As the second evolution of the Internet, Web 3.0 jumps into the middle of the information exchange problem with a semantic view of the information highway. Much like how humans use spoken language to communicate, Web 3.0 uses a linguistic base for exchanging information on the Internet, as opposed to the functional and process-driven base used by Web 2.0. Presently, there are two main problems associated with Web 3.0:

Machines have a terrible time resolving ambiguity. Humans are good at it, because we have learned to use contextual clues and pattern interpolation. Just one look from an annoyed spouse, along with the comment, "You're in my way," are enough to get us to move. A computer might have a difficult time resolving the spatial and temporal context for the indirect request for movement.

Core pieces of the semantic "language" of Web 3.0 are missing, such as domain-based languages, ontology builders, and classification and authentication services. Some of us already know what those pieces are, and where they go.

Web 2.0 applications can be useful for focused and well-defined tasks. When information is integrated from many sources, in varying degrees of completeness and meaning, the problem set ceases to be imperative (rule and sequence driven), and is more functional in nature. Think of the world as one giant spreadsheet with thousands of simple rules linking different areas of interest. Change one variable, and whole parts of the spreadsheet can be impacted. It is in this functional model of the world that Web 3.0 provides a useful paradigm for exchanging information.

Web 3.0 takes a page from nature in constructing useful patterns into clusters of correlated concepts and associations. Some of these patterns can be declared, others derived, but all are based on shared contextual attributes. In applications in which sequences or contexts are not well specified, such as loosely associated news articles, using a semantic approach offers pattern detection based on self-describing pieces of information and associations. Web 3.0 offers a method for arbitrating disparate information sources based on published definitions.

But the big plus is that information can be combined in new and creative ways. Web 2.0 assumes you know how, why and when information is associated. Web 3.0 only assumes the context of information has been declared to some level of specificity, and you have some mechanism to make sense out of the pieces.

Search Engine Optimization

- Process of increasing the likelihood of your webpages being found when a user makes use of a search engine to find related content
- Web crawlers follow links on webpages, parsing them along the way for keywords that provide some context relative to the topic of the page.
- Only invoke GET requests based on hypertext links found on a page
- Portions of your website that are accessible only via a POST request, or a JavaScript submit operation, will not be found by the search engine bots.

 ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Customer facing applications must be concerned with Search Engine Optimization (SEO). SEO is the process of designing webpages and managing their presence, in such a way as to increase the likelihood of the pages being found when a user makes use of a search engine to find related content.

SEO is of significant architectural concern for the client tier. Unaccounted for, it could result in a failed business venture. Every effort needs to be made to keep web applications as web crawler friendly as possible.

A Web crawler is an automated HTTP client that follows links on webpages, parsing them along the way for keywords that provide some context relative to the topic of the page. Search engine bots (short for robots), or web crawlers, do not submit POST requests or process JavaScript code. They only invoke GET requests based on hypertext links found on a page. Therefore, any portion of your website that is accessible only by a POST request, or a JavaScript submit operation, is not found by the search engine bots.

Search engine algorithms very slightly in how they rank webpages. In general, the following are considered best practices for SEO:

- Populate the <title> tag accordingly
- Populate the description meta tag content with not more than 250 words
- Populate image alt attributes with descriptive text
- Provide descriptive content within the first 100K of the HTML

Contrary to popular belief, the keyword's meta tag is no longer observed by most major search engines, and is probably not worth the time to implement.

Great caution should be exercised with the usage of RIA frameworks relative to their effect on SEO. It might be acceptable to draw the view component dynamically with the JavaScript language, but acquiring the data content using a JavaScript callback is not wise. Such content would be transparent to the search engine bot. Therefore, be wary of RIA frameworks that take an all or nothing approach to content management. You might end up with nothing of significant value from a Web crawler perspective.

If you have a catalog of products, consider writing a script that dynamically generates pages with lists of links to all of your products. Placing links to these pages in an inconspicuous location on the main page of your website might help the search engine bots crawl your website.

What Do You Think?

“I would argue that the model for today is the chubby client. The chubby client is the client that runs a lightweight application component that is largely responsible for the user interface. This application component acts on behalf of the user to request data from external sources and then format that data for presentation. Run an app on the iPhone and that's what you see. That's the model.

That model tends to pull development focus forward into the frontend part of SOA. In my view, it shifts the focus from Web Service focused development and all that rigorous issue set into something much more RESTful.”

- *Tom Nolle*, Chief Strategist at ExperianSphere



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Client tier development roles
- Information Architecture Client tier concerns
- Selecting user interface devices and technologies
- **Discovering reusability in the Client tier**
- Deployment strategies for the user interface
- Security concerns in the Client tier
- Testing



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Discovering Reusability in the Client Tier

- You can have architectural reusability in the Client tier by documenting repeatable human interaction patterns and design elements.
- You can also have it through the use of third-party web services.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

User Interface Design Patterns

- Reference existing HCI pattern catalogs.
- Consider creating a set of User Interface Design Guidelines that establish the style and method of presentation and interaction components for the enterprise.
- Look at <http://developer.yahoo.com/ypatterns> for sets of UI patterns.

 ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

You can apply pattern languages to any repeatable process. User interface design elements that implement some sort of human computer interaction are no different.

Many companies author their own set of User Interface Design Guidelines that establish the style and method of presentation and interaction components for the enterprise.

This takes the definition of the look and feel from an individual designer and makes it available to all would-be interface designers within the enterprise.

Such a catalog provides end users with the continuity they expect from the interface, and significantly increases the time to market for new interfaces.

Third Party Web Service APIs

- When it comes to websites, a mashup is a site that uses third-party APIs to create a value-added experience for the users of the website.
- These APIs are web services that are usually accessed using the XMLHttpRequest JavaScript mechanism.
- Example: BAM Dashboards



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The free service provided by the API author has the effect of increasing the publicity and marketing presence of its author.

Web service APIs tend to provide one of the following sorts of services.

- Summary or full view components to related data items (for example, RSS and streaming media feeds)
- Implementations of difficult to develop human interaction components (for example, mapping tools, address validators, dictionaries)
- Metadata services designed to help users find your content
- Product catalog access to help affiliate sales

While third-party web service APIs might seem attractive on the surface, great care should be taken in their selection. The availability of a free web service API is often designed to divert the attention of the user toward the provider. This could result in a break in the continuity and consistency of the user interface, disorient the user, or cause them to leave your website and go to the providers website.

Lesson Agenda

- Client tier development roles
- Information Architecture Client tier concerns
- Selecting user interface devices and technologies
- Discovering reusability in the Client tier
- Deployment strategies for the user interface
- Security concerns in the Client tier
- Testing

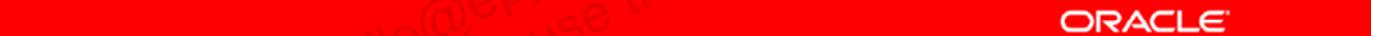


ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Deployment Strategies for the User Interface

- User interfaces based on a Java GUI toolkit require an installation and update process.
- The following topics address these concerns:
 - Using installation utilities
 - Java Web Start
 - Application Client Container



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Installation Utilities

Use installation utilities such as:

- InstallShield (Commercial)
- Inno Setup (Freeware)
- NullsoftScriptabe Install System (NSIS) (Open Source)



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Installation Utilities

- While desktop GUI toolkit-based applications might seem attractive on the surface for their rich client interface and broad flexibility, they suffer from issues relating to installation and maintenance.
- Getting users to properly install and execute Java applications can be problematic. Users might have to:
 - Download and install a Java Runtime Environment (JRE).
 - Download and save an application JAR file in a directory.
 - Modify the system path to include the bin directory containing the java command.
 - Open a command shell and execute
`java -jar sampleapp.jar` to start the application.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Some application developers use an installation utility, such as the commercial InstallShield, the free Inno Setup, or the Open Source Nullsoft Scriptable Install System (NSIS) to simplify software installation and subsequent application execution. However, do you include the JRE software with the installation bundle? Do you make two installation packages available, one with and one without the JRE software?

The next question is, how do you manage updates? Will the user need to completely reinstall with the new version? Does the user need an uninstall script? How do you force an update in the event of the discovery of a major security vulnerability, or force an update to a new communication protocol or feature set?

Java Web Start

- Installation, update, and execution kicked off from a web URL.
- Required JRE is automatically installed if not present on user machine.
- Application is cached on user machine.
- Updates are easily forced by changing the server-side JNLP file.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

If JRE version 1.4.2 or greater is installed on the user's system, there is a good chance that the Java Web Start software is already installed on the system.

If not, some appropriate JavaScript code on the webpage that kicks the application off can start the install of the appropriate JRE software on the user's system first, and then invoke the Java Web Start installation process.

During installation of the JRE, Java Web Start software registers itself as the appropriate handler of the application/x-java-jnlp-file mime type within the user's browser or system mime-type mappings or both. The Web server that serves the JAR file containing the client application, must also have a mime-type mapping that associates all files ending in JNLP to be of content type application/x-java-jnlp-file.

To execute a client application that is served using Java Web Start, a user must first click a link in the Web browser that points to a Java Network Launching Protocol (JNLP) file. A JNLP file is an XML file that describes metadata about the client application, including the required version of the JRE and the location of the client application Jar file.

If the current version of the client application JAR file is not available in the Java Web Start cache, it is downloaded before execution.

Application Client Container

- A client application can certainly run inside a standalone JVM implementation, and remotely connect to an application server.
- However, if you run your client application inside an Application Client Container (ACC), you can have some of the application container services that the server uses available to you.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The ACC is a lightweight container for the client that provides support for security authentication and authorization, referential naming, and annotations. Transactional semantics can also extend across the client-server communication. Using the ACC with your client can greatly simplify the coding effort for remote clients.

Java Web Start requires that the client application be deployed in an ACC. Application servers may support bundling ACC applications into Java Web Start.

Lesson Agenda

- Client tier development roles
- Information Architecture Client tier concerns
- Selecting user interface devices and technologies
- Discovering reusability in the Client tier
- Deployment strategies for the user interface
- **Security concerns in the Client tier**
- Testing

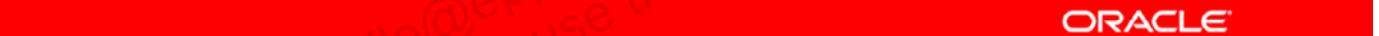


ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Security Concerns in the Client Tier

- The Client tier interface should provide the user with an adequate defense against well known exploits that could compromise the user's machine while using your interface.
- It should also constrain the user input to a predetermined format to prevent possible exploits using basic user input mechanisms.
- Such constraints can also be considered a feature, because they allow the user the opportunity to correct any data format problems before its submittal to a server process.
- The following topics address these concerns.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Input Data Validation

- User input data that is destined for processing on the server side needs to be validated for predefined conformity before execution.
- Users do not always follow the instructions for filling out text field items.
- Users might become increasingly annoyed by interfaces that do not validate the user's input until the data is sent to the server for processing.
- Consider the usage of an RIA JavaScript framework that generates client-side input validation based on a common data validation mechanism.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

However, the user might become increasingly annoyed by interfaces that do not validate the user's input until the data is sent to the server for processing.

Therefore, you should provide a data validation mechanism on the client side as well as the server side.

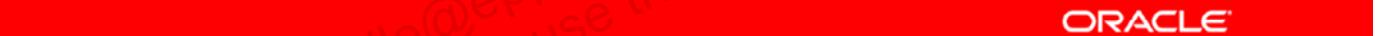
With web browser interfaces, this is commonly achieved using JavaScript technology.

In the past, it was not uncommon for the data validation to be manually implemented both on the server side, in Java technology, and on the client side in JavaScript technology. This tended to create a data validation consistency problem between the web tier and business tier.

Modern approaches to data validation on the client side involve frameworks that either source code generate JavaScript technology or use existing JavaScript libraries to implement the data validation as defined by configuration on the server side.

Code Injection

- All input from the client, regardless of whether the values were hard coded into the client interface, must be validated to ensure the data does not contain an exploitation attempt.
- Malicious users might try to inject code into an interface designed to exploit either the user's computer or the server.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Cross Site Scripting (XSS)

- JavaScript has access to the DOM of other browser windows and documents that are currently open on the user desktop.
- Malicious JavaScript could get access to sensitive data or submit browser requests on the user's behalf.
- All form fields should be checked for code injection.
- Never leave a browser window with sensitive information open while you visit an untrusted website in another window.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Any code that is designed to be executed on the server side is generally subjected to the same origin policy. The same origin policy says that code, such as Java Applets and JavaScript, to be effective, needs to come from the same place that provided the HTML document. Object signing of the code can allow extended capabilities for code that comes from a different origin.

Hackers usually attempt to inject code that appears to come from the same origin as the webpage. There are numerous creative ways to accomplish this. One way is to act as a man-in-the-middle between the user and the actual web server to which the user is making requests. A man in the middle would then modify the response content to include their malicious code. This approach has significant technical challenges. It is more likely that the hacker will simply try entering the script into text fields that might end up becoming interpreted as content on a Web page.

The code might attempt to read cookie values, read or manipulate the current webpage, or engage in Cross-Site Scripting (XSS). XSS refers to the notion that JavaScript technology in one browser can actually manipulate the DOM of a document in another browser, regardless of the website of origin.

For example, imagine malicious code on an exploited website that looks for other browser windows currently opened, and logged into, a well known stock trading application. Upon finding such a lucky strike, the malicious code might be able to execute stock trades on the user's behalf.

To help prevent such exploits. Validate all fields for the expected content and remove anything that can be interpreted as script before its display. Web sites that display user input like wikis, forums, social networking sites, reviews, and blog and article comments are particularly prone to XSS.

As a user, you should close all other browser windows before visiting Web sites of a personal or financial nature, where XSS exploits could do a great deal of damage.

Lesson Agenda

- Client tier development roles
- Information Architecture Client tier concerns
- Selecting user interface devices and technologies
- Discovering reusability in the Client tier
- Deployment strategies for the user interface
- Security concerns in the Client tier
- Testing



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Testing

- Design client with view to impact on NFRs
- Simulate testing of prototype: stress test
- How much above 100% do you go? 200%? 300% Why?
- Need to measure:
 - Usability
 - Consistency of Brand
- Testing tools:
 - Selenium
 - Mercury
 - JMeter
 - Badboy
 - Oracle e-Test Suite

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

In software testing, stress testing refers to tests that determine the robustness of software by testing beyond the limits of normal operation. Stress testing is particularly important for "mission critical" software, but is used for all types of software. Stress tests commonly put a greater emphasis on robustness, availability, and error handling under a heavy load, than on what would be considered correct behavior under normal circumstances.

User interface (UI) prototyping is an iterative development technique in which users are actively involved in the mocking-up of the UI for a system. UI prototypes have several purposes:

- As an analysis artifact that enables you to explore the problem space with your stakeholders.
- As a design artifact that enables you to explore the solution space of your system.
- A basis from which to explore the usability of your system.
- A vehicle for you to communicate the possible UI design(s) of your system.
- A potential foundation from which to continue developing the system (if you intend to throw the prototype away and start over from scratch then you don't need to invest the time writing quality code for your prototype).

Additional Resources

The following references provide additional information on the topics described in this lesson:

- Raskin, Jef. *The Humane Interface*. Boston, Addison Wesley, 2000.
- Anderson, Jonathan. *Effective UI: The Art of Building Great User Experience in Software*. O'Reilly Media, 2010.
- Norman, Donald. *The Design of Everyday Things*. Basic Books, 2002.
- Governor, James. *Web 2.0 Architectures: What Entrepreneurs and Information Architects Need to Know*. O'Reilly, 2009.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

- O'Rielly, Tim. *What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software*. [<http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-Web-20.html>] accessed Mar 2007
- Garrett, Jesse James. *Ajax: A New Approach to Web Applications*. [<http://www.adaptivepath.com/publications/essays/archives/000385.php>] accessed Mar 2007.
- Marinilli, Mauro. *Professional Java User Interfaces*. John Wiley & Sons, Ltd, 2006.
- Yahoo UI Patterns: <http://developer.yahoo.com/ypatterns>
- Shneiderman, Ben. *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (5th Edition) Addison Wesley, 2009.
- Morville, Peter. *Information Architecture for the World Wide Web: Designing Large-Scale Web Sites*. http://www.amazon.com/Information-Architecture-World-Wide-Web/dp/0596527349/ref=reg_hu-wl_mrai-recs
- Apple Computer. *Macintosh Human Interface Guidelines*. Addison-Wesley Professional, 1993.

Quiz

Unlike the other tiers of an Enterprise Architecture, the Client tier is most successful when:

- a. End users have a positive experience with the user interface
- b. All of the web browser types are supported
- c. Response time is sub-second for every screen
- d. A Model-View-Controller pattern is used



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: a

b, and c are really QoS considerations.

Quiz

Which of the following roles are key for the development of the Client Tier? (Choose all that apply.)

- a. Interface Designer
- b. Network Engineer
- c. Graphic Artist
- d. Data Table Architect



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: a, c

Quiz

BadUserExperience.com has designed a web site where the user sits for 10 seconds after logging in, and on a 20 item input form, will wipe out the data if the user makes an error, but only after they hit the submit button. Which of the following design principles are they violating? (Choose all that apply.)

- a. Don't make the user type more than they have to
- b. Don't let the user make a mistake
- c. Don't waste the user's time
- d. Don't have a single submit button on a page



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: a, b, c

d is a possibility, but there is no design principle for a single button.

They may also want to change their URL.

Quiz

Given the nature of the business your company does with local and foreign governments, which of the following are important to consider? (Choose all that apply.)

- a. Section 508
- b. Use Internet Explorer as the browser
- c. Use JSF or ADF
- d. Internationalization



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: a, d

Quiz

Given a situation where the company is exploring the use of mobile devices by customers of their new enterprise application, which technology for the UI would you recommend?

- a. Swing
- b. AWT
- c. JSF
- d. JavaFX



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: d

the best choice for support of mobile apps

Summary

In this lesson, you should have learned how to:

- Describe the roles involved in client tier development
- Define Information Architecture client tier concerns
- Describe how to select a user interface device that will fit your application requirements
- Describe how reuse can apply to the client tier
- Describe strategies for deploying Java desktop-based applications
- Be familiar with the security concerns of the client tier



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 8 Overview: Choosing Client Technologies

This practice covers the following topics:

- Choose appropriate client technologies based on the user's requirements.
- Refine the architecture to enable a better user experience.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

In this practice you will evaluate the user requirements and make recommendations for appropriate client technologies. You will also evaluate and incorporate technologies and refine the architecture to produce a better user experience.

Unauthorized reproduction or distribution prohibited. Copyright© 2015, Oracle and/or its affiliates.

Iveth Tello (iveth.tello@eppetroecuador.ec) has a non-transferable
license to use this Student Guide.

Developing an Architecture for the Web Tier

9

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the roles involved with the development of the Web tier
- Describe the Separation of Concerns
- Describe strategies for implementing the presentation concerns of the Web tier
- Describe strategies for implementing the data concerns of the Web tier
- Describe strategies for managing the presentation, data, and logic-related concerns of the Web tier
- Describe the advantages and disadvantages of request- and component-oriented Web tier frameworks



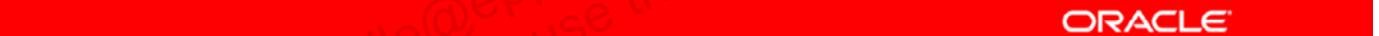
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

- Describe strategies for implementing authentication and authorization in the Web tier
- Address the concerns of scaling web applications

Discussion Questions

- What is the Separation of Concerns and why is it important? What is the worst application you ever had to maintain? How was it architected?
- How have you implemented security concerns in the past? How invasive was it to the architecture?
- What is the difference between an HTTP request-oriented and a component-oriented framework?
- What is web flow?
- What are the advantages of storing meta data with your content?
- How do you scale a web application?



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Responsibilities of the Web tier
- Separation of Concerns
- Comparing Web tier frameworks
- Providing security in the Web tier
- Scaling the Web tier



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Responsibilities of the Web Tier

- Managing user sessions and stateful entities
- Delegating service requests to the business tier
- Dispatching presentation to the client interface
- Managing access to content
- Managing resource security concerns

ORACLE®

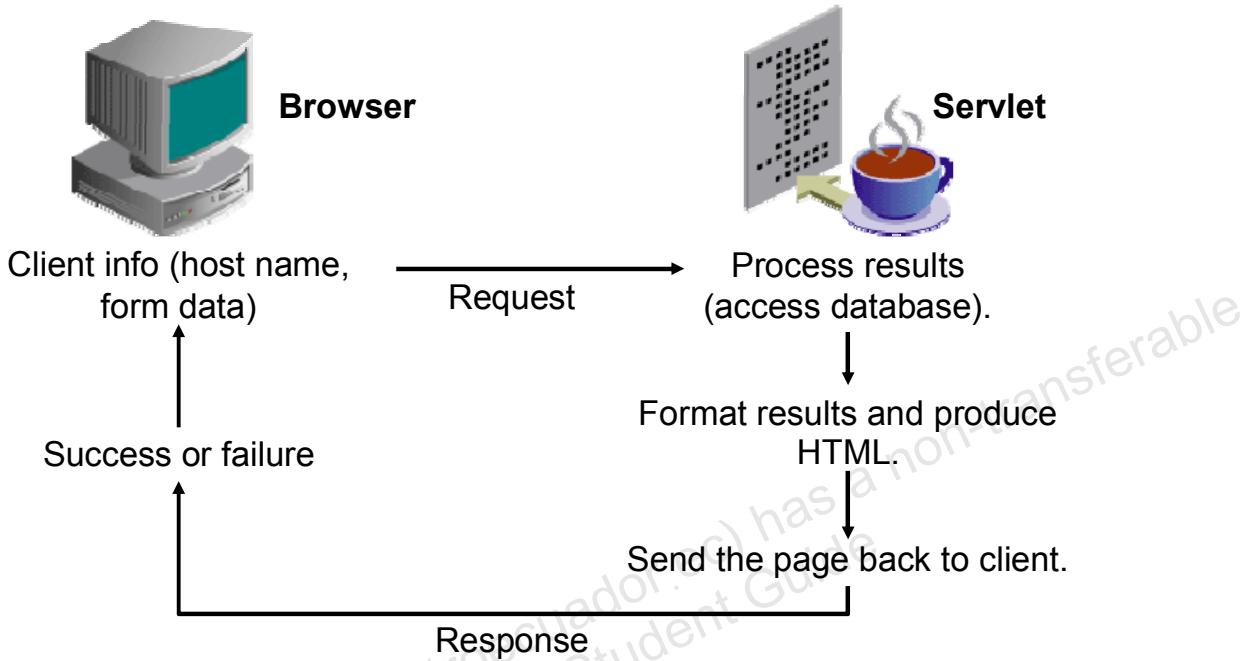
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The Web tier is responsible for bridging the gap between the user interface and the domain model of the business tier. Its goal is to provide the server-side programmatic support for the client tier, as dictated by the information architecture, without actually implementing any business logic. From an architectural standpoint, it is far from trivial. An improperly architected Web tier can have profound effects on the systemic qualities of the overall enterprise application.

Some of the core responsibilities of the Web tier include:

- Managing user sessions and stateful entities
- Delegating service requests to the business tier
- Dispatching presentation to the client interface
- Managing access to content
- Managing resource security concerns

What Is a Servlet?



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A servlet is a Java program that runs on a Java EE server and produces dynamic pages typically in response to client requests. The pages are then sent back to the client's web browser. The output format for servlets usually is HTML, but any other output format, including XML, can be used.

Before returning HTML pages, a servlet can perform any operation that a Java application can perform. For example, in a business environment, servlets access the business objects, so that you can send custom HTML pages and forms with embedded data to your end users.

Because servlets contain Java code, they are best suited for programs that perform more processing than presentation. Presentation is best left to JSPs, which are discussed in the following slide.

What Is a JavaServer Page?

A JSP:

- Is a text-based document that includes:
 - HTML
 - JSP tags
 - Java code (including calls to JavaBeans and servlets)
- Cleanly separates content creation from presentation logic
- Focuses on rapid development and easy modification of the user interface
- Focuses on presentation



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

JSPs are servlets that are written differently. The JSP technology separates the user interface from dynamic content generation, so that designers can easily change the overall page layout without altering the dynamic content. The JSP technology supports a reusable component-based design, making it easy and fast to build Web-based applications.

A key feature of the JSP technology is simplified page generation, HTML-like tags, and scriptlets written in the Java programming language that encapsulate the logic that generates the content for the page. By separating the presentation design from the application logic that generates the data, the JSP-enabled pages make it easier for organizations to reuse and share application logic through custom tags and JavaBeans-based components. This also separates the job responsibilities of the Web designer and the Java programmer. For example, custom tags and beans can be developed by the Java programmer and implemented by the web designer.

What Is JavaServer Faces?

A JSF:

- Is a server-side component framework for Web applications
- Implements the Model-View-Controller (MVC) framework
- Provides separation of navigational and data flow
- Is built for rapid application development (RAD) style development

Java EE 6: JSF 2 has deprecated JSP as its view layer, replacing it with the more JSF-centric Facelets.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

JSF is a component-based framework for developing Web-based applications. JSF provides a framework for using robust UI components in a consistent manner. JSF provides a RAD environment for building Web applications. JSF is developer tool-centric so that developers can use visual editors to create applications. JSF also combines the functionality of a JSP with a built-in Controller. These parts make JSF a true MVC framework.

Facelets: Replaces JSPs as the view layer in Java EE 6/JSF 2. Facelets supports templating via composite components. Facelets allows the developer to create reusable composition components. For example, you can take an existing JSP page and turn it into a reusable component. Facelets helps to resolve the mismatch between JSP's dynamic content and JSF's component-based model. Facelets was designed with the JSF life-cycle in mind. Using Facelets, you create templates that build a component tree, not a servlet. This supports greater reuse at the component level. Facelets also eliminates the need to write custom tags to use JSF components, because Facelets uses JSF custom components natively.

Oracle ADF Faces

The screenshot displays a web-based application titled "Storefront Demo". At the top left is a photograph of a person working at a computer in a store environment. The main title "Storefront Demo" is centered above two grids. The left grid, titled "Select a Supplier for Update", contains a table with columns: SupplierId, SupplierName, SupplierStatus, PhoneNumber, and Contact. The right grid, titled "Products", contains a table with columns: ProductName, Description, and Price. Both grids include "Search" and "Show Cart" buttons at the top.

SupplierId	SupplierName	SupplierStatus	PhoneNumber	Contact
100	Stuffz	ACTIVE	402.555.0158	contact
101	Nexus	ACTIVE	608.555.0114	contact
102	Gifts-N-More	ACTIVE	225.555.0181	contact
103	Emporium	ACTIVE	212.555.0198	contact
104	Jeffery And Michael's	ACTIVE	419.555.0167	contact
105	Games Galore	ACTIVE	630.555.0127	contact
106	Transistor City	ACTIVE	303.555.0177	contact
107	Mercury Imports	ACTIVE	862.555.0108	contact
108	BigSwamp	ACTIVE	248.555.0154	contact
109	Z-Mart	ACTIVE	959.555.0120	contact
110	Wally's Mart	ACTIVE	470.555.0180	contact
111	Ellis	ACTIVE	641.555.0194	contact
112	Electronics and More	ACTIVE	413.555.0140	contact
113	Zibbers	ACTIVE	717.555.0162	contact

ProductName	Description	Price
Treo 650 Phone/PDA	The unlocked PalmOne	\$299.99
Treo 700w Phone/PDA	The Palm® Treo™ 700	\$399.99
RAZR Cellular Phone	Thin is definitely in. At	\$259.99
Bluetooth Adaptor	SyNET's TBW-101UB Bl	\$19.99
Bluetooth Phone Head	Streamlined and sophis	\$49.99
Chocolate Phone	LG's newest mobile tre	\$499.99
Blackberry 8100c	You'll be amazed at th	\$499.99

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Oracle's Application Development Framework (ADF) Faces rich client (known also as ADF Faces) is a set of JavaServer Faces (JSF) components that include built-in Asynchronous JavaScript and XML (AJAX) functionality. While AJAX brings rich client-like functionality to browser-based applications, using JSF provides server-side control, which reduces the dependency on an abundance of JavaScript often found in typical AJAX applications.

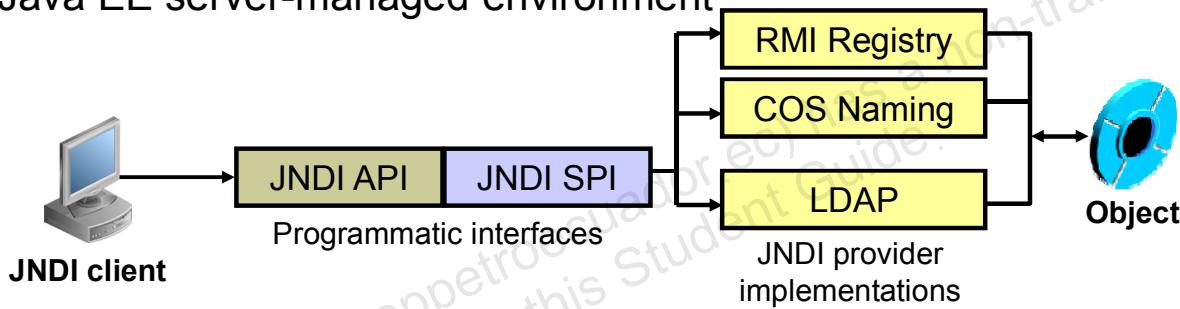
In addition to providing the JSP tags and UIComponent instances that would be expected of any JSF component set, the ADF Faces rich client framework (RCF) provides a client-side programming model familiar to developers accustomed to the JSF development model. However, the RCF is specifically different where necessary to deal with practical realities of JavaScript development required in standard AJAX development practices. Most of the RCF differs little from any standard JSF application; the server programming model is still JavaServer Faces, and the framework still uses the JavaServer Faces lifecycle, server-side component tree, and the expression language (EL). However, the RCF also provides a client-side programming model and lifecycle that execute independently of the server. Developers can find and manipulate components from JavaScript, for example use get and set properties, receive and queue events, and so forth, entirely from JavaScript. The RCF makes sure changes to component state are automatically synchronized back to the server to ensure consistency of state, and events are delivered, when necessary, to the server for further processing.

Java Naming and Directory Interface

The JNDI is:

- A standard set of interfaces that provide:
 - Naming services
 - Directory services
- A service provided by Java EE containers to locate Java EE resources or objects

Provides explicit lookups for applications running outside the Java EE server-managed environment



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Java EE applications use the following parts of the JNDI specifications to find other distributed objects:

- An application-level interface used by client programs to access a naming and directory service
- A service-provider interface used by the JNDI API to communicate with a provider of a naming and directory service, in a vendor-independent manner

Client applications use JNDI properties and the API to:

- Connect with a JNDI service provider by establishing an initial context
- Locate an object by its registered name by calling a lookup operation from the initial context

The server implements the JNDI Service Provider Interface (SPI) library that generalizes access to different types of JNDI provider implementations, such as Remote Method Invocation (RMI) Registry, Common Object Services Naming, Lightweight Directory Access Protocol (LDAP), and others. This enables different types of directory-service and naming-service implementations to be transparently used by the Java EE container.

The types of objects typically stored in a JNDI service provider include the bean interface for Enterprise JavaBeans (EJB) components, Java Database Connectivity (JDBC) data sources, and Java Message Service (JMS) resources.

Note: With Java EE 6, applications that run outside the Java EE-server managed container environment, such as Java SE applications must perform an explicit lookup. JNDI supports a global syntax for identifying Java EE components to simplify this explicit lookup. Applications running inside the Java EE-server managed container should use dependency injection.

Web Tier Development Roles

- **Web Designer:** Interface designer for web browser-based applications
- **Web Developer:** Implements all aspects of the Web tier
- **Information Architect:** Creates the vision for site navigation and templating

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Web Designer

A Web designer fulfills the role of the interface designer for browser-based client applications. If the source code for the view components is sufficiently decoupled from the framework that manages the Web tier, then the role of the Web designer is purely a Client tier concern. Otherwise, the web designing and developing might need to be handled by individuals fluent in the concerns of both the Client and Web tiers.

Web Developer

The web developer implements all aspects of the web tier. They must also work closely with the web interface designer to ensure that domain entities and content is properly integrated into the XHTML presentation components. They might also develop the client proxies and business delegates to the business tier components. While it might be considered more of a Client tier concern, because of their expertise from a programming perspective, the web developer often provides assistance with the JavaScript technology aspects of the Web-Client tier. That way, the web interface designer can focus on design-oriented issues.

Information Architect

The information architect works just as closely with the web designer (interface designer) as they do with the web developer. The information architects vision for site navigation and templating is implemented by the web developer.

Lesson Agenda

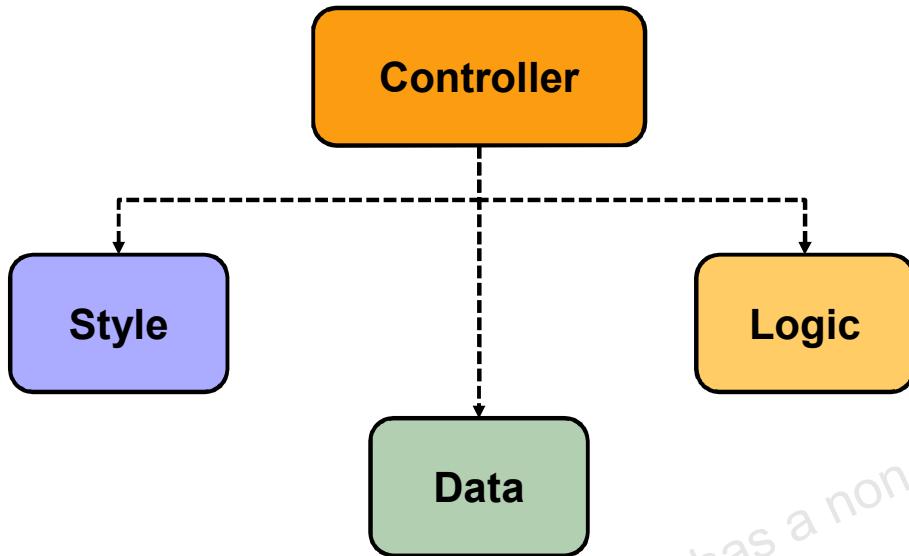
- Responsibilities of the Web tier
- Separation of Concerns
- Comparing Web tier frameworks
- Providing security in the Web tier
- Scaling the Web tier



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Separation of Concerns (SoC)



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The Separation of Concerns (SoC) is a paradigm for viewing the architectural aspects of an application as discrete elements that are loosely coupled. This loose coupling between the concerns of an application can speed application development and maintainability, by allowing designers and developers to focus on cohesive aspects with little, if any, cross cutting concerns from other aspects of the application.

In the Web tier, the most common division of concerns involves the separation of style, data, and logic, with a controlling mechanism to piece them together.

The SoC paradigm is the driving force behind many architectural patterns. It has proven to be one of the most effective ways to ensure a robust set of NFRs across most tiers of the architectural model. It can be found in RIA/Ajax frameworks in the Web tier, presentation frameworks in the Web tier, dependency injection frameworks in the business tier, and persistence frameworks in the integration tier.

Presentation Concerns

- Templating the layout view
- Compositing the view
- View generation
- Previewability



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Templating the Layout View

- **Inclusion:** Include reusable elements into the requested content page at runtime
- **Transclusion:** Access a lightweight view component, which selects a view template and the content to plug into it



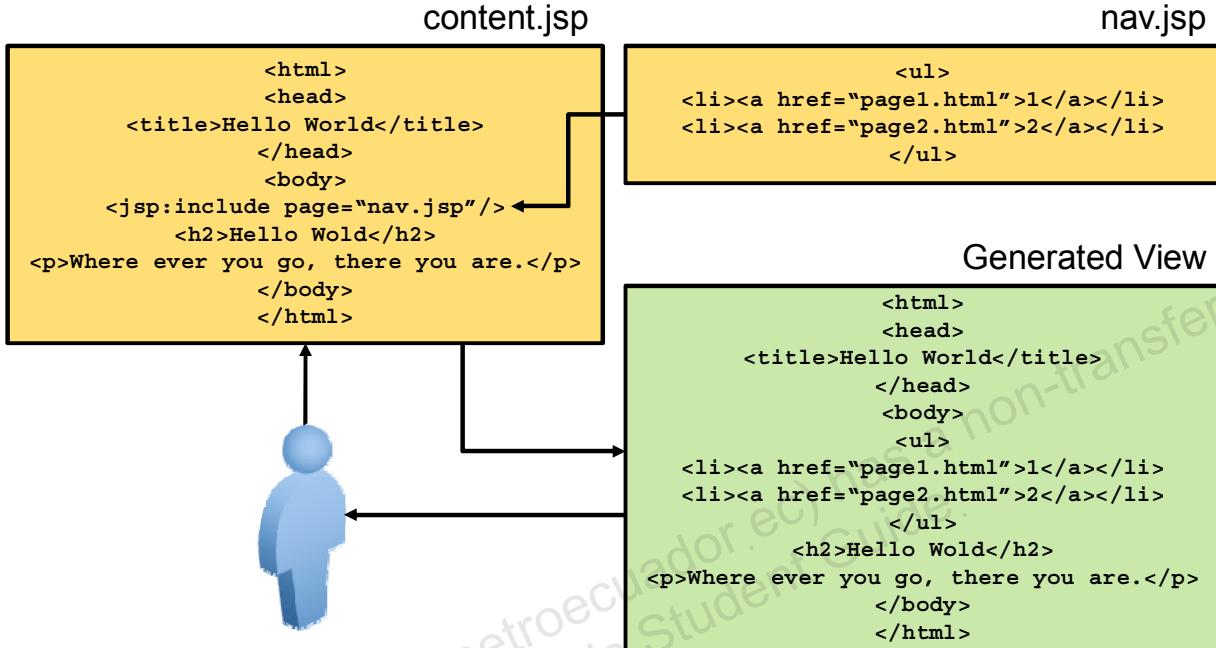
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

To create any sort of continuity on a web interface, the web designer and information architect should have consistent navigational elements on all webpages.

Deciding how this is going to be accomplished with as little redundancy and as much reuse as possible is part of the architectural concerns of the Web tier.

Inclusion Templating

XHTML Fragment Inclusion



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

In the past, it was common practice for the web developer to make use of an inclusion mechanism on each content page. The inclusion reference plugs the included reusable elements into the content page at runtime.

While the inclusion approach does offer some degree of reuse, it becomes increasingly brittle as a website grows in complexity and size. Imagine adding numerous inclusions to each content page. The content page then needs a great deal of layout-oriented code to manage the positioning of the included elements. Should the layout need to change, that change must be made to every content page.

Transclusion Templating

XHTML Fragment Transclusion

body.jsp

```
<h2>Hello Wold</h2>
<p>Where ever you go, there you
are.</p>
```

requested_page.jsp

```
<span template = "layout.jsp">
<param name="title" type="string"
value="Hello World"/>
<param name="body"
value="body.jsp"/>
</span>
```



layout.jsp

```
<html>
<head>
<title><span insert="title"/></title>
</head>
<body>
<ul>
<li><a href="page1.html">1</a></li>
<li><a href="page2.html">2</a></li>
</ul>
<span insert="body" />
</body>
</html>
```

Generated View

```
<html>
<head>
<title>Hello World</title>
</head>
<body>
<ul>
<li><a href="page1.html">1</a></li>
<li><a href="page2.html">2</a></li>
</ul>
<h2>Hello Wold</h2>
<p>Where ever you go, there you are.</p>
</body>
</html>
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Transclusion addresses some of the maintenance concerns of inclusion. Transclusion allows the markup associated with pieces of content to only contain enough markup to manage the presentation of the content itself. The content that the user is requesting is then inserted into a common template. The relationship between the content and the layout is managed in one of two ways. It is either directly referenced in the content, or pieced together in a lightweight view component for an even greater degree of loose coupling between the style and data concerns.

Transclusion scales well. Changes to the layout do not impact any pieces of content beyond the possible references to templates named in the content. Lightweight view components that bind the content to the layout can help achieve an additional level of loose coupling between the content and the layout, at the price of an additional component to manage.

View Compositing

- Aggregate reusable atomic subviews into a parent view.
- Portlets provide a standard representation of atomic subviews within different view contexts:
 - Minimized
 - Normal
 - Maximized
- Portal frameworks provide mechanisms to efficiently aggregate portlets from multiple sources.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The idea of compositing is represented in the core Java EE design pattern called the Composite View. This design pattern describes the placing of multiple atomic subviews on a single view or page. These atomic subviews are simply content fragments themselves. Layout templates that include a header, navigational bar, main content element, and footer are forms of composited views themselves.

Atomic subviews of a composite are normally as loosely coupled as possible, so that they can be composited into any given view. They do not even need to come from the same application or even server for that matter.

Composite views that span multiple data sources are often referred to as portals. Portal frameworks are feature rich templating mechanisms that assist in the layout and sourcing of their atomic subview components. Some portal frameworks use standard APIs for requesting specific styles of representations from their atomic subviews. For example, subview components that want to abide by the *Java Portlet Specification (JSR-168)* must provide different window states that present their content differently depending on whether the requested state is minimized, normal, or maximized. These window states are indicative of the size of the screen area within which the portlet is expected to be contained.

View Generation Frameworks

- Abstracted technology independent user interface toolkits
- Proprietary desktop style user interface toolkits
- Swing based view generation frameworks
- Usage may be trading rapid application development of a majority of the view components for:
 - Tight coupling with the technology
 - Inflexibility for extension
 - Complexity or lack of support for some aspect of your required view components



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The idea of view generation suggests having an abstracted representation of the targeted view format. In the case of the Web tier, this targeted format is most likely XHTML and perhaps some JavaScript to go along with it.

Certainly, the manner in which Java Server Pages (JSPs) are converted into servlets and executed to output XHTML is a form of view generation. However, JSPs require the insertion of code or tags or both into XHTML.

Most view generation frameworks have abstracted user interface toolkits that exist for the purpose of being used as the source of a process that generates a view in the target format. Some frameworks assist the multiserving strategy by attempting to target different view formats or representations from a shared interface representation in the semantics of the abstracted toolkit. The overall goal of most, though, is to try to hasten the development of view components.

Such frameworks, however, should be analyzed thoroughly before adoption, and not be considered substitutes for skilled Web designers. While some of the Java API oriented toolkits might appeal to Web developers, they tend to make good interface design challenging. In fact, many of these sort of frameworks can easily accommodate only 80 percent to 90 percent of the Web development goals. The last 10 percent to 20 percent is usually problematic, and the application ends up being married to the technology.

Previewability

- View components do not have to be deployed to a web server or web container for preview.
- Great SoC for web designers; lets them use What You See Is What You Get (WYSIWYG) XHTML editors.
- Tags and attributes that provide context at runtime are ignored when viewed locally within a browser.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

View components that can be viewed locally in a browser, without having to be served through either a Web server or an application server first, are said to have a high degree of previewability. Most browsers will only attempt to make sense of recognized XHTML elements that it discovers within the source, everything else can be assumed to be content, or just ignored.

It should not be expected that the Web designer or graphic artist can make sense of any non-XHTML or non-JavaScript source code, or any custom tags that might be embedded in the source of the view components. Nor should they. That is not part of their job description. Talented Web developers that are also fantastic Web designers are hard to find. So any arrangement that requires the Web designer and the Web developer to work on the same file can be frustrating for the Web designer. That experience is further frustrated when the act of unit testing a minor design modification requires a deployment step to a server process.

If you want to make the Web designer productive, you should let them use a What You See Is What You Get (WYSIWYG) style editor, and plug-in default values for the data elements that will be bound at runtime. This is just the sort of capability that a template engine gives you.

A template engine interprets entity placeholders in the XHTML at runtime, and inserts the appropriate content. The previewability of view components destined for interpretation by a template engine at runtime is lost if a Web browser finds any nonignorable programming oriented code in it. As such, JSPs and custom tag libraries are not good candidates for previewability.

Scripting Languages

- Interpreted for high previewability
- Can integrate with enterprise Java components
- Languages require separate JAR to run:
 - Groovy
 - JRuby
 - Jython
 - PHP implemented in Java



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Interpreted scripting languages, such as Perl and PHP Hypertext Preprocessor (PHP) have been popular for some time. They allow a high degree of previewability in that there is no compilation step before deployment, and provide immediate viewing of dynamic behavior through a Web server that supports the script handler.

Scripting languages have always been popular for smaller projects, due to the rapidity of development. That rapidity, however, is often accomplished through a sacrifice of good SoC, even though frameworks exist for most scripting languages to support good SoC.

The use of scripting languages for the Web tier of enterprise applications is starting to find mainstream acceptance. Most scripting languages have a bridge technology that allows communication with enterprise Java technology components. There are, however, scripting languages that run inside the JRE with greater degrees of integration with Java enterprise applications. Those languages include:

Session State Management

- Client needs to hold on to a session token ID so the server can map them to their server-side state.
- No guarantee that the user will permit the usage of cookies for session token ID storage.
- Consider embedding the session token ID into all URLs that link back to the server.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Web browsers are inherently stateless. No user information is persisted between requests. A Web developer can make use of client-side cookies for the storage and retrieval of a small set of simple name/value pairs, but there is no guarantee that the user will permit the use of cookies. What is more common, is for the user's stateful information to be persisted on the server side. That way, objects like collections and complex types can be easily handled.

However, for a user to be bound to their persisted state again on additional requests, some bit of shared data must exist on the client-side. That bit of information is referred to as a session token ID. It can either be stored in a cookie, or if cookies are disabled, embedded in all URLs that link back to the server. The embedding of session IDs in links should be the responsibility of the process that manages URL references to content.

Input Data Validation

- A good validation framework should provide:
 - A common set of instrumentable validators
 - The application of validators via configuration
 - The ability to provide customized validators
 - Local, remote, and web service-based service providers
 - Runtime JavaScript validator generation for the client tier
- Java EE 6 introduces Bean Validation.
- JSF includes built-in and custom validators.
- Client-side validation should be performed on the server as well.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Regardless of the extent of data validation taking place in the client tier, data validation must also take place on the server side. Ideally, a ubiquitous data validation mechanism cuts across the Web, business, and integration tiers as an application service.

Some client tier validators found in RIA Ajax frameworks can make XMLHttpRequest callbacks to the server at runtime for validation within the server. A great user interaction pattern that fits this feature quite nicely is the attempt to select a user handle in a Website registration form. The framework could validate the user's selected handle by invoking an XMLHttpRequest callback to the server side validator that checks to make sure that the handle the user wants is not already selected. The end user could be told to select another handle before they actually submitted the form.

Bean Validation affords a standard framework for validation, in which the same set of validations can be shared by all the layers of an application. Bean Validation offers a framework for validating Java classes written according to JavaBeans conventions. You use annotations to specify constraints on a JavaBean—you can annotate a JavaBean class, field, or property. You can also extend or override these constraints through XML descriptors. A validator class then validates each constraint. You specify which validator class to use for a given type of constraint.

Some client tier validators found in RIA Ajax frameworks can make XMLHttpRequest callbacks to the server at runtime for validation within the server. A great user interaction pattern that fits this feature quite nicely is the attempt to select a user handle in a website registration form. The framework could validate the user's selected handle by invoking an XMLHttpRequest callback to the server side validator that checks to make sure that the handle the user wants is not already selected. The end user could be told to select another handle before they actually submitted the form.

JSF has a number of Built-in Validations and also it comes with a Powerful Validation Framework for plugging-in Custom Validators.

Internationalization and Localization

The ten most popular languages spoken worldwide and the approximate number of primary or first language speakers for that language.

1. Mandarin Chinese: 882 million
2. Spanish: 325 million
3. English: 312-380 million
4. Arabic: 206-422 million
5. Hindi: 181 million
6. Portuguese: 178 million
7. Bengali: 173 million
8. Russian: 146 million
9. Japanese: 128 million
10. German: 96 million



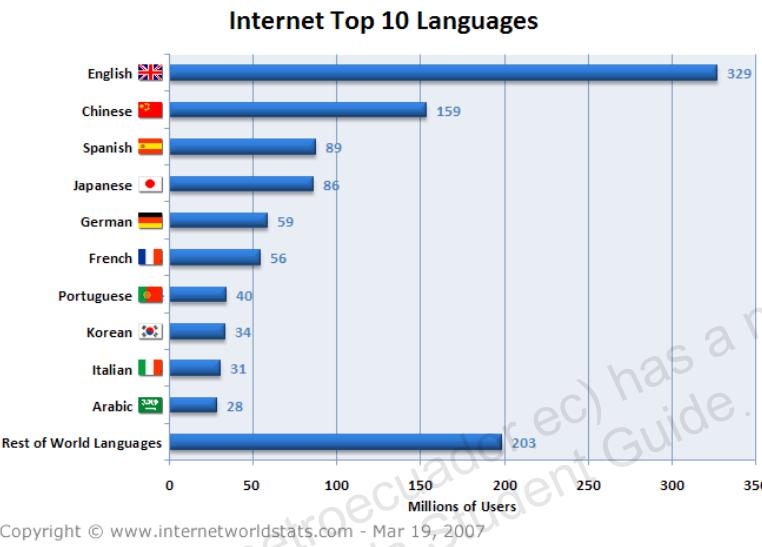
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The WWW is certainly a global phenomenon. People that have Internet access just about anywhere in the world have the potential of navigating to just about any website. That is a powerful thing for any company looking to expand their product and service offerings on a global level. Certainly, however, not everyone in the world is going to speak the same language. In some cases, even neighbors do not speak the same language.

There are over 6,900 languages spoken worldwide among approximately 6.5 billion persons on the planet. About 40 percent of them speak one of the top ten most spoken languages. The worldwide demographics for the top ten most spoken native languages approximately breaks down as shown in the following figure (rough estimates based on various sources as of 2010).

Languages Spoken by Internet Users

Among persons that actually have access to the Internet, the demographics look like the following figure



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Internationalization (i18n), as it applies to web applications, means that the web application supports multiple languages. Offering the support of another language can instantly broaden the potential market demographic, but doing so poses a challenge to information architects and Web developers.

It is more difficult to refactor an application for i18n support than to architect in advance for it. It has a profound effect on how content is handled on just about every tier of the application.

Larger Web sites that attempt to offer i18n support often end up just copying the content from the currently architected application to a new server environment and retrofitting it for the new language. This results in a completely new code base to manage. This might get an implementation out the door faster, but maintaining consistency with the master code base from the original site can be costly.

A better approach would be to manage the support of multiple languages from the same application code base. These sorts of websites can offer the user the choice of languages to select from on the main page.

Localization (l10n) is the process of automatically determining a users preferred language and providing it to them. This is a configuration feature of most browsers that ends up being sent in the Accept-Language header of every browser request.

Specifying the Preferred Language

```
GET / HTTP/1.1
Host: www.sun.com
Accept:text/xml,application/xml, ...
Accept-Language: de,de-de;en-us;en;
Accept-Charset: ISO-8859-1,utf-8;*
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1 ...
```

For an example of l10n support, modify your browser language preferences and visit <http://docs.sun.com>.

The only difference between basic i18n support and l10n support is the automated nature of the language selection. Not many persons fluently speak a second language. Therefore, it is not hard to imagine how a Mandarin speaker could overlook the English “click here for Mandarin” link on a webpage. You are less likely to lose readers if you give them the language they want on the first pass.

Content Management

Collect metadata on content such as:

- A unique document id
- A brief description of the content
- Keywords that might help locate the content
- The authors of the content
- The creation and last modified dates
- The location of the content



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A more scalable approach to managing this content is to collect metadata on each content element in a database. As Web sites grow in size, so does the content and the number of embedded links that refer to the content. If those links refer directly to the local file system, they have the potential of creating a brittle system. If the decision is made to reorganize content on the file system, those links will all have to be updated.

This way, the documents can be referenced using their unique ID. Such systems normally cache their references to content locations for performance. Extending this concept with document versioning can offer additional benefits. Many document repositories exist for the purpose of providing this sort of functionality in a robust manner.

The URL manager responsible for generating the document links in the generated view can also be used to embed additional GET attributes. It can also embed session IDs for users that have cookies turned off and need a stateful token to be bound to their server-side session.

Searchability

- Many users have come to rely on site searches to quickly find the content they want.
- Website re-architecting can result in the accidental removal of all links to a piece of content.
- A good search toolkit should provide for the indexing of file and database content in as flexible of a manner as possible.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Any website of reasonable size should give serious consideration to the presence of an instrumentable search facility. Many users have come to rely on site searches to quickly find the content they are looking for, in lieu of a game of hide and seek with the categorized link structure of the website. Website rearchitecting might result in the accidental removal of all links to a piece of content. In those circumstances, a search facility is a savior to the end user.

A good search toolkit should provide for the indexing of file and database content in as flexible of a manner as possible. Architects looking for a good search toolkit might want to consider the Lucene search toolkit.

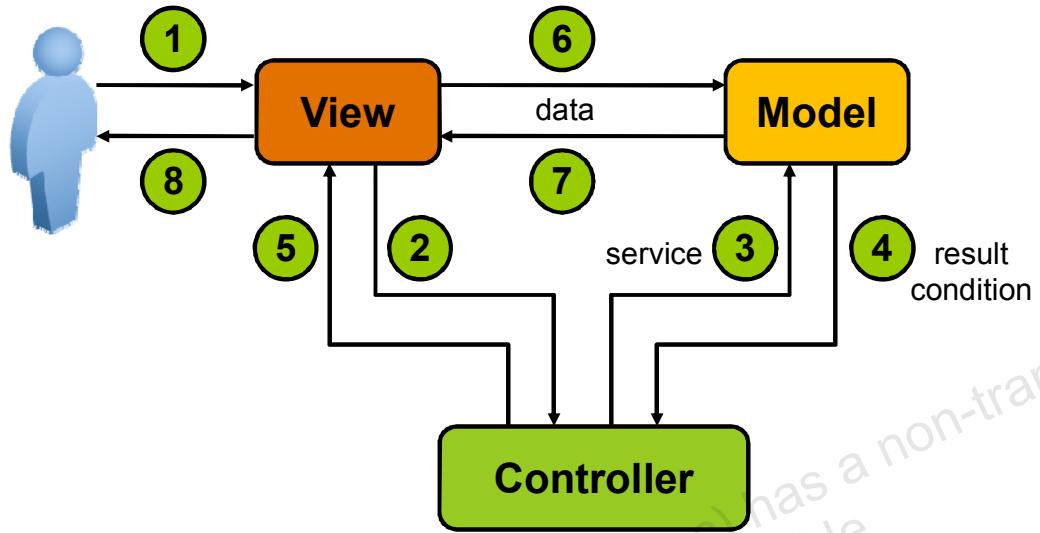
Control and Logic Concerns

The following topics discuss the Web tier concerns relating to the management of view components, data entities, web framework-related logic, and the delegation of requests for business logic to the business tier.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Model View Controller



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

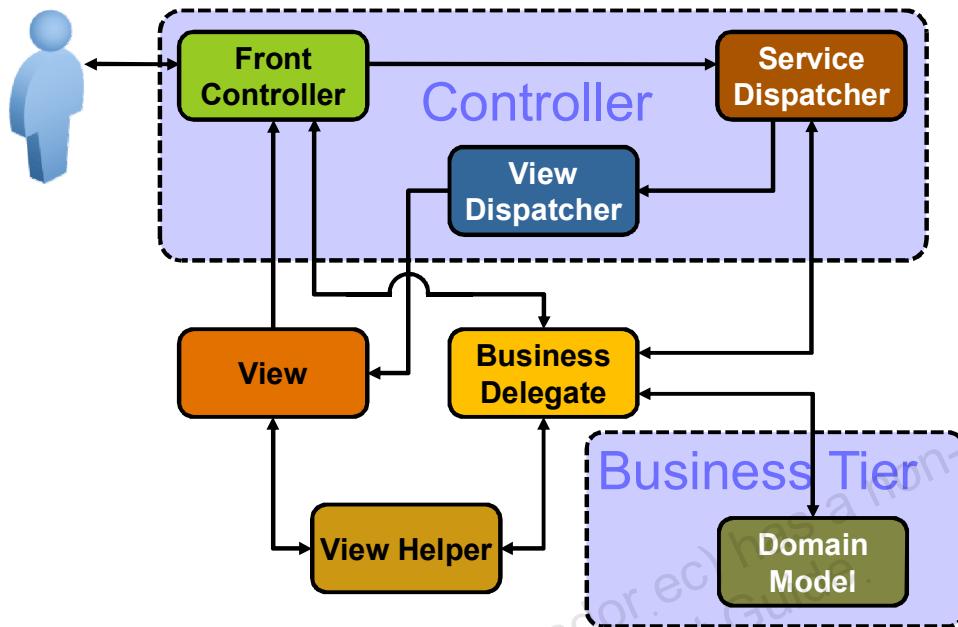
Within the MVC architecture, as it is typically used within the Web tier, all data and business logic are encapsulated inside the model. The original MVC had an event-based mechanism for the model to communicate state changes to the view. Because HTTP is a request/response-oriented communication, the view has to explicitly make a request of model state if it was not returned with the result condition.

The controller has two responsibilities:

- Dispatching of service requests to the model
- Dispatching of views to the end user

The components within the model might be somewhat coupled to the MVC framework. Because it is considered good practice to keep your business data and logic completely decoupled, what goes on in this model needs to be scrutinized.

Service-to-Worker Pattern



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

So as not to confuse the true meaning of a model, and to more accurately represent how state and behavior are managed in the Web tier, architects often refer to the Service-to-Worker pattern. The Service-to-Worker pattern leaves the idea of a model to the concerns of the business tier, where it belongs. Specifically, the Service to Worker pattern just applies to the state machine relationship between a service dispatcher and business delegates. However, the term is usually used within the context of an architecture that refers to the overall usage of the components depicted in the slide.

A front controller is the front door to the web application. It is usually implemented as a servlet or filter that all requests must come through. It may use business delegates directly for domain model operations that provide support for the web framework itself, or pass business logic requests to the service dispatcher for dispatching to the appropriate domain model components through business delegates.

The service dispatcher routes business logic requests to the appropriate business delegate. The business delegate calls the appropriate domain model component in the business tier. The business delegate then persists the value object returned from the business delegate to the appropriate scope (page, request, session, or application) and returns a result condition to the service dispatcher. If there are no further business logic operations to perform on the request, the final result condition is passed to the view dispatcher.

The view dispatcher decides what view component to dispatch to the end user based on the result condition for the users request. The view then accesses the value object in the appropriate scope and renders the final view for the user. Views can also use view helpers to help render pages and get access to business logic through business delegates for situations where service dispatching is not needed.

Filtering

Offers a high degree of decoupling and instrumentability to do things like:

- Check for authentication and authorization
- Log information about requests and responses
- Provide (de)compression services
- Modify HTTP Headers



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Can be a useful architectural mechanism to easily add behavioral concerns to a running process prior to, or immediately after, its invocation.

Web Flow

- A process flow of user interactions through a series of webpages, such as an interactive wizard
- Should be managed by a framework that provides a context for piecing together the user interactions into a workflow. For example: JSF navigation.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Web flow is the term used to describe a process flow of user interactions through a series of webpages in a wizard-style fashion. Frameworks that provide for web flow management build on top of the service-to-worker request dispatching pattern as robust state management mechanisms. They provide a managed context for piecing together the user interactions into a workflow. Some frameworks even offer GUIs to assist in the web flow definition process.

Lesson Agenda

- Responsibilities of the Web tier
- Separation of Concerns
- Comparing Web tier frameworks
- Providing security in the Web tier
- Scaling the Web tier

Popular Web Tier Frameworks



- | | |
|---|--|
| <ul style="list-style-type: none">• JBoss Seam• FreeMarker• Turbine• Struts• Wicket• Tapestry• Velocity | <ul style="list-style-type: none">• Cocoon• Spring Web Flow• JavaServer Faces• NextApp Echo 2• Google Web Toolkit• Stripes• Spring MVC |
|---|--|



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Selecting the appropriate Web tier framework might be the most daunting task that a Java Enterprise Application Architect can undertake. Few other architectural decisions have such a profound impact on the development process and maintainability of the code. The architectures are as diverse as the quantity of choices, and those choices are constantly in a state of flux. Frameworks occasionally fall by the wayside based on a lack of community support, relative to their complexity. Sometimes they change drastically between version numbers.

As of this writing, the frameworks noted in the slide were considered relatively popular, but should be considered with suspicion, as the landscape of such frameworks is constantly changing. View template engines, such as Velocity and FreeMarker, are not frameworks although for brevity they are typically listed with web tier frameworks.

Criteria for Framework Selection

- Previewability
- Testability
- Separation of Concerns
- Approach to transclusion templating
- Approach to state and session management
- Amount of configuration required
- Support for third party frameworks
- Community support
- Availability of expertise



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Unfortunately, there really is no one-size-fits-all framework. They are each good and bad at various tasks, and in many cases, multiple frameworks can be used at the same time for different purposes. Overall, an architect should choose frameworks based on their interpreted acceptability of the characteristics listed on the slide and continued below.

Additional criteria:

- Complexity and learning curve
- Level of component reuse afforded
- Ability to manage web flow
- Search engine friendliness
- Code coupling to framework
- Security
- Integration with the business tier

Request-Oriented Frameworks

- Make their service dispatching decisions based on the URL of the request sent into the framework
- Often up to the class acting on the request to observe and make sense of the user input
- Classes that the Web developer writes operate on a single behavioral concern
- Often a one-to-one mapping between a URL and a particular class
- Reuse can be difficult to achieve
- Supported by JAX-RS in Java EE6



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Page request (also known as action or operation) oriented frameworks make their service dispatching decisions based on the URL of the request sent into the framework. Often, these frameworks will look at a configuration file to determine what class is supposed to act on the request. It is often up to the class acting on the request to observe and make sense of the user input.

The classes that the web developer writes operate on a single behavioral concern. Therefore, there tends to be a one-to-one mapping between a URL and a particular class. Reuse can be difficult to achieve with this approach.

Component Oriented Frameworks

- HTML components on the interface are bound to particular classes.
- Markup within the HTML component is usually indicative of a method to be invoked in a class on the server.
- Tends to have more implied behavior, sometimes resulting in a zero configuration framework
- Tends to result in a higher degree of cohesion and less code, since multiple actions can be addressed by a single class



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Component-oriented frameworks tend to offer more interpretation services than action-oriented frameworks. In most cases, the developer never interacts with the Servlet API at all. HTML components on the interface are bound to particular classes, and markup within the HTML component is usually indicative of a method to be invoked in a class on the server. This can result in fewer classes having to be written, since there is often a greater degree of flexibility in the method binding and numerous methods providing behavioral concerns to a particular concern can be aggregated into a cohesive class. Many frameworks allow these classes to be decoupled from the framework itself.

Oracle Application Development Framework (ADF)

- Is an end-to-end Java EE framework that is extensible
- Utilizes and adds value to the Java EE platform
- Abstracts Java EE complexity
- Provides declarative and visual development
- Enables developers to focus on the application, not the low-level infrastructure
- Creates reusable and maintainable code
- Uses metadata, simplifying the basic task of wiring user interfaces to services
- Implements Java EE best practices and design patterns, including MVC



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Oracle ADF is an end-to-end application framework that builds on Java EE standards and open-source technologies to simplify and accelerate implementing Java EE applications. It is fully extensible and customizable by adding or modifying libraries.

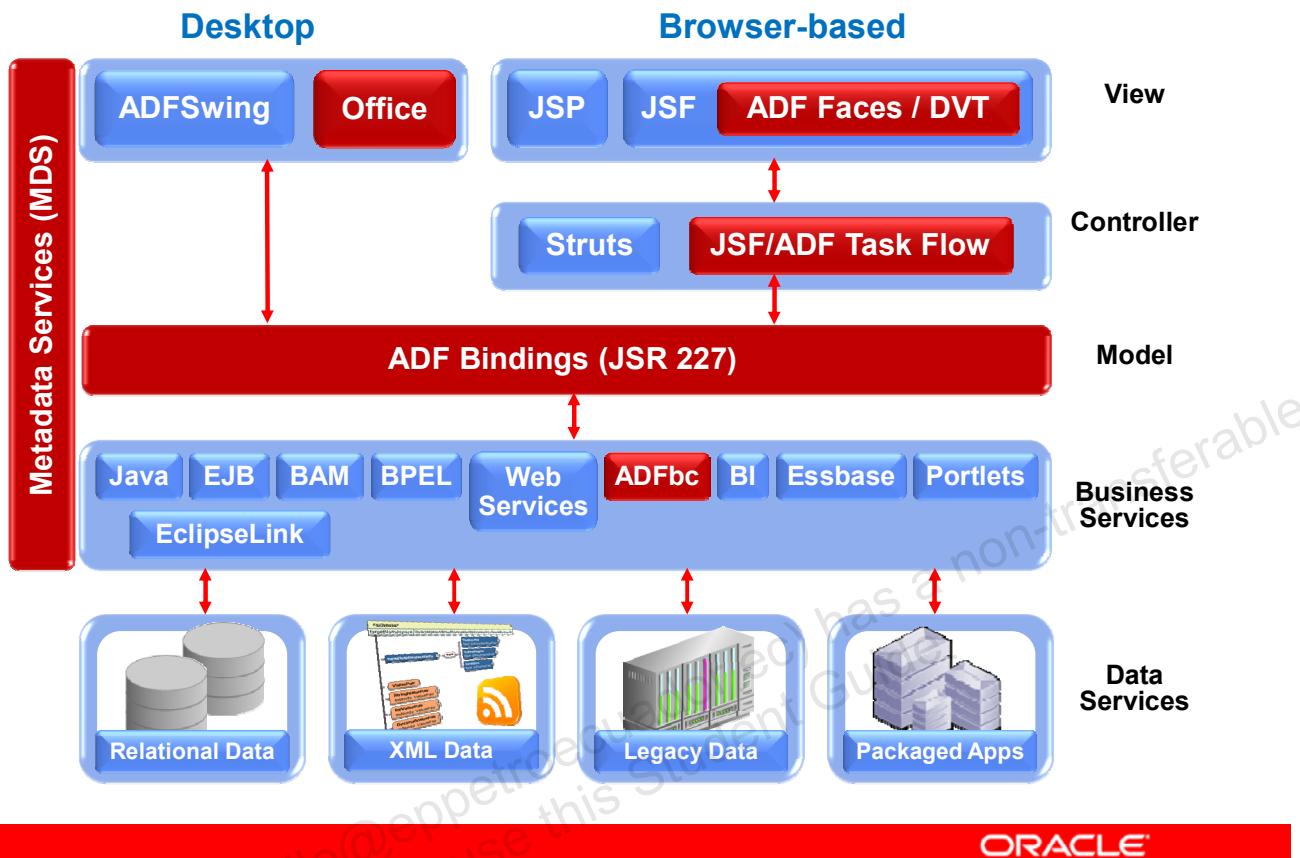
ADF simplifies the tasks of:

- Creating business services
- Designing user interfaces to access those services

Oracle ADF simplifies Java EE development by minimizing the need to write code that implements design patterns and the application infrastructure. These implementations are provided as part of the framework. Oracle ADF also provides a visual and declarative development experience that minimizes the need to write code and reduces the learning curve for 4GL developers.

Business services are implemented as metadata, enabling them to be bound to user interfaces in the same manner regardless of the technology employed in the underlying data model. The use of metadata also enables business rules for databound fields to be specified at the model layer, along with labels, validation, and tool tip properties.

Technology Choices for ADF BC Applications



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

- **Business Services:** ADF Business Components (ADF BC) provides the business services that are responsible for representing database tables and persisting user input values to the database.
- **Model:** ADF data binding is accomplished in compliance with JSR-227, a standard data binding and data access facility for Java EE that provides a standard for interactions between UI components and methods available on the business services. With this standard data binding, any Java UI rendering technology can declaratively bind to any business service.
- **Controller:** ADF Controller
- **View:** Java ServerFaces and ADF Faces; render kits enable the UI components defined by Java ServerFaces or ADF Faces to be rendered differently on different devices. For example, the UI can be rendered on a computer's browser or on a mobile device such as a cell phone or PDA.
- **Metadata Service:** ADF BC applications can also use Metadata Service (MDS) for customization and personalization. This is outside the scope of this course.

Competing Frameworks

- Spring Web Flow
- Apache Struts
- Apache Wicket
- Ruby on Rails
- Django (python)
- Grails
- PHP (Cake, Codeigniter, Zend)
- GWT (Google Web Toolkit)
- Adobe Flex



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

- **Spring Web Flow:** Spring Web Flow (SWF) is the subproject of the Spring Framework that serves as a Web application controller component focused on management of web application page flow.
- **Apache Struts:** The Jakarta Project's Struts framework, from Apache Software Organization is an open source framework for building web applications that integrate with standard technologies, such as Java Servlets, JavaBeans, and JavaServer Pages. Struts offers many benefits to the web application developer, including Model 2 implementation of Model-View-Controller (MVC) design patterns in JSP web applications. The MVC Model 2 paradigm applied to web applications lets you separate display code (for example, HTML and tag libraries) from flow control logic (action classes).
- **Apache Wicket:** Apache Wicket, commonly referred to as Wicket, is a lightweight component-based Web application framework for the Java programming language conceptually similar to JavaServer Faces and Tapestry.
- **Ruby on Rails:** Rails is a capable web application platform and has gained significant traction among Java EE and PHP programmers. That attraction makes a lot of sense when you consider the strengths of Rails. For one thing, it uses a strict Model-View-Controller architecture that any self-respecting design-patterns wonk would admire; this fact explains its attraction to JEE developers. Second, it's easy to build basic systems with Rails—which is attractive to PHP developers.

However, Rails has some pretty significant limitations from a database perspective. Rails makes a lot of assumptions about your database layout and application needs. For example, Rails assumes that all tables use a single, noncomposite, primary key. Composite primary keys are supported via a Rails plugin! In addition, Rails supports multiple databases and can coordinate transactions among them; Rails does not support two-phase commit.

- **Django (python):** Django, the Python-based web application framework, was originally designed to simplify development of database-driven, news-oriented Web applications. Since then, it has evolved into a full-featured web framework often chosen to simplify the development of complex, database-backed web applications.
- **Grails:** Grails uses the Groovy programming language; Groovy syntax is similar to Java's, but it also adopts some concepts from other languages. Like Java, Groovy generates bytecodes and runs on the Java platform, so it has the power of Java but also adds a few niceties and simplifications. Grails seeks to deliver high developer productivity by adopting convention over configuration, providing a stand-alone development environment and hiding much of the configuration details from the developer.
- **PHP (Cake, Codeigniter, Zend):** PHP is an open-source, interpreted, HTML-centric, server-side scripting language. PHP is especially suited for Web development and can be embedded into HTML pages. PHP is comparable to languages such as JSP (Java Server Pages) and Oracle's PSP (PL/SQL Server Pages). PHP is a recursive acronym for "PHP Hypertext Preprocessor."
- **GWT (Google Web Toolkit):** GWT proposes a different way to create Ajax applications. It uses Java as a single programming language for both the client and server sides. Is it the return of Java applets? Not at all: GWT provides a compiler that translates the Java code on the client side into JavaScript and DHTML. This solution greatly simplifies the technology stack from the programmer's point of view: You have to master only Java. The downside is that you have less control over the client-side code of your application because it's eventually generated by the GWT compiler.

The Java code for the client side of your application is subject to restrictions because JavaScript doesn't implement the entire object-oriented concepts and APIs available in Java. You can use only a subset of Java keywords and APIs (`java.lang` and `java.util`).

Adobe Flex: Flex is a free, open source framework for building expressive web applications that deploy consistently on all major browsers, desktops, and operating systems by leveraging the Adobe Flash Player and Adobe AIR runtimes.

Lesson Agenda

- Responsibilities of the Web tier
- Separation of Concerns
- Comparing Web tier frameworks
- Providing security in the Web tier
- Scaling the Web tier



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Providing Security in the Web Tier

- Authentication aims to validate the identity of the requester.
- Authentication schemes vary in their ability to assert the likelihood of an accurate authentication.
- The evidence that a requestor presents for validation is referred to as credentials.
- Those credentials could vary in potential verity from a simple statement of name all the way to a DNA sample.
- It is easy to spoof a name. However, faking a DNA test is much more difficult.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

There is a classic balance between security and functionality that an architect must find. Having no security is not much of an answer to most security related concerns, nor is waiting two weeks for the results of a DNA test to provide access to a resource.

Once a user has been reasonably authenticated, the user must be authorized before accessing resources. The process of determining the appropriateness of the user to access a resource is referred to as authorization. Authorization can be based on the user, a group to which the user belongs, or a role to which the user is assigned.

For Java enterprise applications, there are two layers of authorization. One within the Web tier, and another within the business tier.

Web tier resource authorization can be achieved by defining the access control within the Web.xml deployment descriptor. It is applied to resources based on URL matching patterns.

Assuming you are using Enterprise JavaBeans (EJB) within the business tier, the access to specific methods within EJBs can also be controlled using a deployment descriptor.

Authentication and Authorization

- Credentials for authentication:
- Username
- Password
- Encryption key
- Time based token value
- Voice recognition
- Iris or pupil scan
- Fingerprint scan

Web tier authorization:

- Based on requested URL
- Defined in the `web.xml` deployment descriptor

JAAS

- Java Authentication and Authorization Service (JAAS)
- Based on Pluggable Authentication Modules (PAM)
- Authentication or authorization modules can be swapped in and out without impacting the application code using JAAS
- No standardized approach regarding the injection of JAAS based modules into the security contexts of web or EJB containers



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A security framework called the Java Authentication and Authorization Service (JAAS) is part of the Java Standard Edition. It is based on the concept of Pluggable Authentication Modules (PAM) found in many UNIX®-based operating systems. JAAS defines an API for developing portable authentication and authorization modules that are domain-specific. For example, JAAS modules for access to RDBMS, LDAP, and RADIUS-based systems are not uncommon. Application components that require either authentication or authorization facilities can just code to the generic JAAS API. New authentication or authorization modules can be swapped in and out without impacting the application code using JAAS.

As of Java Enterprise Edition 5; however, there is no standardized approach regarding the injection of JAAS-based modules into the security contexts of Web or EJB containers. Most web and EJB containers have a proprietary means of hooking up JAAS modules into their security framework with varying degrees of success and ease.

The Java Authentication Service Provider Interface for Containers (JSR-196) is a specification designed to define the manner in which JAAS modules should be plugged into containers.

The ability to plug JAAS authentication modules into various framework security contexts is more extensive than the ability to plug-in JAAS authorization modules. Careful consideration should be given to the decision to implement a JAAS authorization solution as its potential lack of portability might not give it any strategic advantage over the ease of implementation of a vendor-specific solution.

Java Authorization Contract for Containers

It defines the semantics of policy providers that employ the permission classes to address the authorization requirements of Java EE, including the following:

- The definition of roles as named collections of permissions
- The granting to principals of permissions corresponding to roles
- The determination of whether a principal has been granted the permissions of a role (that is, `isCallerInRole`)
- The definition of identifier to role mappings that bind application embedded identifiers to application scoped role names



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

JACC (Java Authorization Contract for Containers) is part of the Java EE specification and defined by JSR 115. JACC defines an interface for pluggable authorization providers. Specifically, JACC is used to plug in the Java policy provider used by the container to perform Java EE caller access decisions. The Java policy provider performs Java policy decisions during application execution. This provides third parties with a mechanism to develop and plug in modules that are responsible for answering authorization decisions during Java EE application execution. The interfaces and rules used for developing JACC providers are defined in the JACC 1.0 specification.

Oracle Identity Management

Oracle + Sun Combination

Identity Administration	Access Management*	Directory Services
Identity Manager	Access Manager Adaptive Access Manager Enterprise Single Sign-On Identity Federation Entitlements Server	Directory Server EE Internet Directory Virtual Directory
Identity & Access Governance		
Identity Analytics		
Oracle Platform Security Services		
Operational Manageability		
Management Pack For Identity Management		

*Access Management includes Oracle OpenSSO STS and Oracle OpenSSO Fedlet



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

OIF:

Federation by enabling identity providers and service providers to connect seamlessly
Create trust relationships between partners and agencies by connecting users seamlessly and securely - solution for secure identity information exchange between partners.

Multi-protocol federation server

It ensures the interoperability to securely share identities across vendors, customers, and business partners

Cross-domain SSO

eSSO:

Oracle Enterprise Single Sign-On Anywhere - Simplifies Oracle Enterprise Single Sign-On Sign-On deployments to client desktops

Oracle Enterprise Single Sign-On Logon Manager – enables individuals to securely use a single login credential to all web-based, client-server and legacy applications

Oracle Enterprise Single Sign-On Password Reset - Reduce helpdesk costs and improve user experience by enabling strong password management for Microsoft Windows through secure, flexible, self-service interfaces

- Oracle Enterprise Single Sign-On Authentication Manager - Enforce security policies and ensure regulatory compliance by allowing organizations to use a combination of tokens, smart cards, biometrics and passwords for strong authentication throughout the enterprise
- Oracle Enterprise Single Sign-On Provisioning Gateway - Improve operational efficiencies by enabling organizations to directly distribute single log-in credentials to Oracle Enterprise Single Sign-On Sign-On Manager based on provisioning instructions from Oracle Identity Manager;
- Oracle Enterprise Single Sign-On Kiosk Manager - Enhance user productivity and strengthen enterprise security by allowing users to securely access enterprise applications even at multi-user kiosks and distributed workstations.

OIA:

- Maintaining a centralized identity and access warehouse, optimized for complex analytical, simulation and governance needs (attestation, certification and auditing)
- Providing Persona-centric compliance monitoring in out-of-box compliance dashboards and reports to auditors, business users, compliance officers and IT managers
- Providing a complete view of access-related data that includes the user's access; the "why and how" of that access and if the access violates policies
- Automate the certification process and removing inappropriate access
- Providing evidence that access is compliant to established audit and Segregation of Duty policies
- Enabling changes in access based on changes in users'
- Providing a complete set of role change management, lifecycle management and role change impact analysis tools

OAAM:

- Real-time fraud prevention, multifactor authentication and unique authentication strengthening.
- Oracle Adaptive Access Manager consists of two primary components:
- Adaptive Strong Authenticator provides multifactor authentication and protection mechanisms for sensitive information such as passwords, PINs, security questions, account numbers and other credentials.
- Adaptive Risk Manager provides real-time and offline risk analysis and proactive actions to prevent fraud at critical login and transaction checkpoints. Adaptive Risk Manager examines and profiles a large number of contextual data points to dynamically determine the level of risk during each unique login and transaction attempt.

Security Token Service (STS):

- Simplifies the orchestration of standards-based and proprietary tokens between web services clients and providers enabling businesses to abstract security from web services.
- Solution for abstracting web services security and handling token issuance, validation and translation via WS-Trust. Supports both std. Based and proprietary tokens.
- Also provides means to propagate identity and security info across infrastructure tier for example. By converting a web sso token issued for enterprise portal to SAML token that is consumed by applications or webservices.

OpenSSO Fedlet:

- Is a service provider implementation of SAML 2.0 SSO protocol.
- A lightweight way for service providers to quickly federate with an identity provider
- An 8.5 MB package that identity providers give to service providers (SPs) to enable them to federate back to a company without the need for any additional federation products.
- To become federation enabled, the service provider simply adds the Oracle OpenSSO Fedlet to their application and deploy the application. No configuration is required and it works with both Java and .NET applications. With Fedlet, SPs can consume identity assertion and receive user attributes from OIF.

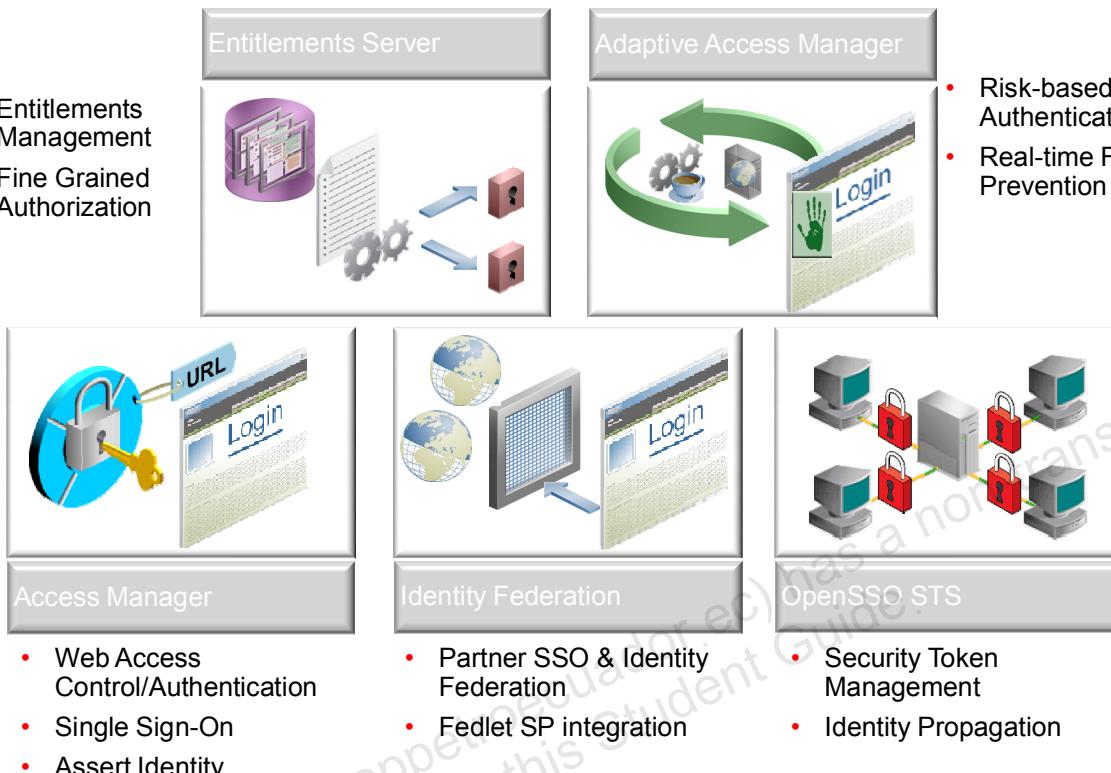
OES:

- Management of fine grained authorization policies and provides a standardized enforcement mechanism as an alternative to embedding one-off security within the application.

OPSS:

- OPSS provides an abstraction layer in the form of standards-based application programming interfaces (APIs) that insulate developers from security and identity management implementation details. With OPSS, developers do not need to know the details of cryptographic key management or interfaces with user repositories and other identity management infrastructures. By leveraging OPSS, in-house developed applications, third-party applications, and integrated applications all benefit from the same uniform security, identity management, and audit services across the enterprise.
- It is in a nut shell a standards-based, portable, integrated, enterprise-grade security framework for Java Standard Edition (Java SE) and Java Enterprise Edition (Java EE) applications.

Oracle Access Management Suite Plus



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Oracle Access Manager:

OAM 11gR1 is a Web SSO solution that is primarily focused on controlling access to resources accessible via HTTP that is resources identified by a URL.

OAM primarily perform three things:

- Provide Authentication services
- Manage sessions and SSO for web tier
- Assert Identities to application tier

OAM consists of two components – agents and OAM server.

Protected Resources are the applications that are protected by OAM11gR1 (also referred to as partner applications). Access to these resources is subject to the access control policies in OAM11gR1 and is enforced by enforcement agents that are deployed in the access path of the protected resource (for example, Web Agents deployed in the Web Server, J2EE agents deployed in the Application Server).

Agents are entities that control access to protected applications based on security policies. Agents present in the resource access path intercept every resource access request to enforce the security policy that protects the resources. Since Agents have to integrate with the protected

application environment, the following Agents are supported in OAM 11gR1:

OAM 10g/11g Webgates

Oracle Single Sign On (OSO) 10g Apache Module (mod_osso)

OAM 10g Access Gates

Protected application must be protected by a WebGate or mod_osso instance that is registered with Oracle Access Manager as a policy-enforcement agent. The policy- enforcement agent acts as a filter for HTTP requests. Oracle Access Manager enables administrators to define authentication and authorization policies.

OAM Server – This is the server side component that provides the core runtime access management services. OAM Server is a J2EE application that provides various IdP (Identity Provider) services. The OAM Server provides SSO, Authentication and Authorization services (coarse-level).

Oracle Entitlement Server:

- Provide Authorization services
 - Fine-grained application and data-level security
 - Manage entitlements, grants, policies
- Oracle Adaptive Access Manager:
 - Provide Fraud Detection services to access suite
 - Strengthen authentication security
 - Enable transaction security
- Oracle Identity Federation:
 - Provide cross domain SSO
 - Manage trust across partners
 - Enable interoperability across providers

Note: Oracle eSSO is not part of OAM suite plus. It is sold as a stand-alone product.

Salient Features

Oracle Access Manager provides:

- Authentication service for web-based applications
- Single Sign On access for applications
- Identity Assertion service
- Session Management
- Coarse level authorization protection



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Single Sign on access for applications:

Single sign-on is a process that gives users the ability to access multiple protected resources (webpages and applications) with a single authentication. Oracle Access Manager 11g provides single sign-on (SSO) through a common SSO Engine that provides consistent service across multiple protocols. Oracle Access Manager 11g is the Oracle Fusion Middleware 11g single sign-on solution. Single sign-on (SSO) enables users, and groups of users, to access multiple applications after a single sign-on and authentication. SSO eliminates multiple sign-on requests.

Oracle Access Manager 11g is a Java-based enterprise-level security application that provides restricted access to confidential information and centralized authentication and authorization services. All existing access technologies in the Oracle Identity Management stack converge in Oracle Access Manager 11g, including Oracle Access Manager 10g, OracleAS single sign-on, and other products. To summarize:

Oracle Access Manager 11g SSO provides authentication and authorization services to OSSO and OAM Agent-protected resources.

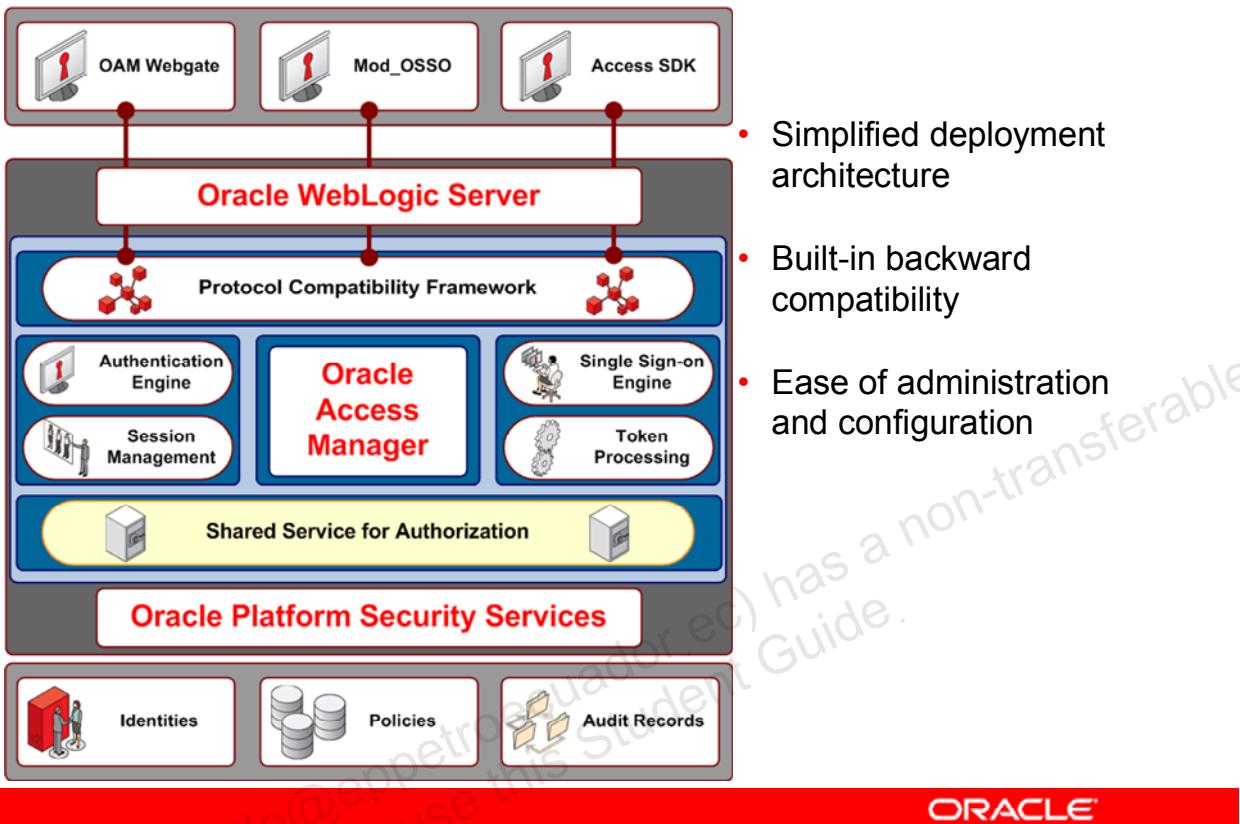
OracleAS single sign-on (OSSO) 10g provides authentication to mod_osso-protected resources.

Note: The mod_osso module is an Oracle HTTP Server module that provides authentication to OracleAS applications (such as Forms, Reports, Discoverer, Portal). This module resides on the Oracle HTTP Server that enables applications protected by OracleAS Single Sign-On to accept HTTP headers in lieu of a user name and password once the user has logged into the OracleAS Single Sign-On server. The values for these headers are stored in a mod_osso cookie.

The goal for OAM 11gR1 is to provide a single authoritative source for all authentication and authorization services. The core service provided is the checking of valid session tokens, the requesting of credentials if the session token is invalid or missing, and the issuing of session tokens, intercepting resource requests and evaluating access control policies to control access to resources.

OAM provides Single Sign On, thus preventing the user from re-logging in every time after authenticating once. It accomplishes this by managing the user session life cycle, which also involves facilitating global logout by orchestrating logout across all relying parties in the valid user session. OAM also ensures that resource access by users is authorized subject to the specified authorization policy.

OAM 11g Architecture



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Shared Services

The shared services consists of infrastructure components in OAM 11g such as Token Processing Engine, SSO Engine, AuthN Engine and Session Management Engine.

The two primary types of actors in the OAM architecture are the policy servers (OAM Servers) and OAM policy enforcement agents (WebGates or AccessGates). In the security world, Agents represent the policy enforcement point (PEP), while OAM Servers represent the policy decision point (PDP).

The Agent plays the role of a gatekeeper to secure resources such as http-based applications and manage all interactions with the user who is trying to access that resource. This is accomplished according to access control policies maintained on the policy server (OAM Server).

The role of the OAM Server is to provide policy, identity, and session services to the Agent to properly secure application resources, authenticate and authorize users, and manage user sessions.

Lesson Agenda

- Responsibilities of the Web tier
- Separation of Concerns
- Comparing Web tier frameworks
- Providing security in the Web tier
- Scaling the Web tier



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Web Server Clustering

- Usually the first tier to get clustered
- How do you accommodate load spikes?
- Coding for clustering:
 - call `setAttribute` after setting user session state.
 - Do not put singletons in the Web tier.
 - Session objects must implement `Serializable`



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The Web tier typically manages the user sessions. Therefore, clustering of the Web tier means that user session information needs to be available to any server in the cluster to which the user might be directed by a load balancer.

Depending on the audience and the expected usage of your web application, the system resources of the Web tier might need to be flexible and capable of responding to brief periods of heavy load.

The manner in which Web developers interact with objects in a user's session is of great importance when considering clustering of the Web tier. Most session management facilities will propagate the sharing or persistence of session information by placing an interceptor on the `setAttribute` method of an `HttpSession` object. This means that the only way that session information is available to other servers in a cluster is if this method is called. Therefore, web developers must ensure that this method is invoked even after having manipulated the state of objects within a session by reference. Peer code reviews should look rigorously for such oversight.

Oracle Coherence

- In-memory data grid solution
- Provides:
 - Caching
 - Analytics support
 - Transaction and event processing
- Fast access to frequently used data
- computation can be done in parallel
- Java runtime solution provides deterministic garbage collection with worst case, one millisecond latency
- Adaptive application infrastructure for assured QoS
- Application server-agnostic
- Works with .Net and C++



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

- **Data grid capabilities:** With capacity on demand, Oracle Coherence Suite can linearly scale out middleware infrastructure from a few to thousands of servers. Through Oracle Coherence in-memory data grid solution, it provides fast access to frequently used data. Leveraging this grid capability, computation can be done in parallel, further improving application performance.
- **Java runtime capabilities:** Oracle Coherence Suite's Java runtime solution provides deterministic garbage collection with extremely low latency, further improving the predictability and speed of performance.
- **Application management capabilities:** Automated application resource management capabilities provide an adaptive application infrastructure for assured Quality of Service (QoS). With a rich policy-based framework, resources are automatically provisioned in real time, improving resource utilization while reducing administration costs. The management layer provides rich and comprehensive management capabilities.

Singletons in the Web Tier

- Some external resources need a single point of management for stability.
- For example, having write access to a nonclustered file system could be dangerous if each server in a Web tier cluster had such access.
- Therefore, you might want to move this sort of code out of the Web tier, if possible.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The potential maintenance issues associated with the management of replicated message queues in a cluster encourages most architects to move such processes to the business tier, and handle load by scaling vertically, rather than horizontally.

The general rule of thumb is, scale processes that you want to keep as singletons vertically, instead of horizontally. Because the systemic concerns of the Web tier are best handled by scaling horizontally, you should avoid the use of enterprise application-wide singletons in the Web tier.

Accommodating Load Spikes

- Deciding on the quantity of servers in a farm is an important architectural concern, and can actually require the input of a business analyst.
- The reason for the business analyst is that the cost of hardware in the server farm needs to be weighed against potential lost revenue as a result of cluster overload under a Slashdot Effect (see note).
- These massive spikes in server load can be overly expensive to accommodate, given the 70 percent and 90 percent utilization rules discussed previously.
- Physical assets can be repurposed if severely underutilized.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

If you are selling household goods, you might want to accommodate being slashdotted by a favorable recommendation on the Oprah Show, or risk the loss of potential revenue. If your website is purely informational, you want to be less accommodating to such spikes.

Slashdot.org is a news website geared toward technologists. Its readership is vast. Therefore, the posting of website links on Slashdot has been known to significantly overload the web server(s) to which the link points. Any major media outlet can cause a similar effect.

A Service-Level Agreement (SLA) at a large data center that hosts numerous websites might be able to accommodate a Slashdot effect by automatically, and quickly, deploying new servers into the cluster. When the traffic dies down, servers can be undeployed and made available for any other applications. This might be a more cost effective approach than purchasing rarely used hardware, because you are only buying time on these additional servers for a short period of time.

Session Persistence

- Shared memory creates a great deal of network traffic.
- Most Web containers can persist sessions to a filesystem or RDBMS.
- Storing session information in an RDBMS may make the information searchable.
- All session objects must implement `Serializable`.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

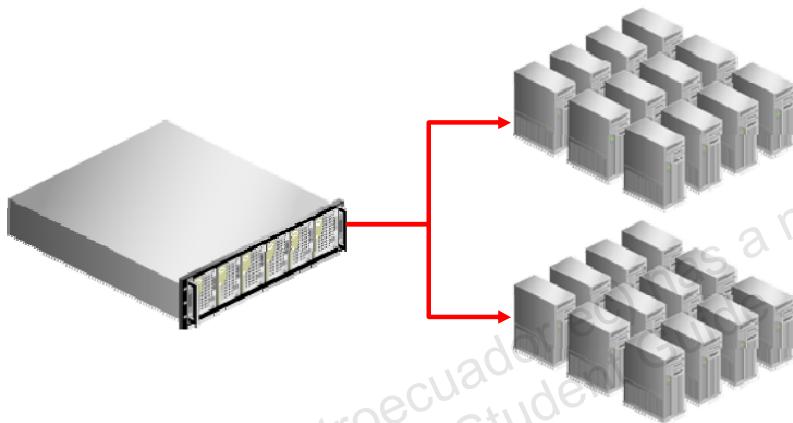
Using a network-mounted file system can be faster, but assurances must be made that the file system has an effective distributed locking mechanism. These sorts of file systems are sometimes referred to as clusterable file systems.

The database approach is a more robust session persistence strategy. Customizing the manner in which session information is stored in an RDBMS might make the information searchable, and provide a valuable metric gathering mechanism from a marketing perspective.

To use session persistence, all objects in the session must implement `java.io.Serializable`.

Load Balancing

- A load balancer should have session stickiness, or server affinity.
- A load balancer should have a hot standby.
- What happens if the load balancer fails?



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The load balancer is the gateway to the cluster. As such, it should be highly available and have some degree of intelligence. At the bare minimum, a load balancer should have session and the ability to be aware of the health and availability of servers in the cluster.

A common way for the load balancer to know about the state of servers in a farm is for them to send regular heartbeats to the load balancer. The load balancer keeps track of the available servers, and quickly removes servers from a list of available servers if it stops receiving heartbeats.

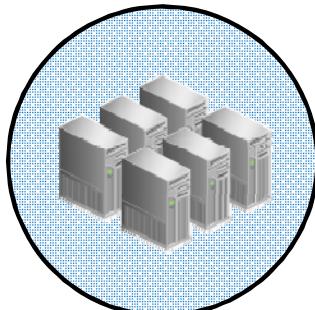
Session stickiness is the process of assigning a request to a particular server and sending the user to the same server on subsequent requests. This process is usually keyed off of the requesting IP address and port number. Any other strategy, such as one that keys off of the session ID, might not support Secured Socket Layer (SSL) connections for encrypted traffic.

If an assigned server is removed from the pool of available servers, the user is reassigned to the next available server. When that server does not find the user's session information in its memory, it tries to load it from the session persistence mechanism.

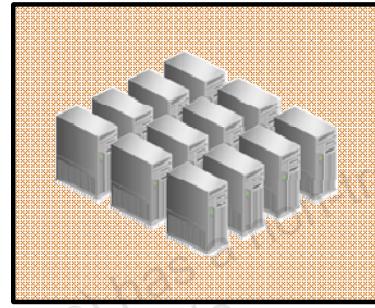
The actual strategy for assignment of servers within the load balancer could be a simple round robin. A better approach would be a weighted one that looks for resource load levels in the heartbeats from servers in the cluster and assigns new requests based on a least load strategy.

Single System Images

- Operating system spans across multiple physical machines to represent itself as a single logical box
- Java Virtual Machine spans across multiple physical machines to represent itself as a single JVM



OS



JVM

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

You might have inherited an application that was poorly architected for scalability. For example, perhaps the concerns of the application are so tightly coupled that making it clusterable requires a complete rewrite. Just because it does not cluster, however, does not mean it does not scale. Such a system could be deployed to a single system image (SSI).

A single system image is the representation of a single logical machine that is spread across many physical servers. In practice, this can happen either at the operating system or Java Virtual Machine level.

One such SSI implementation for the Linux operating system is OpenSSI [<http://openssi.org>]. OpenSSI manages the memory, process, and file sharing, among many other concerns, across the cluster in a dynamic fashion. A single JVM could be installed across the cluster and managed by the SSI.

An example of a cross cutting SSI for the JVM is Open TerraCotta [<http://terracotta.org>]. With Open TerraCotta, agents are installed on each server in the cluster. These agents unite to form a single logical JVM.

Great care should be taken with the selection of third party software. Restrictive licensing models can prevent the usage of clusters from either a legal or licensing cost perspective.

Additional Resources

- Xiaoying Kong, Li Liu, David Lowe, *Separation of Concerns: a Web Application Architecture Framework* [<http://jodi.tamu.edu/Articles/v06/i02/Kong/Kong-final.pdf>].
- Nimphius, Frank, *Oracle Fusion Developer Guide: Building Rich Internet Applications with Oracle ADF Business Components and Oracle ADF Faces*. Osborne ORACLE Press Series, 2009.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

David Czarnecki, *Java Internationalization*. O'Reilly Media, 2001

Erik Hatcher, *Lucene in Action*. Manning Publications, 2004.

Ka, lok Tong, *Enjoying Web Development with Tapestry*. Lulu.com, second edition, 2006.

Karhik Gurumurthy, Pro Wicket. Apress, 2006.

Seth Ladd, *Expert Spring MVC and Web Flow*. Apress 2006.

David Geary, Cay Horstmann, *Core JavaServer Faces*. Prentice Hall PTR, third edition, 2010.

Quiz

The web tier bridges the gap between the user interface and the domain model. What core responsibilities does the web tier have?

- a. Manage user session state
- b. Insert and delete user data into database tables
- c. Act as go between with the business tier
- d. Display a progress meter on a web browser



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: a, c

Quiz

The primary goal of Separation of Concerns is to:

- a. Define a loose coupling between style, data and logic
- b. Define how to implement an MVC pattern
- c. Define how to divide business logic from view logic
- d. Define functional requirements for the web tier

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: a

Quiz

Given a company profile that prevents intranet users from using cookies, what is the best choice to maintain session identity?

- a. Use a database table
- b. Use a Java Applet to maintain a connection to the database
- c. Embed session token ID in URLs that link to the server
- d. Session management may not be possible



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: c

Quiz

When considering scaling for the web tier, which of the following are good practices to follow? (Choose all that apply.)

- a. Move singletons to the business tier
- b. Persist session state in an RDBMS
- c. Use RESTful web services whenever possible
- d. Use a load balancer



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: a, b, d

Summary

In this lesson, you should have learned how to:

- Describe the roles involved with the development of the Web tier
- Describe the Separation of Concerns
- Describe strategies for implementing the presentation concerns of the Web tier
- Describe strategies for implementing the data concerns of the Web tier
- Describe strategies for managing the presentation, data, and logic-related concerns of the Web tier



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Summary

- Describe the advantages and disadvantages of request- and component-oriented Web-tier frameworks
- Describe strategies for implementing authentication and authorization in the Web tier
- Address the concerns of scaling web applications



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 9 Overview: Architecting for the Web Tier

This practice covers the following topics:

- Modify your web architecture to support your new UI choices from the lesson titled “Developing an Architecture for the Client Tier.”

Unauthorized reproduction or distribution prohibited. Copyright© 2015, Oracle and/or its affiliates.

Iveth Tello (iveth.tello@eppetroecuador.ec) has a non-transferable
license to use this Student Guide.

10

Developing an Architecture for the Business Tier

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the value in using enterprise application container services
- Describe the architectural options for implementing domain-model services
- Describe the architectural options for implementing domain-model entities
- Distribute domain-model components
- Describe the best practices for exception handling and logging



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Business tier technologies
- Architecting the domain model
- Development best practices

What is an Enterprise Bean?

- 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.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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. Provided that they use the standard APIs, these applications can run on any compliant Java EE server.

Enterprise Bean Types

Session: Performs a task for a client; optionally, may implement a web service

Message-driven: Acts as a listener for a particular messaging type, such as the Java Message Service API

Accessing Enterprise Beans

- Clients access enterprise beans either through a no-interface view or through a business interface.
- The business interface or no-interface view defines the client's view of an enterprise bean.
- Well-designed interfaces and no-interface views simplify the development and maintenance of Java EE applications.
- Clients obtain references to EJBs through dependency injection (preferred) or JNDI (outside Java EE server).
- In Java EE 6, EJBs can be invoked asynchronously.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A no-interface view of an enterprise bean exposes the public methods of the enterprise bean implementation class to clients. Clients using the no-interface view of an enterprise bean may invoke any public methods provided by the enterprise bean implementation class or any superclasses of the implementation class. A business interface is a standard Java programming language interface that contains the business methods of the enterprise bean.

A client can access a session bean only through the methods defined in the bean's business interface or through the public methods of an enterprise bean that has a no-interface view.

All other aspects of the enterprise bean (method implementations and deployment settings) are hidden from the client.

Not only do clean interfaces and no-interface views shield the clients from any complexities in the EJB tier, but they also allow the enterprise beans to change internally without affecting the clients. For example, if you change the implementation of a session bean business method, you won't have to alter the client code. But if you were to change the method definitions in the interfaces, you might have to modify the client code as well. Therefore, it is important that you design the interfaces and no-interface views carefully to isolate your clients from possible changes in the enterprise beans.

Session beans can have more than one business interface. Session beans should, but are not required to, implement their business interface or interfaces.

Using Enterprise Beans in Clients

The client of an enterprise bean obtains a reference to an instance of an enterprise bean through either dependency injection, using Java programming language annotations, or JNDI lookup, using the Java Naming and Directory Interface syntax to find the enterprise bean instance.

Dependency injection is the simplest way of obtaining an enterprise bean reference. Clients that run within a Java EE server-managed environment, JavaServer Faces web applications, JAX-RS web services, other enterprise beans, or Java EE application clients, support dependency injection using the javax.ejb.EJB annotation.

Applications that run outside a Java EE server-managed environment, such as Java SE applications, must perform an explicit lookup. JNDI supports a global syntax for identifying Java EE components to simplify this explicit lookup.

Asynchronous Session Bean Invocation

One of the powerful features introduced in EJB 3.1 is the ability to invoke session bean methods asynchronously. For an asynchronous invocation, control returns to the client before the container dispatches the invocation to an instance of the bean. This allows the client to continue processing in parallel while the session bean method performs its operations.

Deciding on Remote or Local Access

Whether to allow local or remote access depends on the following factors.

- Tight or loose coupling of related beans
- Type of client
- Component distribution
- Performance



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

When you design a Java EE application, one of the first decisions you make is the type of client access allowed by the enterprise beans: remote, local, or web service.

Whether to allow local or remote access depends on the following factors.

Tight or loose coupling of related beans: Tightly coupled beans depend on one another. For example, if a session bean that processes sales orders calls a session bean that emails a confirmation message to the customer, these beans are tightly coupled. Tightly coupled beans are good candidates for local access. Because they fit together as a logical unit, they typically call each other often and would benefit from the increased performance that is possible with local access.

Type of client: If an enterprise bean is accessed by application clients, it should allow remote access. In a production environment, these clients almost always run on machines other than those on which the application server is running. If an enterprise bean's clients are web components or other enterprise beans, the type of access depends on how you want to distribute your components.

Component distribution: Java EE applications are scalable because their server-side components can be distributed across multiple machines. In a distributed application, for example, the server that the web components run on may not be the one on which the enterprise beans they access are deployed. In this distributed scenario, the enterprise beans should allow remote access.

Performance: Owing to such factors as network latency, remote calls may be slower than local calls. On the other hand, if you distribute components among different servers, you may improve the application's overall performance. Both of these statements are generalizations; performance can vary in different operational environments. Nevertheless, you should keep in mind how your application design might affect performance.

If you aren't sure which type of access an enterprise bean should have, choose remote access. This decision gives you more flexibility. In the future, you can distribute your components to accommodate the growing demands on your application.

Although it is uncommon, it is possible for an enterprise bean to allow both remote and local access.

Local Clients

A local client has these characteristics.

- It must run in the same application as the enterprise bean it accesses.
- It can be a web component or another enterprise bean.
- To the local client, the location of the enterprise bean it accesses is not transparent.
- The no-interface view of an enterprise bean is a local view. The public methods of the enterprise bean implementation class are exposed to local clients that access the no-interface view of the enterprise bean. Enterprise beans that use the no-interface view do not implement a business interface. The local business interface defines the bean's business and lifecycle methods

Accessing Local Enterprise Beans Using the No-Interface or Business interface View

Client access to an enterprise bean that exposes a local, no-interface view or business interface is accomplished through either dependency injection or JNDI lookup.

Remote Clients

A remote client of an enterprise bean has the following traits:

- It can run on a different machine and a different JVM from the enterprise bean it accesses. (It is not required to run on a different JVM.)
- It can be a web component, an application client, or another enterprise bean.
- To a remote client, the location of the enterprise bean is transparent.
- The enterprise bean must implement a business interface. That is, remote clients may not access an enterprise bean through a no-interface view.

Web Service Clients

A web service client can access a Java EE application in two ways. First, the client can access a web service created with JAX-WS. Second, a web service client can invoke the business methods of a stateless session bean. Message beans cannot be accessed by web service clients. Provided that it uses the correct protocols (SOAP, HTTP, WSDL), any web service client can access a stateless session bean, whether or not the client is written in the Java programming language. The client doesn't even "know" what technology implements the service: stateless session bean, JAX-WS, or some other technology. In addition, enterprise beans and web components can be clients of web services. This flexibility enables you to integrate Java EE applications with web services. A web service client accesses a stateless session bean through the bean's web service endpoint implementation class. By default, all public methods in the bean class are accessible to web service clients.

What Is a Session Bean?

- A session bean encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views.
- To access an application that is deployed on the server, the client invokes the session bean's methods.
- The session bean performs work for its client, shielding it from complexity by executing business tasks inside the server.
- Clients access an enterprise bean via CDI (preferred) or JNDI.
- A session bean is not persistent.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Session beans are of three types: stateful, stateless, and singleton.

Stateless Versus Stateful Session Beans

There are three types of session beans:

- Stateless session bean (SLSB):
 - Interaction is contained in a single method call.
 - Business process does not maintain client state.
- Stateful session bean (SFSB):
 - Interaction may invoke many methods.
 - Business processes can span multiple method requests, which may require maintaining a client state.
- Singleton session bean
 - is instantiated once per application and exists for the lifecycle of the application.
 - designed for circumstances in which a single enterprise bean instance is shared across and concurrently accessed by clients.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A stateless session bean does not maintain a conversational state with the client. When a client invokes the methods of a stateless bean, the bean's instance variables may contain a state specific to that client but only for the duration of the invocation. When the method is finished, the client-specific state should not be retained. Except during method invocation, all instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client. That is, the state of a stateless session bean should apply across all clients.

A stateless session bean can implement a web service, but a stateful session bean cannot.

Stateful Session Beans

The state of an object consists of the values of its instance variables. In a stateful session bean, the instance variables represent the state of a unique client/bean session. Because the client interacts ("talks") with its bean, this state is often called the conversational state.

As its name suggests, a stateful session bean is similar to an interactive session. A stateful session bean is not shared; it can have only one client, in the same way that an interactive session can have only one user. When the client terminates, it should remove the bean instance.

The state is retained for the duration of the client/bean session. If the client removes the bean, the session ends and the state disappears. This transient nature of the state is not a problem, however, because when the conversation between the client and the bean ends, there is no need to retain the state.

Note: In Java EE 6 EJBs can be collocated in the same WAR file as other components, like web components. Also, using CDI, EJBs can be injected into other components enabling direct calls to the EJBs from components in other tiers without requiring intervening or helper objects.

Singleton Session Beans

Singleton Session Beans are:

- New in Java EE 6
- Instantiated once per application and exists only for that application lifecycle; no pool of beans
- Designed to be shared across concurrent access by clients
- Designed to maintain state between client invocations
- A convenient way for EJB applications to receive callbacks during application initialization or shutdown



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Applications that use a singleton session bean may specify that the singleton should be instantiated upon application startup, which allows the singleton to perform initialization tasks for the application. The singleton may perform cleanup tasks on application shutdown as well, because the singleton will operate throughout the lifecycle of the application.

When to Use Session Beans

Stateful session beans are appropriate if any of the following conditions are true.

- The bean's state represents the interaction between the bean and a specific client.
- The bean needs to hold information about the client across method invocations.
- The bean mediates between the client and the other components of the application, presenting a simplified view to the client.
- Behind the scenes, the bean manages the work flow of several enterprise beans.

To improve performance, you might choose a stateless session bean if it has any of these traits.

- The bean's state has no data for a specific client.
- In a single method invocation, the bean performs a generic task for all clients. For example, you might use a stateless session bean to send an email that confirms an online order.
- The bean implements a web service.

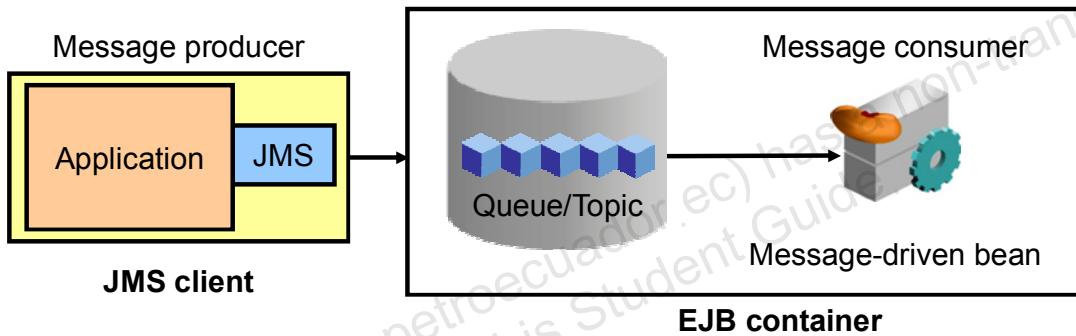
Singleton session beans are appropriate in the following circumstances.

- State needs to be shared across the application.
- A single enterprise bean needs to be accessed by multiple threads concurrently.
- The application needs an enterprise bean to perform tasks upon application startup and shutdown.
- The bean implements a web service.

Message-Driven Beans

A message-driven bean (MDB):

- Is a stateless EJB that asynchronously consumes JMS messages
- Is never called directly by a client



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A message-driven bean is an enterprise bean that allows Java EE applications to process messages asynchronously. This type of bean normally acts as a JMS message listener, which is similar to an event listener but receives JMS messages instead of events. The messages can be sent by any Java EE component (an application client, another enterprise bean, or a web component) or by a JMS application or system that does not use Java EE technology. Message-driven beans can process JMS messages or other kinds of messages.

Timer Service

- The timer service of the enterprise bean container enables you to schedule timed notifications for all types of enterprise beans except for stateful session beans.
- You can schedule a timed notification to occur according to a calendar schedule, at a specific time, after a duration of time, or at timed intervals.
- Enterprise bean timers are either programmatic timers or automatic timers.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Applications that model business work flows often rely on timed notifications. Enterprise bean timers are either programmatic timers or automatic timers. Programmatic timers are set by explicitly calling one of the timer creation methods of the `TimerService` interface. Automatic timers are created upon the successful deployment of an enterprise bean that contains a method annotated with the `java.ejb.Schedule` or `java.ejb.Schedules` annotations.

Timers can be set according to a calendar-based schedule, expressed using a syntax similar to the UNIX `cron` utility. Both programmatic and automatic timers can use calendar-based timer expressions.

What Is JAX-WS?

JAX-WS:

- Stands for Java API for XML web services
- Defines the core specification for the web services standard in Java EE 5/6
- Enables you to expose both POJOs and stateless Enterprise JavaBeans (EJBs) as web services by implementing metadata annotations
 - Is the ability to create a web service so easily from any POJO a good thing? What are some of the implications, impacts and pitfalls of this approach?



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Java API for XML Web Services (JAX-WS) is a specification for building Web services and Web service clients that communicate by using XML-based protocols (such as SOAP). JAX-WS allows developers to write message-oriented as well as RPC-oriented Web services.

For more information about Web services technology and its standards, refer to the *Oracle Fusion Middleware 11g: Build Web Services* course.

Representational State Transfer (REST)

- Is an architectural style for designing web services
- Is not associated with the implementation of the web service
- Enables you to define the interface of the web service as a URI
- Supports sending messages as simple XML documents
- Has the following characteristics:
 - Uniform interface
 - Named resource
 - Interconnected resource representation
- Provided by JAX-RS in Java EE 6



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

It is analogous to the client-server architectural style. REST emphasizes on the representation of a resource as a Uniform Resource Identifier (URI).

REST uses simple XML documents (not SOAP envelopes) over HTTP to send messages.

Some of the basic characteristics of REST systems are:

- All resources are accessed with generic interfaces such as HTTP GET, POST, PUT, and DELETE.
- All resources of the system are named using a URL.
- All resource representations are interconnected using URLs.

JAX-RS

JAX-RS provides a standardized API for building RESTful web services in Java. The API basically provides a set of annotations and associated classes and interfaces. Applying the annotations to Plain Old Java Objects (POJOs) or stateless EJBs enables you to expose web resources. The specification for the initial release of the technology, JAX-RS 1.0, was finalized in October 2008 and a reference implementation named Jersey is also available. Java EE 6 includes the latest release of the technology, JAX-RS 1.1, which is a maintenance release that aligns JAX-RS with new features in Java EE 6.

Enterprise Application Container Services

- Concurrency services
- Transaction services
- Security services
- Connection pooling services
- Messaging services
- Naming services
- Instrumentation services
- Persistence services



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Enterprise applications tend to require a core set of services that ensure the integrity of its operation and the systemic qualities that allow it to scale well. These services are not trivial to implement from scratch, and if not architected properly, could result in concerns that cut across your domain model and diminish reusability.

Concurrency Services

Any running thread that relies on a resource outside the JVM, implementation must wait for a response from that service before allowing the thread to continue operation. Without the creation of new threads to run that same process concurrently, numerous requests for the same operation can create a significant bottleneck in an enterprise application without taxing the resources of the hosting server.

Therefore, the usage of a multithreaded environment for long running processes and processes that access external resources is highly advisable. When implementing a multithreaded environment, great care needs to be taken to ensure the integrity of any shared state within the running application. This is accomplished in Java technology with the use of object locking.

Object locks ensure that threads take turns accessing a piece of code around which an object lock might have been created. These segments of code, referred to as synchronized blocks, should be as short as possible.

Note: Synchronized blocks that access external resources have the chance of becoming performance bottlenecks themselves.

Transaction Services

Transactions allow you to treat a sequence of operations as a single atomic operation. It is often desirable to have a commit or rollback approach to business logic. With a transaction, if you encounter a failure in one of the steps of your atomic sequence, you can rollback each of the previous steps in your sequence, in reverse order, back to the point where your transaction began.

Security Services

Certainly, securing access to sensitive resource that are accessible using a public Web interface is a good idea. However, it is also important to secure the access of any distributed components that make themselves available for remote execution.

Connection Pooling Services

In Java EE 6, the connection management contract supports connection pooling, a technique that enhances application performance and scalability. Connection pooling is transparent to the application, which simply obtains a connection to the EIS.

DataSource objects that implement connection pooling also produce a connection to the particular data source that the DataSource class represents. The connection object that the getConnection method returns is a handle to a PooledConnection object rather than being a physical connection. An application uses the connection object in the same way that it uses a connection. Connection pooling has no effect on application code except that a pooled connection, like all connections, should always be explicitly closed. When an application closes a connection that is pooled, the connection is returned to a pool of reusable connections. The next time getConnection is called, a handle to one of these pooled connections will be returned if one is available. Because connection pooling avoids creating a new physical connection every time one is requested, applications can run significantly faster.

A JDBC connection pool is a group of reusable connections for a particular database. Because creating each new physical connection is time consuming, the server maintains a pool of available connections to increase performance. When it requests a connection, an application obtains one from the pool. When an application closes a connection, the connection is returned to the pool.

Messaging Services

Messaging-oriented services provide the framework for the delivery of asynchronous out-of-band messages. Messaging systems should have a high degree of fault tolerance to lessen the likelihood of losing a message in an outage condition. The ability to persist messages that have been submitted for delivery, as well as an intelligent handshake process for the production and consumption of messages within the context of a transaction, is an important concern.

Naming Services

One of the core motivations behind architecting for an enterprise application is the ability to distribute components to separate servers for hosting. Scaling a single server might not be the most cost effective way to handle high load in all tiers.

For objects in two distinct JVM heap spaces to properly communicate with each other, the calling object needs a proxied handle to the item with which it wants to communicate. That handle directs the caller to the appropriate location of the callee in the callee's JVM heap space. The mechanism for managing the registration and lookup of these handles is referred to as a naming service.

Instrumentation Services

Java Management eXtensions (JMX) provides a facility for the runtime configuration and stateful monitoring of your application components. The classes that manage the state of a component are referred to as Managed Beans (MBeans). MBeans are instrumented and monitored through an MBean Server. Java EE 6 application servers provide MBean servers and invariably provide for their own configuration using that MBean server.

Persistence Services

Java EE 6 application servers provide an implementation of the Java Persistence API (JPA). JPA is an Object/Relational Mapping (ORM) technology that provides a framework for the management of application state within an RDBMS. The objects that represent the application state in JPA are called entity classes, and are discussed more fully in the Integration tier module.

Lesson Agenda

- Business tier technologies
- Architecting the domain model
- Development best practices

Architecting Domain Model Services

- Domain Model Services represent the business logic of an enterprise application.
- To architect your domain model services you can use:
 - Session beans (synchronously and asynchronously)
 - JMS and MDBs (asynchronously)
 - CDI and JNDI for injecting and locating services
 - Rules engines
 - Workflow engines



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Stateful Session beans are used to represent a conversational state between a client and an instance of an object that either provides a facade to, or implements a domain model component. Because they are conversational, clients can expect that when they set the state of a session bean in one method invocation, that state is reflected in subsequent method invocations.

The client might be located in a different JVM, and hence be considered remote to the session bean. Without a direct reference to an instance from within the same JVM, the garbage collection mechanism would most certainly throw the instance that the client is using out of the heap space. The client would then be left without any stateful representation on the server side. To keep the garbage collector from kicking in, the EJB framework keeps track of all the session beans that clients use, and manages their life cycles based on client requests.

Stateful session beans are instantiated when a client makes a request for a new bean. They are explicitly destroyed when a client asks for a bean to be removed. Stateful session beans that have sat idle for some period of time might get passivated to a persistence mechanism like a file system. This behavior is preferred to just throwing the client's object away or consuming the heap space memory with idle objects. Objects that have been passivated are reactivated and placed back into the heap space memory when the client calls a method on the object again.

The EJB specification does not make any reference to the manner in which session beans should be clustered for high availability and load balancing. As such, it is left up to each application server vendor to implement a clustering strategy. In most cases, for stateful session beans that participate in container-managed transactions, the replication of a stateful session beans state is initiated by the commit operation.

Some application servers initiate state replication across the cluster every five seconds for stateful session beans that are declared to have bean-managed transactions. Therefore, there is the possibility of data loss within this five second window in the event that your transactions are bean managed. Consult your application server's documentation for specific details on their clustering strategy.

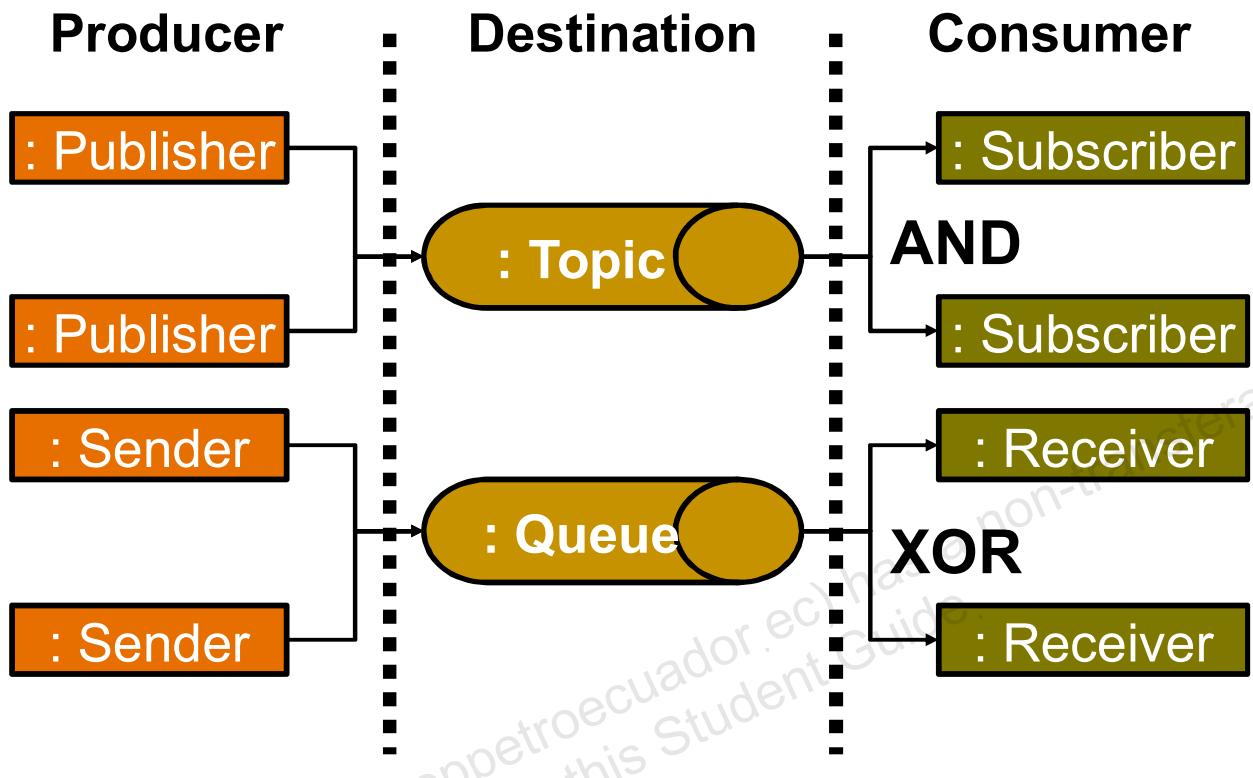
Part of the deployment process of a session bean involves the source code generation of stubs for the local or remote home interface and the local or remote business interface. Many application server vendors take advantage of this opportunity to inject their load balancing strategy into these stubs that the client uses.

Stateless Session Beans

Even though a client might be holding on to what appears to be a business proxy to a server component, each method invocation on a stateless session bean might be using a different session bean from the ready pool in the container. In a clustered scenario, it is even possible that each method invocation is using a different stateless session bean on a different machine.

While the EJB specification suggests that stateless session beans placed back into the ready pool should be re-initialized, short of reinstantiating the object, which would defeat the performance enhancements of the pool, it never suggests a strategy for achieving this. You should err on the side of caution and refrain from setting any state on the session bean, or at least provide a remove life-cycle method that cleans up the state. Otherwise, there is a chance that this state could be made available to another client.

JMS Asynchronous Communication



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The Java Messaging System (JMS) is a specification that defines a facility for the delivery of asynchronous messages. A producer that passes a message to a JMS service does not have to wait for a response. The JMS service creates a new thread for the processing of the message, and allows the producer of the message to continue with its own processing.

JMS has two message destination models. One is a publish-subscribe model associated with topics. The other is a point-to-point model associated with queues. Topics and queues are almost identical in their operation. They act as first-in-first-out (FIFO) pipelines with filtering capabilities. Producers of messages in either model are also similar in their operation, except in the case of topics, you call them publishers, and in the case of queues, you call them senders. Either model can have multiple producers publishing or sending messages to a destination.

Where the models differ is on the consumption side. Each subscriber of a topic can be guaranteed to get a copy of all messages within a topic. Topics can be thought of as a broadcast distribution mechanism. However, with queues, only a single receiver is allowed to consume each message.

Most application servers allow multiple receivers to register as consumers of messages from a queue. Because only one receiver can consume the message, this creates a useful load balancing mechanism. The first receiver to establish a handshake for message consumption is given the opportunity to process the message. Any receiver that is currently operating on a message does not attempt to consume additional messages until it is finished with its current process.

The slide shows the two different message models.

Message Driven Beans

- Can be configured to listen to topics or queues
- Easier to write than stand-alone receivers
- Can utilize container managed transactions (CMT)
- Can now use session beans and servlets asynchronously with a simple annotation, in place of JMS and MDBs



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A message driven bean (MDB) is a special message consumer designed to run within an application server container. It has available to it all the enterprise application services that are required of enterprise applications. With EJB 3.1, any domain model service can be annotated in such a way as to become an MDB without being tightly coupled to the Java EE 6 API.

Message-driven bean instances are instances pooled in a manner similar to stateless session beans. If a message-driven bean is configured to act as a listener of messages from a queue, the load balancing characteristics are inherently applied. A good JMS implementation should provide a persistence mechanism behind the destinations. This persistence mechanism should save messages as they are published or sent to the destination. Those messages are not removed from persistence until the consumer of each message commits their transaction. This helps ensure the fault tolerance of the messages in the event of a server failure.

Messages that were persisted are loaded back into the destination when the server comes back online. JMS servers that implement an RDBMS message persistence mechanism can be useful for the analysis of messages in the destination. consumption of messages off the destinations. While there is certainly a great deal of value in the use of transactions, there is also the danger of message thrashing.

Message thrashing

Great care needs to be exercised with the usage of transactions that originate with the occurs when the consumer's transaction gets rolled back and the message that the consumer was operating on becomes available for consumption on the destination again. If the underlying reason for the failed transaction is not fixed, the message might get immediately consumed off the destination again and ultimately hit the same transaction rollback condition.

Message thrashing can put such a tremendous load on server resources that it becomes totally unresponsive. Policies should be enacted that prevent messages that have been rolled back from being placed on the same destination.

Note: While using Message Driven Beans for asynchronous processing certainly works, it also forces you to deal with messaging and JMS, even for relatively lightweight functionality. This is precisely the problem asynchronous session bean invocation is designed to solve. With this enhancement, you can do asynchronous processing simply by annotating a session bean method.

JNDI Naming Server

- Maps simple names to object instances in some JVM on some machine
- Implemented with:
 - RMI Registry
 - LDAP Directory
 - COSNaming
- Choose naming server based on replication needs for fault tolerance
- Required for explicit lookups for applications running outside the Java EE-server managed container.



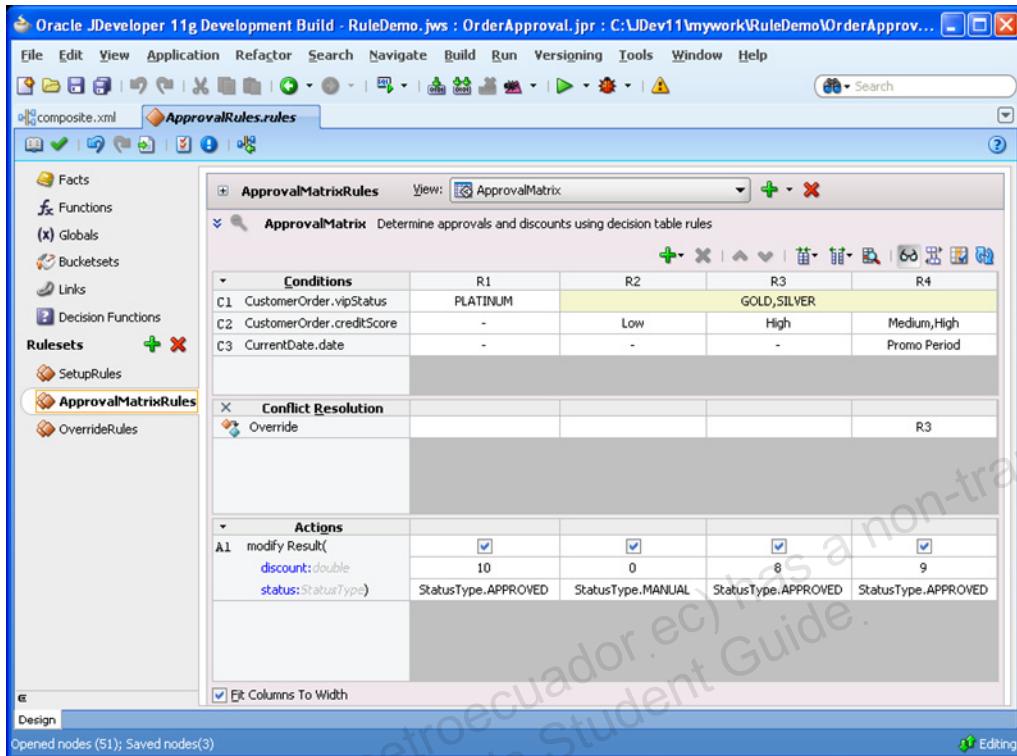
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A Java Naming and Directory Interface (JNDI) server provides the naming services for components in a Java enterprise application. JNDI provides the abstraction layer that servers and clients use to register and lookup names that are mapped to object instances on some particular server.

The replication strategy used by JNDI naming servers is dependant upon the underlying implementation. The most common implementations are RMI Registries, LDAP Directories, and CORBA COSNaming Providers. Of those, JNDI naming servers based on LDAP Directories tend to have the most robust replication and synchronization mechanisms.

JNDI naming servers can either be deployed for redundancy, or simple hot-standby on a shared IP address. Strategies for the handling of redundant naming servers vary. One approach is to configure clients with the locations of all naming servers, and cycle to the next server should a lookup fail.

Rules Engines



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

If your business logic is particularly complex, or somewhat dynamic in nature, there might be a significant maintenance, and hence cost, advantage in the externalization of that business logic from the source code itself. Rules Engines are designed to create the semantics for easily managing fine-grained business logic, and executing it at runtime.

You can view a rules engine as a sophisticated interpreter of if-then programming constructs, geared towards the derivation of a single result. A rule itself, similar to an if-then construct, is composed of a condition and an action.

Rules engines operate on a predefined collection of rules called a rule execution set. At runtime, these rules are fed a combination of predefined facts and the input data objects on which to operate. The rules engine automates the execution of the entire rule set to output a new data object.

Using rules engines create the following strategic advantages:

- The rules themselves are more easily communicated and understood, especially if managed with a graphical tool.
- They separate the business logic from the implementation logic.

- Business logic can be easily changed without requiring recompilation.
- Businesses can gain a competitive advantage with the ability to rapidly reengineer their business processes.

Rules engines are most often used to manage business logic in the business tier. However, they can also be used to manage complex web page flows in the web tier.

You should probably consider choosing a rules engine that implements the Java Rule Engine API (JSR 94).

Basic Oracle Business Rule Concepts

- Facts:
 - Are data or business objects on which the Rules Engine evaluates rule conditions
- Rules:
 - Are declared as: “IF *condition* THEN *action*”
 - Have an action: Assign, Assert, Call function (or Java method)
- Ruleset:
 - Has a collection of rules
 - Is a unit of execution
 - May be chained
- Dictionary:
 - Has a collection of fact types, global variables/constants, functions, and rulesets



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Oracle Business Rules are stored in a rule dictionary, which is a file in the file system. A rule dictionary contains one or more definitions of:

- Facts
- Constraints
- Functions
- Rulesets

A ruleset is a collection of one or more related rules that are seen as a unit of execution and can be chained together to yield an outcome.

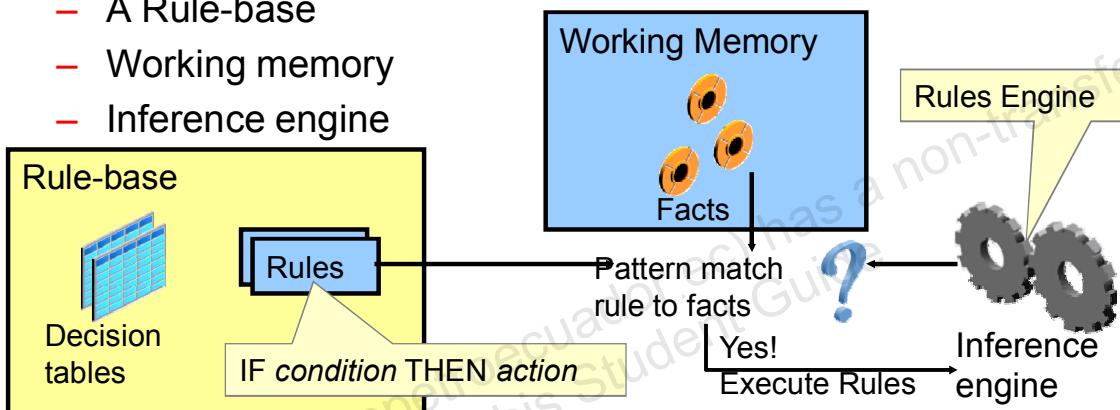
Each rule is declared as a conditional expression with an action. The condition evaluates and compares facts. If the evaluation of facts is true, then one of the following actions is performed:

- Assigning values to other facts or results
- Asserting values
- Calling functions (or Java methods) to execute procedural code

Oracle Business Rule Components

Oracle Business Rules:

- Enables evaluation of dynamic decisions at run time
- Allows automation of policies, computations, and reasoning separate from application code
- Executes a Rete algorithm with the following components:
 - A Rule-base
 - Working memory
 - Inference engine



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Oracle Business Rules enable dynamic decisions to be evaluated at run-time enabling automation of policies, computations, and reasoning that can be separated from underlying application code. Oracle Business Rules is a rule-based system based on the foundations of the Rete algorithm. The Rete algorithm stores partially matched results in a single network of nodes in working memory, enabling the Rules Engine to avoid unnecessary rechecking when facts are added, modified, or deleted. To process facts and rules, the Rete algorithm creates and uses an input node for each fact definition and an output node for each rule. A rule-based system consists of the following:

- **The Rule-base:** Business policies expressed as IF-THEN rules and Decision Tables
- **Working memory:** Fact information added to the system by applications using assert calls
- **Inference Engine (Rules Engine):** Processes the rules, performs pattern-matching to determine which rules match the facts, for a given run through a set of facts

Oracle Business Rules is a data-driven forward chaining system, such that the facts determine which rules are evaluated. When a matching rule fires, the rule may add more facts. New facts are again compared against the rules. The process repeats until a conclusion is reached or the cycle is stopped or reset. This process is called an inference cycle.

Workflow Engines

- In general, rules engines are geared towards automated execution of rules, and do not focus on supporting wait states.
- A wait state would be treated as the end of a rule set.
- While wait states could be handled with the appropriate use of event handling mechanisms to piece rule sets together, a workflow engine is a better choice.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Workflow engines are geared towards the implementation of business processes. The difference between business logic and business processes is really just one of granularity. Business logic is fine grained, whereas business processes are coarse grained. A workflow interprets each of its elements as being service providers. Those service providers might be synchronous processes that provide an immediate automated response, or, they might be asynchronous processes that require human intervention.

All good workflow engines operate on workflow definitions that can be easily modeled in a graphical tool to support Graphical Oriented Programming. These modeling tools are designed to be used by business analysts that endeavor to implement a high level business process. At the center of most Service-Oriented Architectures, you will probably find a workflow engine. In particular, a workflow engine that implements the Business Process Execution Language (BPEL) XML schema are good candidates for ensuring flexibility and maintainability.

Architecting Domain Model Entities

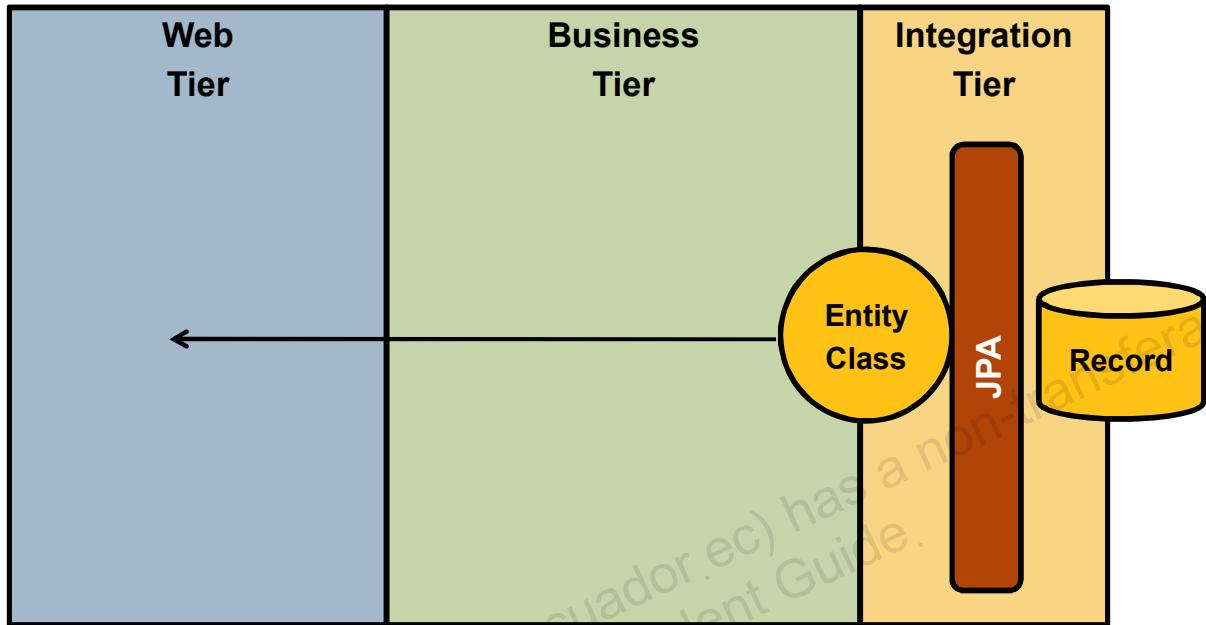
- Most enterprise application development methodologies are use case driven processes.
- The use cases should drive a domain model discovery process.
- Part of that domain model includes the stateful entities and their structural relationships.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Development strategies that attempt to derive the domain model entities from a Relational Database Management System (RDBMS) table schema, often find the process of mapping an object oriented domain model object graph to the existing table schema a challenging one. If at all possible, you should generate the RDBMS table schema based on the Java Persistence API (JPA) annotations in the domain model entities that you discovered during the analysis and design of the application. The following topics discuss how this approach can simplify the state management and handling issues typically found in the business tier.

Java EE 6 Business State



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

With Java EE 6, entity classes in JPA now give you a way to have an ubiquitous business state representation that is portable across the web, business, and integration tiers, as shown in the slide.

The strategy to achieve this involves fitting your domain model entities with JPA annotations that map the entities to tables in a database.

Mapping Domain Value Objects to XML

- The Java Architecture for XML Binding (JAXB) 2.2 specification drives the implementation of robust XML marshalling and unmarshalling frameworks.
- With JAXB 2.2, the same domain value objects that are annotated for JPA, can also be annotated with JAXB to customize the mapping between the value objects and the XML.
- Most JAXB 2.2 implementations also have a process for source code generating the XML schema using the entity classes and their JAXB annotations as input.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

JAXB 2.0 is also the serialization and deserialization mechanism for the Java API for XML Web Services (JAX-WS). So, the same marshalling and unmarshalling framework that you use independently for Java technology and XML can be used with your web services as well.

Distributing Domain Model Components

- Scaling vertically in the business tier is not always cost-effective. Sometimes, it is more cost-effective to scale horizontally.
- The remoting capability of Java EE components supports this model.
- To find the most appropriate division of components for deployment to multiple servers, you should look for components with the least amount of communication between them.
- Profile your application to get metrics on communication between components and processing times for components.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Figuring out the best way to divide up the domain model in the business tier, if at all, to run on multiple machines is one of the most challenging jobs of an architect. A simple division based on package boundaries is not always appropriate. There might also be particular components that need to be replicated in a cluster based on their demand.

An application profiling tool can be of tremendous benefit in the metric gathering process when attempting to calculate the existing or potential load on individual components. These tools should give you an indication for the amount of communication between components and the time spent inside each component processing individual requests. Long request times might be indicative of the need for additional resources, making the component a candidate for replication in a cluster.

Creating a remote client/server relationship between two components that communicate with each other extensively can significantly degrade the performance of the overall system due to network overhead, bandwidth limitations, and latency.

The Component Distribution Golden Hammer

- While you should certainly code for the possibility of component distribution, you should always treat it as a last resort after carefully weighing the costs and benefits associated with scaling vertically instead of horizontally.
- The performance impact of remoting domain model components cannot be overstated.
- Java EE 6 enterprise beans are easier to use locally (collocated) with the components that need to use them.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

All too often, architects start with the assumption that splitting up the domain model, or replicating it, to run on different hosts is the best practice. This thought is a bit of a Golden Hammer antipattern. A Golden Hammer is a solution for all problems, which in reality, is not practical. The cost for performance when scaling horizontally might actually be significantly greater than the cost to scale vertically with an enterprise operating system. If in doubt, you should always load test a single and multi-server scenario, interpolate the results for a production load, and look at the cost comparison. Beyond the simple cost of the hardware, you also need to consider the cost of the complexity associated with writing additional code to minimize network traffic and managing the deployment to multiple servers.

Creating Coarse Grained Façades

- Whether you have made the decision to distribute your domain model, or perhaps you want to make certain components available for remote invocation from outside the domain model, you need an additional layer of code.
- Domain models are fine grained.
- There will be certain circumstances in which particular actions are repeated that represent a sequence of calls to the domain model.
- Rather than taking the performance hit of numerous remote method invocations, it would be better to consolidate those requests to the domain model into a single request.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Coarse-grained services need coarse-grained entities. Therefore, the application service layer needs to communicate with its clients using Data Transfer Objects (DTOs). Ideally, your DTOs are just object graph aggregations of your domain model entities. However, circumstances can arise where, in the interest of limiting the size of the DTO, an entirely new entity must be created and programmatically mapped to a set of domain model entities.

These consolidated requests are put into a set of classes that compose the Application Service layer. They present themselves to as coarse-grained façades, or views, to the domain model.

Lesson Agenda

- Business tier technologies
- Architecting the domain model
- Development best practices



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Development Best Practices

- Whether to use HttpSession within the Web container, or a stateful session bean in the EJB container to hold client session state depends on the nature of the client.
- If the client is a Web browser, then it makes more sense to use HttpSession to hold the state. That way, the client needs only to make requests into the business tier when it is invoking a business method.
- Clients that are themselves application components, wanting stateful behavior from business logic, are better motivators for the use of stateful session beans.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

With Java EE 6, you can have stateful session beans within an HttpSession, using the bean as a POJO and holding it in the HttpSession, then using CDI to inject the reference as needed.

Note: HttpSession can use a stateful session bean to hold state if the session bean is collocated with the web components.

Exception Handling

- Limit the damage caused by exception anti-patterns:
 - Too much information
 - Lie of omission
 - Head in the sand
- Wrap checked exceptions inside runtime exceptions.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The lack of a well-defined framework for handling errors and exceptions can create an application with low systemic qualities and poor user acceptance. Proper error and exception handling strategies need to become part of the corporate coding style guidelines.

There are three main exception handling antipatterns to be avoided:

- Too Much Information
- Lie of Omission
- Head in the Sand

The Too Much Information exception handling antipattern describes the act of catching an exception, logging it, wrapping it in a new exception, and rethrowing it. The example code illustrates this.

Too Much Information Exception Handling AntiPattern

```
catch (Exception e) {  
    e.printStackTrace();  
    throw new WrappedCheckedException(e);  
}
```

This results in the syndrome. Only the original exception provides any pertinent information related to why an exception was thrown, all the other wrapper exceptions just result in hiding the real exception stacktrace at the core of an exception that has been wrapped an unknown amount of times.

The Lie of Omission antipattern throws away important information about the exception as shown:

Lie of Omission Exception Handling AntiPattern

```
catch (Exception e) {  
    // print message only.  
    System.err.println(e);  
    throw new WrapperException(e.getMessage());  
}
```

Any stack trace of the wrapper exception does not contain a stack trace about the real exception. The output only shows the error message.

The Head in the Sand antipattern is a bit of a do nothing approach:

Head in the Sand Exception Handling AntiPattern

```
catch (Exception e) {  
    e.printStackTrace();  
}
```

The exception is logged, but poorly, and is never properly propagated. Significant application errors could be occurring and no one would know until someone had a look at the server logs. A more egregious version of this would be to not even print a stack trace to stdout.

Using Runtime Exceptions

- Unchecked exceptions are not always a bad thing.
- There are some strong opinions with regard to the appropriateness of using checked or unchecked exceptions, and you are encouraged to familiarize yourself with those arguments.
- The Java EE 6 API has made the use of checked exceptions largely unnecessary.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

One strategy for exception handling encourages you to never rethrow the same checked exception. Instead, catch checked exceptions once, then wrap them in unchecked RuntimeException(s). The RuntimeException that you create should have enough meta data attributes to sufficiently describe the nature of the error that occurred. An example of this strategy, with logging, is shown in Code 5-4.

```
Exception Handling with Runtime Exceptions
} catch (SomeCheckedExceptions sce) {
    String errorKey = "error.some.problem";
log.error(errorKey, sce);
    throw new MyAppRuntimeException(errorKey, sce);}
```

Most web frameworks catch all exceptions, including runtime exceptions, somewhere in their controller. The process of deciding the appropriate view to dispatch to the end user can come from an inspection of the error, or a generic error view associated with the requesting page.

You should try to initialize your RuntimeException with key-based descriptors, and pull your user friendly error messages from a ResourceBundle. This prevents developers from accidentally redefining the semantics for a particular type of error, and provides for internationalization support.

Logging

- Dumping messages to `stdout` and `stderr` is expensive.
- Logging frameworks fork off separate threads for logging.
- Java EE 6 interceptors can implement logging.
- Questions:
 - What is your strategy for storing log files?
 - Where and for how long?
 - Is the intended channel open for logging?



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

It is true that just about every and sample application can, at some point, do a `System.out.println()`, either directly or using a `printStackTrace()`. In practice, however, this is a bad idea. The `println` method is not multithreaded, and writing to the console is a resource-intensive operation. Leaving these statements in a production application can hamper the quality of service.

Certainly, these sorts of debugging messages are needed. However, the best practice is to use a logging framework, such as the JDK-based logging framework, or the ever popular Log4J. These frameworks are configurable for the amount of debugging information you want to see. They are also multithreaded, so they have less of a performance impact on the system than the `println` approach.

Error view components should examine the level of debugging currently enabled in the logging framework before displaying the error. If the debugging level is set to high, for development purposes, then perhaps the display of the stack trace is appropriate for the view component. Once the application goes to production, the debugging level can be lowered so as not to subject end users to what looks like a catastrophic server failure.

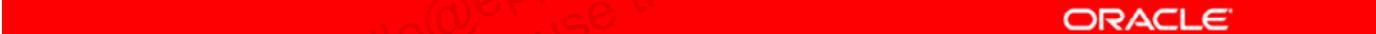
Interceptors can implement logging

Interceptors intercept invocations and lifecycle events on an associated target class. An interceptor is a class whose methods are invoked when business methods on the target class are invoked and/or lifecycle events such as methods that create/destroy the bean occur. Interceptors are typically used to implement cross-cutting concerns like logging, auditing, and profiling.

Note: Any managed object (managed beans, EJBs, and so on) can have an interceptor associated with it.

Additional Resources

- Jendrock, Eric, Ian Evans, Devika Gollapudi, and Kim Haase, *The Java EE 6 Tutorial: Basic Concepts (4th Edition)*. Prentice Hall, 2010.
- Goncalves, Antonio, *Beginning Java EE 6 with GlassFish 3, Second Edition*. Apress, 2010.
- Qusay H. Mahmoud, *Getting Started With the Java Rule Engine API (JSR 94): Toward Rule-Based Applications* [<http://java.sun.com/developer/technicalArticles/J2SE/JavaRule.html>] accessed APR 2007.
- Derek C. Ashmore, *The J2EETM Architect's Handbook: How to be a successful technical architect for J2EETM applications*. DVT Press, Lombard, IL, 2004.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Quiz

A good practice in Java Enterprise Architecture is to encode business logic in: (choose all that apply)

- a. Java Servlets with JPA
- b. JavaServer Pages with JPA
- c. Enterprise JavaBeans
- d. JAX-RS Services



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: c

Quiz

Given an NFR that the user never waits more than a second before a page returns, what feature of EJB's could you use to prevent a long delay in a complex business process?

- a. Timer Service
- b. Stateless session beans methods
- c. Stateful session context methods
- d. Asynchronous session bean methods



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: d

Quiz

Given the emphasis on loosely coupled programming, in what situation would tightly coupled beans make sense?

- a. When both beans call on the same database table
- b. When beans are used to access JNDI resources
- c. When beans fit together as a logical unit
- d. When beans user a transaction service



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: c

Quiz

In a situation where a bean is required to be thread-safe, and is shared by multiple clients and performs discrete tasks, what type of Session bean is the most appropriate?

- a. Stateful
- b. Stateless
- c. Message-Driven
- d. Singleton



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: d

Quiz

Given a design where the client is a web browser, what is a best practice for state management?

- a. HttpSession
- b. Stateful Session beans
- c. URL rewriting
- d. Cookies



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: a

Summary

In this lesson, you should have learned how to:

- Describe the value in using enterprise application container services
- Describe the architectural options for implementing domain-model services
- Describe the architectural options for implementing domain-model entities
- Distribute domain-model components
- Describe the best practices for exception handling and logging



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 10 Overview: Refining the Business Architecture

This practice covers the following topics:

- Refine the business architecture to support the more granular access requirements of the improved user interface and Web tier.
- Interact with persistence frameworks to access data.
- Choose which of several architectural solutions are appropriate for the problem.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

In this practice you first refine the Business tier architecture to support the expanded requirements of the Web tier. As there are multiple possible solutions, you will also discuss your choices and alternatives and their pros and cons.

11

Developing an Architecture for the Integration and Resource Tiers

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the challenges in Enterprise Information System (EIS) integration
- Describe the roles of the integration tier
- Describe the EIS resource tier
- Review Java integration technologies and best practices
- Apply integration-tier patterns
- Describe how Service-Oriented Architecture (SOA) facilitates system integration
- Describe SOA best practices



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Discussion Questions

- Java EE technology systems often need to integrate with existing enterprise information systems. What kinds of systems fall under the category of EIS? What, if anything, can be done to simplify this integration?
- What components in the Java EE technology are available for the integration tier?
- What is a Service-Oriented Architecture? How does it facilitate system integration?



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Examining Enterprise Information System integration
- Reviewing Java integration technologies
- Applying Integration tier patterns
- Examining Service-Oriented Architecture (SOA)



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Challenges of Integration

- The distribution of the EIS resources
- The heterogeneity of the EIS resources
- Rapid changes in business requirements and technology



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Over the years, enterprise organizations have invested heavily in a variety of technologies and Enterprise Information System (EIS) resources to establish their information technology (IT) infrastructure. Many of these technologies and resources were introduced as solutions to isolated problems, such as human-resource management and enterprise content management.

To sustain and increase the values of these investment, enterprise organizations must integrate them in an agile manner so that the overall information system can quickly adjust to business requirement and technology changes.

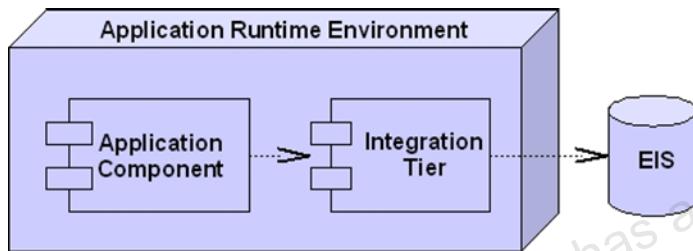
System integration has always been a challenge due to the following factors:

- **The distribution of the EIS resources:** Typically, the systems to be integrated are deployed to different runtime environments. To design an integration solution, you must consider the communication across a potentially slow, unreliable, and insecure network.

- **The heterogeneity of the EIS resources:** Systems implemented by different vendors can vary dramatically in their programming languages, operating platforms, communication mechanisms, and support for advanced features, such as security and distributed transaction. To design an integration solution, you must rely on interoperable technologies that allow these systems to work with each other.
- **Rapid changes in business requirements and technology:** Changes happen inevitably. For example, customer or business requirement change can require a modification to the system. An appropriate new technology might be capable of delivering better qualities. To react quickly to these changes, the integration solution you design must minimize the dependencies between different systems.

The Integration Tier

To facilitate the system integration in an enterprise application, the best approach is to introduce an integration tier as an adapter between application components and the EIS resources.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The advantages of the using an integration tier for EIS resource integration include:

- Reduce the dependency between the application components and enterprise information systems.
- Increase the reusability of integration components.

Compared to the presentation and business tiers in an enterprise Java application, the integration tier is less distinguishable due to the following factors:

- There might not be an obvious distinction between the business and integration tier components. An example of this is the message-driven bean (MDB). MDB is considered a well-defined EJB component model in the business tier. However, if the distributed application needs to consume asynchronous messages produced by a message system that supports JMS, MDBs are available to work as integration-tier components.
- With the exception of the Java EE Connector Architecture (JCA) resource adapters, there are no concrete component models described in Java EE for the integration tier. The technology and design used in the integration tier depend on the type of the EIS resources. In many cases, the integration tier is implemented as reusable adapter classes or components that are deployed to the same runtime environment hosting application components.

Note: Even though the Java EE technology does not specify standard component models for the integration tier, you can still rely on many standard technologies and APIs in both the Java SE and Java EE platforms to design and implement interoperable and reusable integration tier components, for example, JDBC, JMS, and Java Authentication and Authorization Services (JAAS).

The integration tier can be introduced anywhere an application component needs to access the external EIS resource. The application component can be from the presentation tier or the business tier. To this extent, the integration tier reflects an adapter-based design that attempts to decouple the application components and external resource.

A distributed application can require different levels of integration, which leads to a even less clear boundary of the integration tier. A good example of this is a web-based portal that provides a single point of access to essential enterprise services. In this scenario, not only do you need to rely on a variety of technologies to integrate back-end services, you also need to aggregate the access to these services into one web-based user interface.

The EIS Resource Tier

- Data Resources
 - Relational databases
 - Nonrelational data sources
 - Nonpersistent data resources
 - Messages from a messaging system
 - Results from a legacy computational system
 - Real or near-time data feeds
- Operational resources
 - Legacy applications
 - Native applications or functional libraries
 - ERP systems



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The EIS resource tier in an enterprise application encloses all the external resources that need to be integrated. EIS resources include data resources, operational resources, and other server products.

Data that support an enterprise system can come from a variety of sources, including databases that might be relational, hierarchical, or other some other form. Some data can come from other organizations, from other applications, or from real or near-time feeds.

The EIS Resource Tier

- Resource servers
 - LDAP servers
 - Security servers
- ETL tools
- Data mining
- Enterprise data models



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Relational Databases

- Relational databases are the most common form of long-term storage in use today.
- The format, capabilities, and limitations are well understood.
- Relational databases have the following characteristics:
 - They contain data in tabular form.
 - They allow searching based on potentially complex expressions, which describe the contents of the data tables.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

They impose a moderate degree of structure on the data that they store, both in compelling the data to be stored as rectangular tables and in requiring that each field within a column have the same data type. They allow the relations of data records to be represented using foreign keys and constraints.

Migrating from one relational database vendor to another (for example, MySQL, Berkeley DB, IBM DB2 or Microsoft SQL Server to Oracle) can severely impact the performance of the application. While all three are relational databases, the architectural differences of Oracle are significant. Care must be taken to ensure the features of the new database are leveraged. Consult Expert Oracle Database Architecture by Tom Kyte for a good treatment of these issues.

Nonrelational Data Sources

- Nonrelational data sources can be further divided into databases and other data sources.
- One form of a nonrelational database is the hierarchical database, such as IMS.
- This approach allows for exceptionally fast searching, but it is less flexible with respect to online decision support because it uses interactive data searching instead of the automatic data searching that is used in a relational database.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Another type of nonrelational database is the OO database. The great advantage of OO databases is that they map directly between the persistent store and the actual data that are being represented. Like relational databases, OO databases handle searching at the DBMS. However, in contrast to relational databases, OO databases use OO techniques to allow data to be navigated and manipulated in the same manner as data processed in an object-oriented programming language. Versant is one example of an OO database.

The following data, which might be used in an enterprise system, might not come from a persistent store at all:

- Messages from a messaging system
- Results from a legacy computational system
- Real or near-time data feeds, such as stock prices, weather information, and so on
- XML
- GIS (calculations v API)
- High availability, RAC, cluster, and grid

You should be prepared with training and possibility consulting to support developers, architects, analysts, and end users for the change in paradigm. The classic example is hierarchical to relational database.

Alternatives to Relational Databases for High Performance

- NoSQL systems such as key-value stores, document-oriented databases and distributed file systems are gaining popularity for very large scale distributed database deployments.
- One major difference between a NoSQL or SQL solution is in the area of data consistency.
- SQL technology has been used for transactions with ACID properties. NoSQL databases support BASE (**B**asically **A**vailable, **S**oft state, **E**ventually consistent).



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

An alternate way of storing data to account for simplicity in data management which provides horizontal scalability is quite popular within cloud-based internet web applications.
The alternatives, calling themselves NoSQL: Not Only SQL-can be broadly classified into five categories:

- Document Store
- Key/Value Pair Store
- Column Store/Big Table
- Graph DB
- Blended Solutions

A common motivation within NoSQL design is to design for scalability and failover; data is mostly partitioned and replicated across multiple nodes. Though the open source community and adoption is quite vibrant for other popular distributed databases like Cassandra, CouchDB, Riak, MonetDB, and so on, these are not yet backed by large companies offering 24x7 enterprise quality production support.

"Hence, companies wanting to use such distributed databases are forced to hire and maintain a talented pool of resources to manage and customize such complex applications within their enterprise.

Though internet and technology companies tend to pursue this path for strategic reasons, other large enterprises are still reluctant to whole heartedly adopt this new evolution—even within the cloud—for the lack of product maturity, enterprise level support and even lack of enterprise level features like-2 phase commit, atomic row level consistency, complex SQL queries, JOIN support, and so on.

Similarly, large database vendors like Oracle, IBM are equally investing within this distributed database segment to provide a compelling offering that can provide simple administration and scale horizontally while at the same time store their data within the traditional RDBMS so that these data can be used/analyzed with their existing Business Intelligence tools?"

Excerpted from: http://blogs.sun.com/natarajan/entry/a_survey_report_on_nosql

"If ACID provides the consistency choice for partitioned databases, then how do you achieve availability instead? One answer is BASE (basically available, soft state, eventually consistent).

BASE is diametrically opposed to ACID. Where ACID is pessimistic and forces consistency at the end of every operation, BASE is optimistic and accepts that the database consistency will be in a state of flux. Although this sounds impossible to cope with, in reality it is quite manageable and leads to levels of scalability that cannot be obtained with ACID.

The availability of BASE is achieved through supporting partial failures without total system failure. Here is a simple example: if users are partitioned across five database servers, BASE design encourages crafting operations in such a way that a user database failure impacts only the 20 percent of the users on that particular host. There is no magic involved, but this does lead to higher perceived availability of the system." - BASE: An Acid Alternative by Dan Pritchett, July 28, 2008 <http://queue.acm.org/detail.cfm?id=1394128>

Operational Resources

- At times, the information that a system uses does not come from a data store. Instead, it comes from some system that provides a function or operation.
- These operations can be processes or functions that are implemented as monolithic applications, application libraries, or other complex IT systems.
- To access these functional resources, a Java EE technology application requires a software bridge or adapter to enable communication.
- The Java EE Connector Architecture provides some measure of connectivity, but there are times when a more customized approach is required.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Legacy Applications

Legacy applications extend and provide additional functionality to Java EE technology applications. Mainframe applications built with COBOL, CICS, and IMS are examples of legacy applications.

Native Applications or Functional Libraries

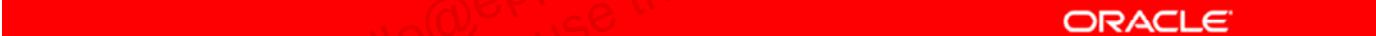
Another operational resource often used by Java EE technology applications is the native code functionality provided by dynamic libraries. Based on the desire to reuse code, many companies have built large, dynamic libraries over time. These libraries are usually very stable and well tested, but they can be difficult to integrate with Java EE technology components. For example: Resource bundle Jars for internationalization.

ERP Systems

ERP systems are multi-module application software products that allow a company to manage all aspects of its business from the perspective of a single unified product. ERP usually requires extensive process analysis and modeling, along with employee retraining or job outsourcing. Many companies are using Java EE technology to either integrate existing ERP systems or to migrate processes from proprietary ERP software. Examples of ERP systems are SAP, PeopleSoft, and J. D. Edwards.

Resource Servers

- A resource server provides access to resources for a Java EE technology application.
- You can purchase some resource servers as off-the-shelf components, while other resource servers require some in-house code development.
- Examples of resource servers include LDAP servers and security servers.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

LDAP Servers

Comparing an LDAP Server to Alternatives

Function	LDAP Server	Alternative Approach
Data storage	Can store any arbitrary data alongside a search key	A name server translates a name into an address.
Lookup	Performs lookups as simple matches	A relational database performs lookups as complex expressions.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

In any distributed system, the various elements of that system must communicate with each other. Typically, the first requirement for this communication is that one system knows the domain name of the other and possibly the port number on which the required service is offered. While you can hard code this data into the programs, or have the system administrators provide this data on the command lines of the participants, it is more effective and easier to maintain to store the information in an LDAP server.

The LDAP server provides a centralized point of contact into which each system stores its contact information so that other systems can look up the information when needed. This approach avoids considerable administrative effort and greatly increases the maintainability and flexibility of the system as a whole. Because LDAP servers do not expect change in the data they store, LDAP servers optimize read operations more highly than write operations. LDAP is mostly used for authentication.

Security Servers

- All programming is prone to errors or bugs, but errors in the security elements of a system are among the most dangerous.
- You can minimize the danger from security errors by putting all the security into a single system and by giving this single system adequate scrutiny and verification.
- This approach has the following benefits:
 - Ensures that each security element is written only once
 - Reduces the risk of introducing errors in duplicated functionality

 ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A security service typically provides an authentication mechanism that has the following characteristics:

- Verifies the authenticity of a user
- Provides the user with some sort of ticket that can be used to avoid multiple sign-on processes

Determines the functionality available to a user

One example of an authentication mechanism is the Kerberos security system available in the Solaris Operating System (Solaris OS) and Microsoft Windows.

Extract, Transform and Load (ETL) Tools

- Extracting data from outside sources
- Transforming it to fit operational needs (which can include quality levels)
- Loading it into the end target (database or data warehouse)



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Oracle Data Integration Suite includes a comprehensive set of data management components for building, deploying, and managing enterprise data integration solutions. It includes the necessary components for data movement, data quality, data profiling, and metadata management for solving any data integration and data management challenge.

Oracle GoldenGate provides real-time, log-based change data capture, and delivery between heterogeneous systems. It supports multiple data replication topologies such as one-to-many, many-to-many, cascading and bidirectional. Its wide variety of use cases includes real-time business intelligence; query offloading; zero-downtime upgrades and migrations; disaster recovery; and active-active databases for data distribution, data synchronization and high availability.

Data Mining

- Data mining is the practice of automatically searching large stores of data to discover patterns and trends that go beyond simple analysis.
- Data mining uses sophisticated mathematical algorithms to segment the data and evaluate the probability of future events.
- Data mining is also known as Knowledge discovery in data (KDD).



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The key properties of data mining are:

- Automatic discovery of patterns
- Prediction of likely outcomes
- Creation of actionable information
- Focus on large data sets and databases

Data mining can answer questions that cannot be addressed through simple query and reporting techniques.

Data mining is a powerful tool that can help you find patterns and relationships within your data. But data mining does not work by itself. It does not eliminate the need to know your business, to understand your data, or to understand analytical methods. Data mining discovers hidden information in your data, but it cannot tell you the value of the information to your organization.

Enterprise Data Model

- An Integrated view of the data that is created and used across an entire organization
- Is the “what,” independent of the how
- Eases identification and elimination of data redundancy
- Modeled with:
 - Entity Relationship diagrams (ERDs)
 - UML Domain class models
 - XML schema (XSD)
 - Messages for Enterprise SOA
- Useful in creating a canonical data model for sharing messages

ORACLE

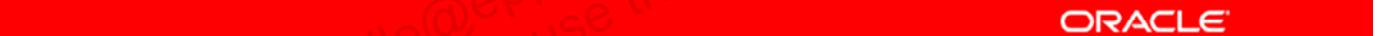
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

An enterprise data model should represent a single, integrated definition of data, independent of implementation. The model focuses on and abstracts out the key, important data concepts in an organization and the rules governing them. The enterprise data model should serve as an architectural framework for integration. It identifies sharable, redundant information across the functional, service and organizational boundaries. It provides the “single version of truth” to resolve the myriad implementations, representations and formats. It minimizes data redundancy, disparity, and errors and is core to data quality, consistency, and accuracy. The enterprise model can serve as the basis for a canonical model.

One caveat is it must be kept current and up to date.

What Do You Think?

- Is it worth it to create and maintain an Enterprise Data Model? Why or Why not?
- If you have an EDM, do you embrace it or submit to it?

A solid red horizontal bar located at the bottom of the slide.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Examining Enterprise Information System integration
- Reviewing Java integration technologies
- Applying Integration tier patterns
- Examining Service-Oriented Architecture (SOA)



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Java Integration Technologies

- The JDBC API and Object-Relational mapping mechanisms
- The Java Message Service (JMS) API and Message Driven Beans (MDBs)
- The Java EE Connector Architecture technology (JCA)
- Java Web Service technologies



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Java EE provides a rich collection of components and APIs to facilitate EIS integration both synchronously and asynchronously. Commonly used techniques include:

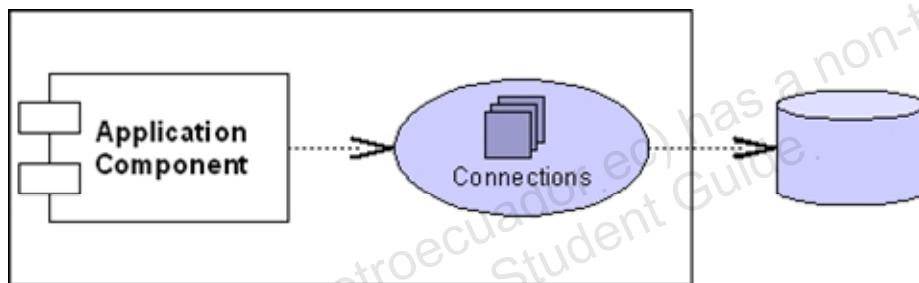
- The JDBC API and Object-Relational mapping mechanisms
- The Java Message Service (JMS) API and message-driven beans
- The Java EE Connector Architecture technology
- Java Web service technologies

Depending on the type of the underlying EIS resource, you can choose the appropriate technique to design a flexible integration solution.

Other less-commonly used techniques include the Java Native Interface (JNI) and Java IDL.

JDBC and Object Relational Mapping (ORM)

- Commonly used technologies
 - The JDBC API
 - The Java Persistence API
 - TopLink
 - Hibernate
- Connection-pooling should always be used to manage the license cost and reduce the overhead



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

JDBC API software enables you to connect Java technology programs to SQL databases. The uniform interfaces of the JDBC API software hide the mechanics of handling the connection from the developer. However, different databases sometimes implement slightly different versions of SQL. Such differences are not hidden by the JDBC API software.

The restriction on the number of concurrent connections that a database can support is a significant issue in many developments. Some databases are licensed based on this number, others suffer performance degradation if too many concurrent connections are open. In all cases, it is a time-consuming exercise to create and tear down a connection.

To efficiently use a limited number of concurrent connections and reduce the overhead of creating and closing physical connections, a common approach is to rely on a connection pool. Connection pools can be used in either a standalone application or an application running inside a Java EE container.

Note: The pooling mechanism is useful in managing not only JDBC connections, but also any scarce resource whose creation and release have a high overhead.

You code individual clients to create a connection, use it, then immediately tear it down. In reality, the create operation is a `create` operation, and the tear down operation is a `close` operation. This approach enables efficient use of multiple connections, without the need for cooperation between the clients of the connections, nor any need for an excessive number of real connections.

In addition to the standard JDBC API, higher-level APIs and components are available to simplify the integration with relation to databases. Examples of these include the Java Persistence API (JPA), TopLink and Hibernate. All these technologies provide a object-relational mapping mechanism that provides an object view to the relational data maintained in a database. In Java EE platforms, JPA is the preferred technology to implement object-relational mapping.

ORM Frameworks

- There is an “impedance mismatch” between object data and relational data.
- Software must be used to:
 - map objects and associations to tables and keys.
 - perform CRUD operations on these data structures.
- There are several products that do this:
 - Oracle TopLink
 - EclipseLink
 - Hibernate
- Usual options are to build or buy. “Rolling your own” ORM framework may be a poor use of your time.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

TopLink is an advanced, object-persistence and object-transformation framework that provides development tools and run-time capabilities that reduce development and maintenance efforts, and increase enterprise application functionality.

TopLink builds high-performance applications that store persistent object-oriented data in a relational database. It successfully transforms object-oriented data into either relational data or Extensible Markup Language (XML) elements.

The Eclipse Persistence Services Project, more commonly known as EclipseLink, is a comprehensive open source persistence solution. EclipseLink was started by a donation of the full source code and test suites of Oracle's TopLink product.

EclipseLink 2.0 supports JPA 2.0 and also offers many advanced features.

EclipseLink's services currently include object-relational with JPA, object-XML binding in MOXy (with support for JAXB), and a Service Data Objects (SDO) implementation sharing a common mapping core. EclipseLink's most popular persistence service is dealing with relational databases through JPA.

EclipseLink is suitable for use with a wide range of Java Enterprise Edition (Java EE) and Java application architectures (see EclipseLink Application Architectures).

TopLink Grid offers an innovative integration between EclipseLink JPA and Oracle's Coherence product which supports a range of configurations from the use of Coherence as a distributed L2 Entity cache to the use of Coherence as the primary data source.

As an L2 cache, Coherence supports caching very large numbers of Entities through the aggregation of all cluster member heaps. As a data source, TopLink Grid will direct all read, write, update, and query operations to Coherence. This configuration supports highly available and responsive JPA applications with asynchronous database write behind.

Hibernate: Historically, Hibernate facilitated the storage and retrieval of Java domain objects via Object/Relational Mapping. Today, Hibernate is a collection of related projects enabling developers to utilize POJO-style domain models in their applications in ways extending well beyond Object/Relational Mapping. It is the default implementation for JBoss and Spring.

OpenJPA: OpenJPA is Apache's implementation of Sun's Java Persistence 2.0 API (JSR-317 JPA 2.0) specification for the transparent persistence of Java objects.

What Do You Think?

- For an ORM solution, is it better to build it or buy it and why?
- Have you been involved in building an ORM solution? What were the benefits of a “build it” approach?
- Approximately what percentage of project time and resources are spent on configuring and using the ORM solution?
- Do you use an ORM solution for 100% of the data integration? If not, then what percentage and why?

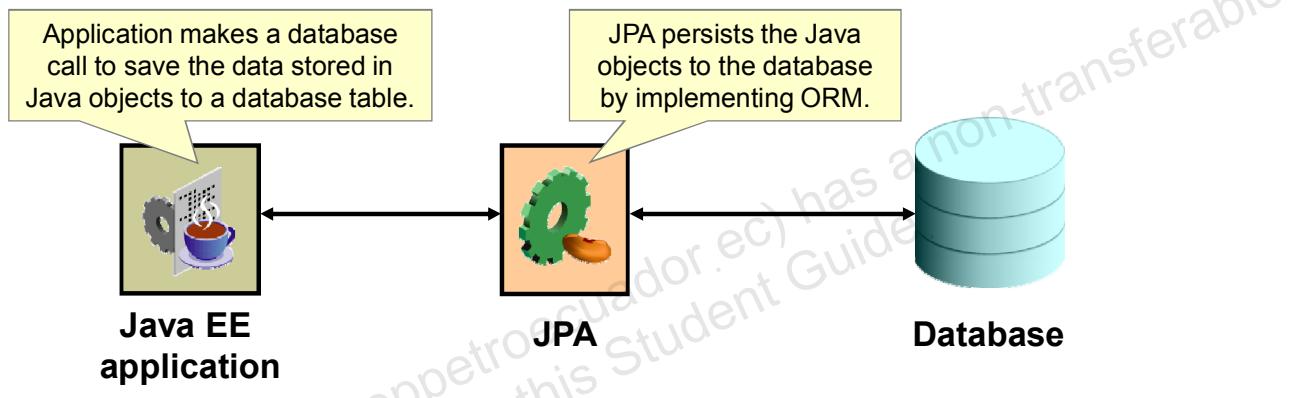


ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Overview of Java Persistence API (JPA)

- JPA is a data persistence framework defined as a part of the Java EE 6 specification.
- JPA defines a standard for:
 - The ORM configuration metadata
 - The EntityManager API
 - The Java Persistence Query Language (JPQL)



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The Java Persistence API (JPA) is a part of the Java Platform, Enterprise Edition 6 (Java EE 6) specification. It greatly simplifies Java persistence and provides an object relational mapping (ORM) approach that enables you to declaratively define how to map Java objects to relational database tables in a standard, portable way.

The JPA also defines:

- Standards for creating the ORM configuration metadata for mapping entities to relational tables
- The EntityManager API for performing CRUD (create, read, update, and delete) and persistence operations for entities
- The Java Persistence Query language (JPQL), for searching and retrieving persisted application data

Note: One of the significant new features introduced in JPA 2.0 is the Criteria API, an API for dynamically constructing object-based queries.

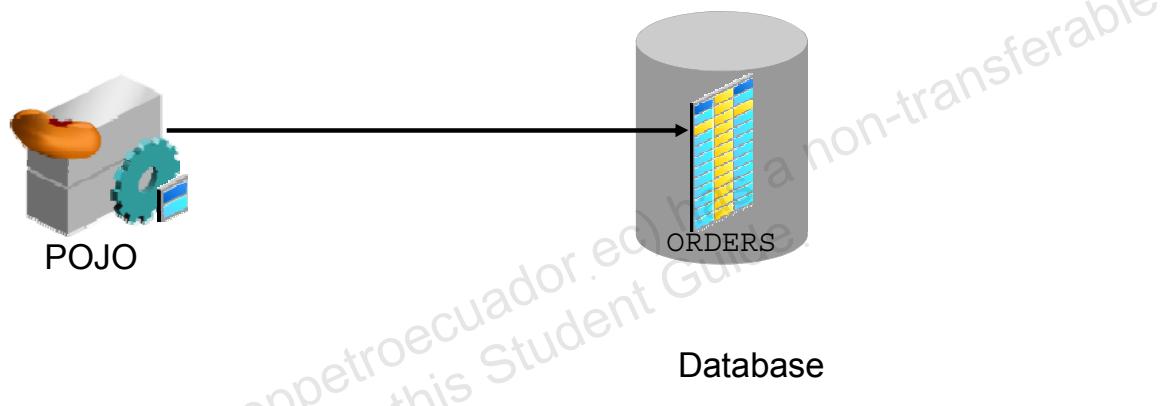
Entities are persistence objects that model persistent data stored in a relational database.

JPA can be used both inside and outside the Java EE environment.

What Are JPA Entities?

A Java Persistence API (JPA) entity is:

- A lightweight object that manages persistent data
- Defined as a Plain Old Java Object (POJO)
- Not required to implement interfaces
- Mapped to a database by using annotations



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

JPA 2.0 is a part of the Java EE 6 specification that simplifies Java persistence to a database. It provides an ORM approach that enables you to declaratively define how to map Java objects to relational database tables. JPA works both inside a Java EE application server, and in a Java Standard Edition (Java SE) application.

A JPA entity, or simply, “entity”:

- Manages persistence data
- Has fields that are mapped to columns in a relational database table by using annotations
- Does not require that any interfaces be defined

Using JPA, you can designate any POJO class as a JPA entity by using annotation. An entity is not required to implement any interfaces.

Messaging Systems

A messaging system is a system that:

- Uses messages to communicate between components (rather than direct method invocations)
- Enables loosely coupled components
- Is peer-to-peer in nature



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Messaging systems have existed for many years in different forms. A messaging system uses messages rather than method invocation (for example, Remote Method Invocation [RMI]) to communicate between components. In this way, components are loosely coupled and do not need to know details of one another's implementation or interface. Instead, each peer client communicates with a message queue to send or receive messages.

Note: Email systems are not considered messaging systems because they are used for communication between humans or between a system and a human, and not strictly between systems.

Message-Oriented Middleware

- Message-oriented middleware refers to an infrastructure that supports messaging.
- Typical message-oriented middleware architectures define these elements:
 - Message structure
 - The way to send and receive messages
 - Scaling guidelines



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Message-oriented middleware has been in use for several decades. In the mid-1980s, when providers created architectures that could operate in a standard way on a variety of platforms, message-oriented middleware became widely used.

These providers made inroads into bridging the gap between the many platforms for mainframes and those for personal computers. Today, hundreds of companies position themselves as middleware firms. Even though there is intense competition and variety in message-oriented middleware products, they all tend to fall into one of these categories:

- Publish/subscribe
- Point-to-point
- Request-reply

Messaging Models

JMS supports point-to-point and publish/subscribe messaging models. The models are very similar, except for the following:

- The PTP messaging model enables delivery of a message to exactly one recipient.
- The pub/sub messaging model enables delivery of a message to multiple recipients.

Java Message Service

Java Message Service (JMS):

- Is a Java EE standard for developing messaging systems
- Provides an architecture for loosely coupled integration of systems through the use of messages
- Supports both point-to-point and publish/subscribe models for message delivery
- Is asynchronous in nature
- Enables reliable communication

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

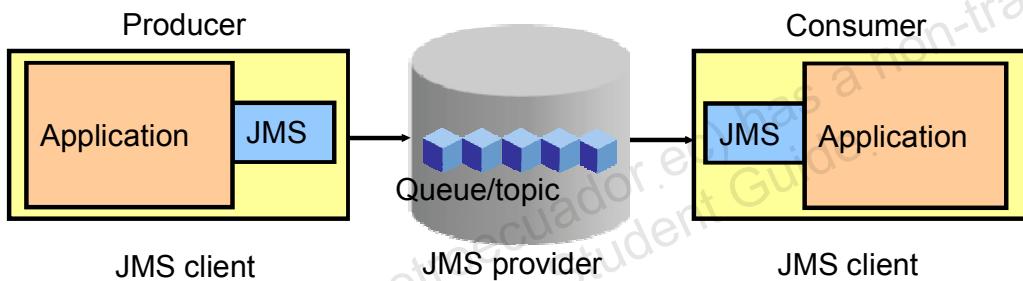
The JMS API is an integral part of the Java EE platform and is its standard for developing messaging systems. Before JMS, most messaging systems implemented only one of the two messaging models: point-to-point or publish-subscribe. JMS supports both models and a Java EE provider must implement both.

Note: JMS can emulate synchronous communication as well, but this practice is not recommended.

JMS Application Architecture

The main components of a JMS application:

- JMS clients (producer and consumer)
- JMS provider
- Administered objects
 - Connection factories
 - Destinations (queues or topics)



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

There are different participants or components in a JMS application.

A JMS client is any Java application that participates in a JMS environment. A JMS client can receive and produce messages.

- A JMS producer produces (sends) messages and is also known as a “publisher.”
- A JMS consumer receives (requests) messages and is also known as a “subscriber.”

A JMS provider handles the routing and delivery of messages. Oracle provides a JMS provider in its application server and database server.

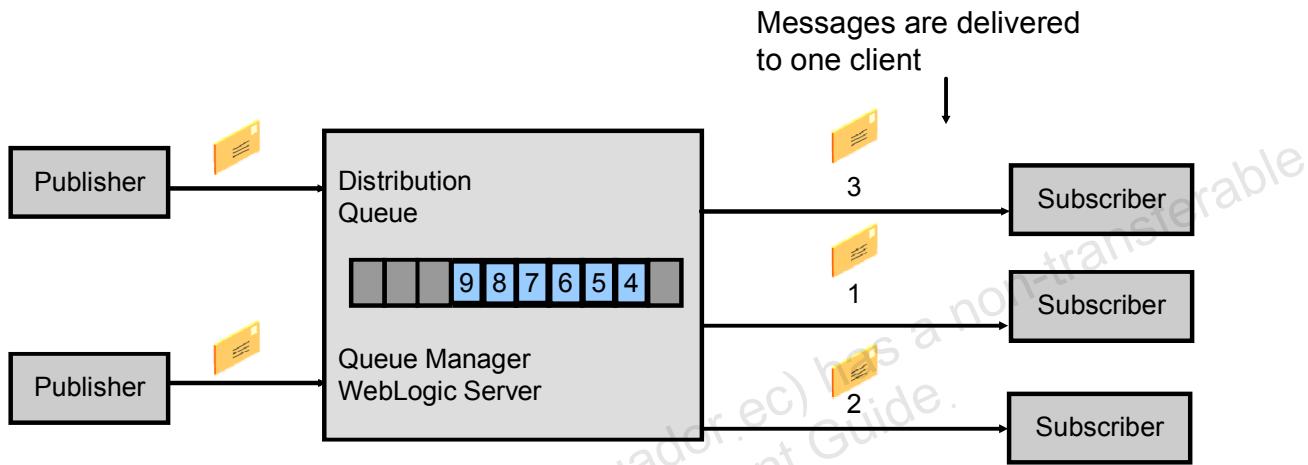
Administered objects, stored in the JNDI namespace of a Java EE container, consist of:

- **Connection factories:** Used by a client to create a connection with a provider
- **Destinations:** Targets of messages that are produced, or sources of messages that are consumed by client applications, such as queues for the JMS Point-To-Point model and topics for the JMS Publish/Subscribe model

A JMS application generally contains one JMS provider and more than one JMS client. The messages are sent through a messaging system called the message broker (or MOM services). OC4J JMS, Oracle JMS, BEA WebLogic JMS, Sun Microsystems iPlanet Message Queue, and IBM MQSeries are examples of JMS providers.

Point-to-Point Queue

Many producers can serialize messages to multiple receivers in a queue.



ORACLE

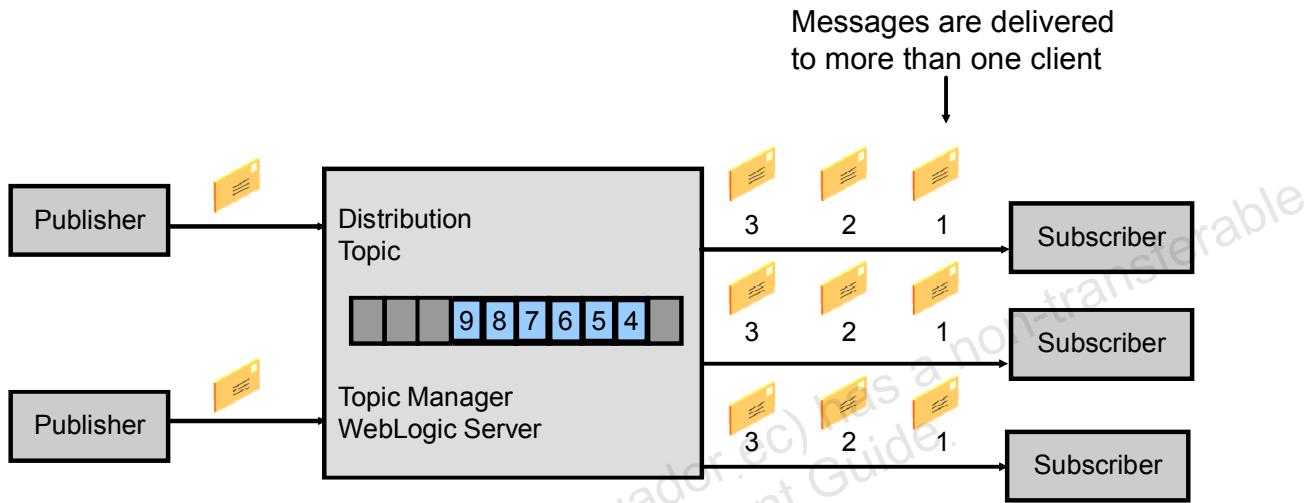
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

When using a PTP queue, multiple producers can put messages onto the queue. The queue serializes the messages into a linear order. Multiple receivers can take messages off the queue; the messages typically come off in a first-in, first-out (FIFO) order; the oldest message on the queue is the first one to be taken off.

A message can be delivered only to one receiver. An example of when to use a PTP queue would be at a call center. Calls are routed into the network through a PBX. The PBX places incoming calls onto an "Incoming Call" queue. When a service representative becomes available, the representative requests the next caller in the system. The system pulls the caller who has been waiting the longest off the queue and routes the caller to the service representative.

Publish-Subscribe Topics

Publishing and subscribing to a topic decouples producers from consumers.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Having the publishers publish to a topic rather than directly to a list of subscribers decouples the publishers and subscribers.

By doing this, a publisher is not required to manage the number of subscribers (if any) that must receive the message. By delegating the work of the message delivery to the message-oriented middleware server (which manages the topic), the publisher does not have to manage the delivery of guaranteed messages, fault tolerance of its production, load balancing, or other issues. By decoupling a subscriber from the publisher, the subscriber does not have to determine whether its publisher is active. If the message-oriented middleware server is executing, the needs of both publishers and subscribers are met.

An example of using a publish and subscribe topic is a stock ticker application. A typical system would set up a topic for each stock that is traded on the exchanges. When a trade is made on a stock, the respective exchange publishes a message to the topic associated with the stock traded. Clients who are interested in receiving updates about the status of their stocks use a program to subscribe to the topics of each stock they are interested in. When it receives a message, the message-oriented middleware server broadcasts the message to all interested stock ticker programs.

WebLogic Server JMS Features

WebLogic Server JMS supports:

- PTP and pub/sub domains
- Guaranteed and transactional message delivery
- Durable subscribers
- Distributed destinations
- Recovery from failed servers



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The WebLogic Server implementation of JMS fully supports the PTP and publish and subscribe domains of messaging middleware.

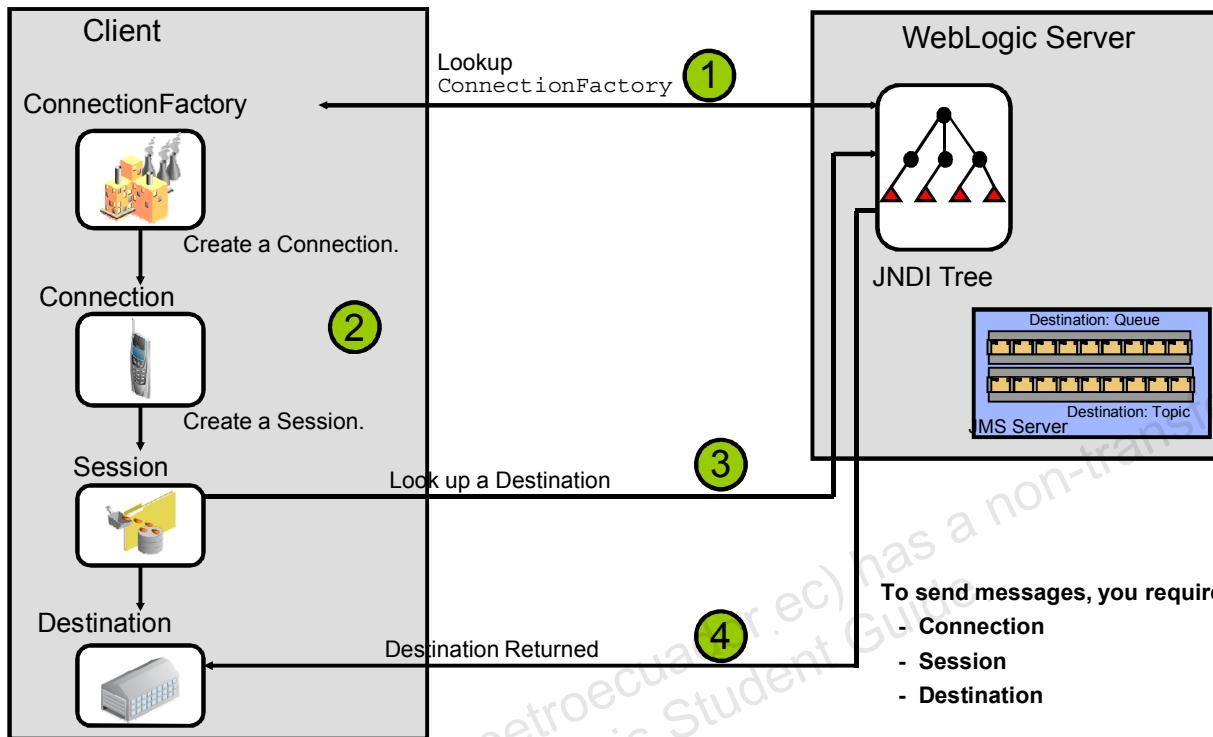
WebLogic Server also provides ACK-based guaranteed message delivery (GMD). ACK-based GMD persistently stores messages until the receiver of the message issues an acknowledgement reply. WebLogic Server JMS uses its built-in support for JDBC and JDBC connection pools to persist JMS messages in a connection pool database.

Transactional message delivery gives the developer the ability to put a JMS session into a transaction context. Sessions that are located in a transaction context “buffer” the messages that they produce and do not transmit the messages until the transaction is committed. After the transaction is committed, the messages that the session has buffered are transmitted to their respective destinations. A session can optionally roll back the transaction, which has the transaction “drop” the messages it had previously buffered.

WebLogic Server allows clients to register themselves as a durable subscriber. A durable subscriber is a client that expects to receive all persistent (GMD) messages sent to a particular destination, whether the client is currently executing or not. If the durable subscriber is not currently executing, WebLogic Server stores the messages in a database until the durable subscriber reactivates and retrieves the stored messages.

Note: WLS implements the JMS 1.1 specification.

JMS Architecture: Connecting



ORACLE

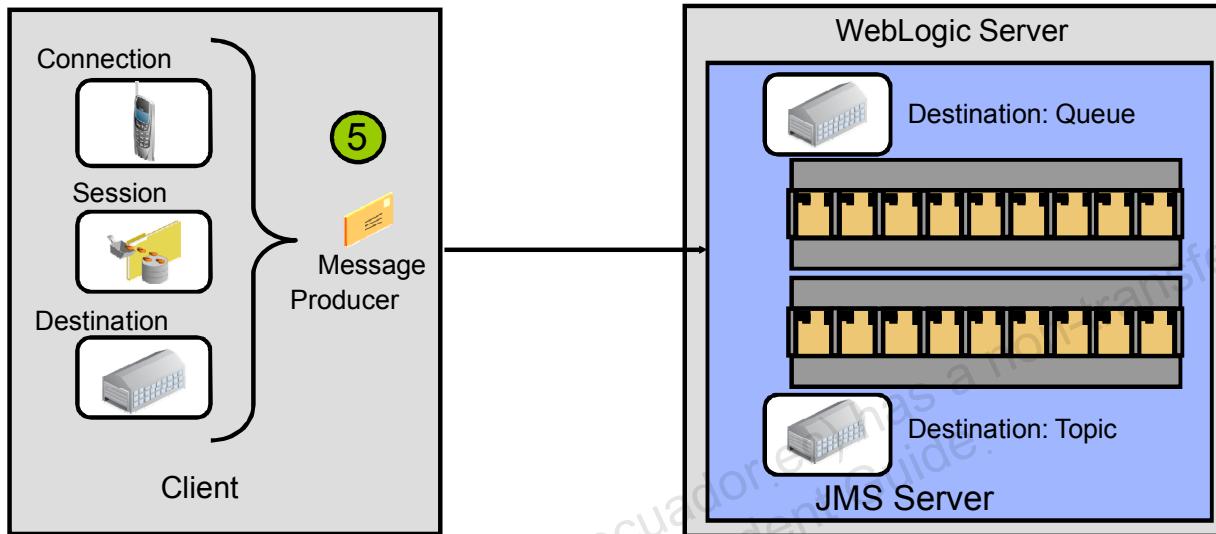
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The Connection Factory is a lightweight object stored on a JNDI tree used to create connections to destinations. Connection is a communication link to the JMS Server used to create sessions. Session is used to create senders, receivers, and empty messages. Session is also used to demarcate transactions. Destination, a lightweight object stored on JNDI, is the target for the messages.

The JMS API provides interfaces for the production and consumption of messages. Before JMS 1.1, point-to-point messaging and publish/subscribe messaging were separated into two different domains with different interfaces. JMS 1.1 now unifies the interfaces, and new development should follow the unified API interface.

JMS Architecture: Sending Messages

Connection, Session, and Destination are used to send a message.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

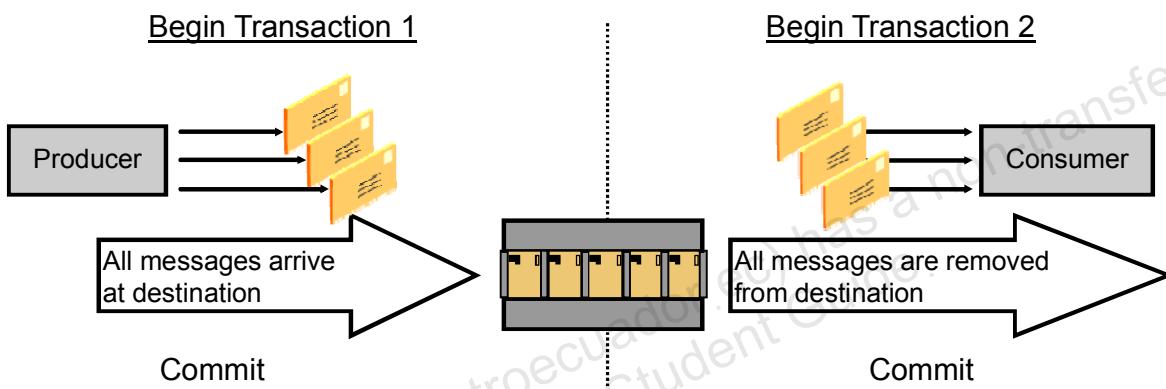
A Message Producer is used to send any type of message to a destination. A Message receiver is used to receive messages from a destination.

The unified interface classes in the javax.jms package are:

- **Connection**: Created from a **ConnectionFactory**
- **Session**: Created from a **Connection** for producers and consumers to interact
- **MessageProducer**: Created from a **session** to produce messages on a **Destination**
- **MessageConsumer**: Created from a **session** to consume messages from a **Destination**
- **Destination**: **Destination** for messages (for example, queue or topic)

Transacted Messaging

- A JMS client can use JTA to participate in a distributed transaction.
- Alternatively, a JMS client can demarcate transactions local to the JMS Session, through a transacted session.
- Participation in a transaction is optional.



ORACLE

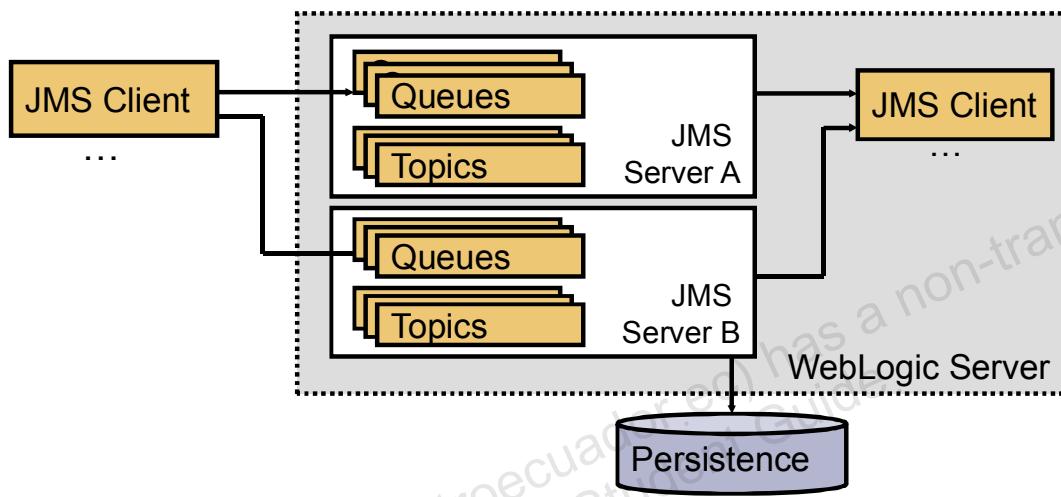
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

JMS clients can participate in a distributed or local transaction. There are two scenarios:

- On the Producer side, a transaction begins and some operations, such as sending messages, are performed. If the transaction commits, all the messages are sent to the destination. If the transaction rolls back, none of the messages arrive at the destination.
- On the Consumer side, a transaction begins and some operations, such as processing messages, are performed. If the transaction commits, the processed messages are removed from the destination. If the transaction rolls back, the messages stay in the destination.

WebLogic Server JMS Server

- In WLS, the messaging service is implemented through a JMS server.
- A JMS server receives and distributes messages.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A JMS server is configured within the administration console and targeted to a server. There can be more than one JMS server targeted to the same WebLogic Server instance.

Connection Factory

- A connection factory:
 - Encapsulates connection configuration information
 - Is used to create preconfigured connections
 - Is stored in JNDI
 - Can be targeted to servers or clusters
- WLS provides a default connection factory that is bound in JNDI to `weblogic.jms.ConnectionFactory`.
- When a new configuration is required, a new connection factory can be created.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A `ConnectionFactory` object encapsulates connection configuration information and enables JMS applications to create a `Connection`. A system administrator configures connection factories to create connections with predefined attributes.

A system administrator defines and configures one or more connection factories, and WebLogic Server adds them to the JNDI space during startup. The application then retrieves a connection factory using WebLogic JNDI.

The system administrator can establish clusterwide, transparent access to destinations from any server in the cluster by configuring multiple connection factories and using targets to assign them to WebLogic Servers. Each connection factory can be deployed on multiple WebLogic Servers.

WebLogic JMS defines one default connection factory. It can be looked up using the JNDI name, `weblogic.jms.ConnectionFactory`. You must define a connection factory only if the default provided by WebLogic JMS is not suitable for your application.

JMS Destination

- A JMS destination is a lightweight object stored in JNDI.
- It is the target on a JMS server for sending or receiving messages.
- The JMS destination types are:
 - Queue
 - Topic



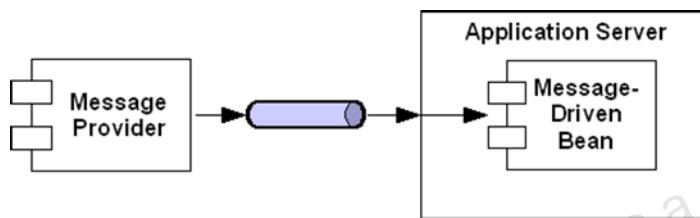
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A JMS destination identifies a queue (point-to-point) or topic (publish/subscribe) for a JMS server.

After configuring a JMS server, configure one or more queue or topic destinations for each JMS server. You configure destinations explicitly or by configuring a destination template that can be used to define multiple destinations with similar attribute settings.

Message Driven Beans (MDB)

- An MDB is an asynchronous message consumer that is deployed in an application server.
- An MDB can be used to asynchronously consume messages from a variety of message providers.



ORACLE

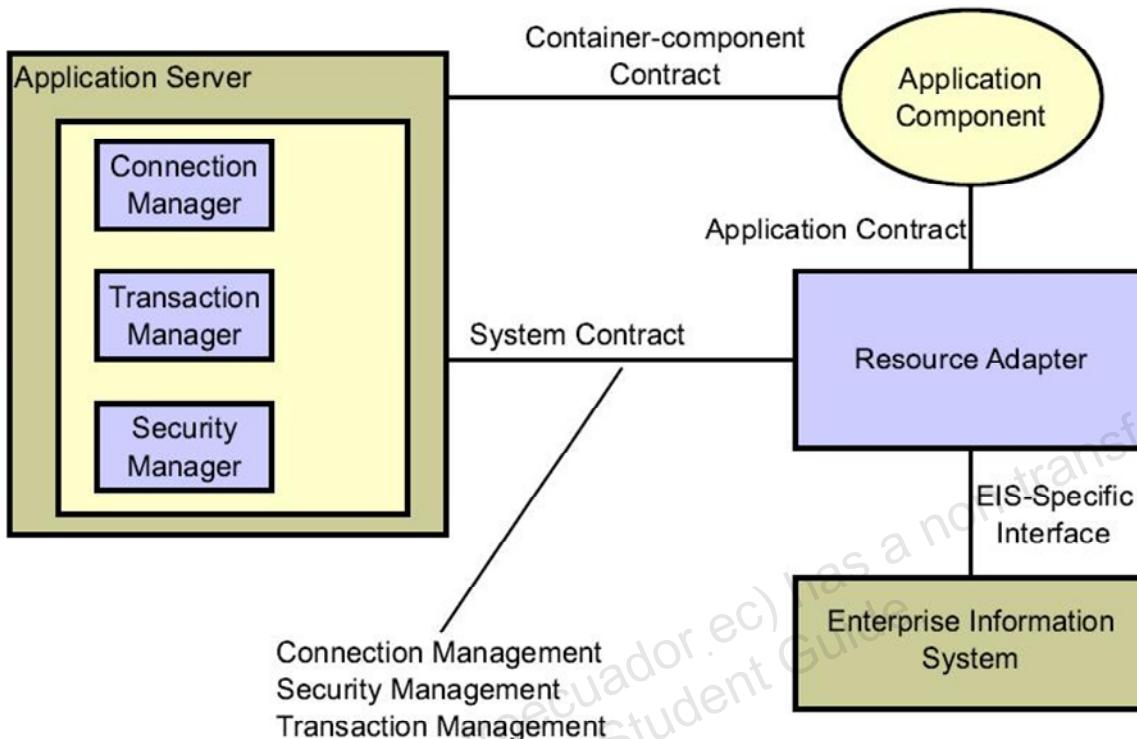
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

An MDB is an asynchronous message consumer that is deployed in an application server, as shown in . An MDB can be used to asynchronously consume messages from a variety of message providers, such as a Java EE Connector Architecture resource adapter, which is primarily used to work with JMS.

Compared to standalone message consumers, MDBs can significantly simplify the design and implementation of message containers in the following areas:

- The application server's MDB container manages the life cycle of the MDB instances
- The application server allows you to configure the message destination from which the MDB consumes the messages
- Advanced features, such as transaction can be incorporated into MDBs declaratively

Java EE Connector Architecture (JCA)



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The Java EE Connector Architecture (JCA) connections provide Java EE technologies access to EIS resources with the use of a non-proprietary API. These resources can be as varied as ERP systems, transactional mainframe resources, and legacy applications that are not based on Java technology.

The concept of the Java EE Connector Architecture is illustrated in the slide. The Java EE Connector Architecture resource adapters provide access to the EIS resource with the use of the specific interface that is provided by the resource vendor. Through its connection, transaction, and security manager services, the application server controls access to the Java EE Connector Architecture adapter and connections.

To simplify the application component's access to the resource adapters, the Java EE Connector Architecture also provides a common-client interface (CCI) to implement the application contract. CCI provides a standard interface that is consistent with the programming models used to access other resources, such as JDBC and JMS.

When you use the Java EE Connector Architecture, you eliminate the need for application server vendors to code to particular EIS resource providers. Instead, the EIS vendor can provide a standard resource adapter that implements the Connector SPI.

The adapter can be plugged into any application server that supports the Java EE Connector Architecture API, and application components can request Java EE Connector Architecture from the server.

Application components use the Java EE Connector Architecture connections through the Java EE Connector Architecture API. Because of the pluggable nature of the Java EE Connector Architecture, architects no longer need to use resource connectivity as a selection criterion for their application servers.

As an example, consider a situation in which a business method implemented in an EJB component needs to be invoked based on a database trigger. The standard JDBC API does not provide the support to triggers. On the other hand, a JCA 1.5 – compliant resource adapter allows you to send an inbound message to the EJB container, which can be used to invoke the business method. Furthermore, the transaction and security context can also be propagated using the resource adapter. Because of this, the Java EE Connector Architecture provides additional flexibility compared to other technologies, such as JDBC and JMS, and an increasing number of database and JMS product vendors provide a JCA-based connection mechanism.

Integration Technologies

- Java Web Services:
 - JAX-WS
 - JAX-RS
 - WSIT
- Distributed Technologies:
 - RPC
 - CORBA
 - RMI



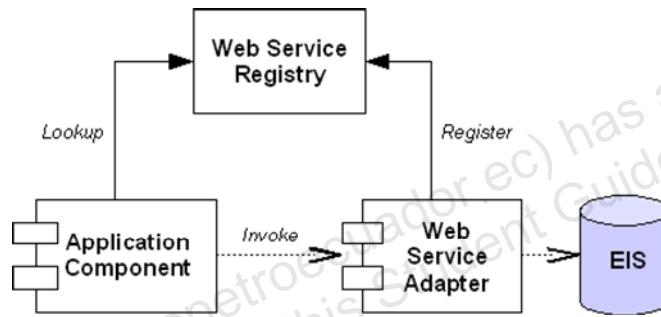
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

CORBA: CORBA, or Common Object Request Broker Architecture, is a standard architecture for distributed object systems. It allows a distributed, heterogeneous collection of objects to interoperate. CORBA defines an architecture for distributed objects. The basic CORBA paradigm is that of a request for services of a distributed object. Everything else defined by the OMG is in terms of this basic paradigm. The services that an object provides are given by its interface. Interfaces are defined in OMG's Interface Definition Language (IDL). Distributed objects are identified by object references, which are typed by IDL interfaces. Pros: language independence. Cons: few services have been implemented, integration issues with Microsoft DCOM and competition from RMI and web services.

Java Remote Method Invocation (Java RMI): enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism. Java RMI is included with Java SE and is available as a separate download for Java ME.

Web Services

- Technical Foundations of Web Services:
 - eXtensible Markup Language (XML) and XML Schema
 - Simple Object Access Protocol (SOAP)
 - Web Service Description Language (WSDL)
 - Universal Description, Discovery and Integration (UDDI)
 - Standard internet protocols
- Web Services for Integration



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A web service is designed to expose business logic that can be accessed by any application. It is a software component, identified by a Uniform Resource Identifier (URI), whose interfaces and bindings can be described by standard Extensible Markup Language (XML) vocabularies.

A web service supports direct interaction with other software applications or components and is network accessible. With a web service, you can make your applications programmatically accessible to other applications over the Internet. Web services establish a method for standardizing communication, making it easier for applications to share information. For example, in the case of a merger or an acquisition, companies can avoid investing large sums of money on developing software to integrate the systems of the different companies. The information systems of different companies can be linked by extending each company's business applications as Web services. These business systems can then be accessed by using simple SOAP messages over the normal HTTP protocol.

Web services have gained significant industry momentum due to its independency of operating platforms and programming languages. In EIS integration, a web service endpoint can be implemented as an adapter to a legacy application or business process. The RPC-like interface exposed by this web service adapter allows application components to gain access to the EIS resource.

To further decouple the application component and the Web service adapter, a Web service registry can be used to allow dynamic registration and discovery of the services. illustrates this integration strategy.

Web services are based on the following industry standards:

- The eXtensible Markup Language (XML) and XML Schema: Provide the standard language for Web service description and communication
- Simple Object Access Protocol (SOAP): Defines the standard message format used in Web service communication
- Web Service Description Language (WSDL): Defines the standard format for describing the Web service interface
- Universal Description, Discovery, and Integration (UDDI): Defines the standard message format and APIs for web service providers to register/publish, and web service consumers to find/browse web service descriptions.
- Standard Internet Protocols: Provide a standard transportation mechanism for web service communications. Conceptually, a web service can be designed to use any application-level protocols as its transport. In practice, however, HTTP and HTTPS are most commonly used.

Java technology provides full-support for implementing both web service clients and endpoints. The core APIs for designing web services in Java EE include JAX-RPC and JAX-WS.

Java Web Service Technologies

- Java API for XML-based Web Services (JAX-WS):
 - A follow-up of JAX-RPC
 - Simplifies the implementation of web service endpoints and clients
 - Both provides APIs for implementing web service clients and endpoints
 - Web service endpoints can be implemented as a stateless session bean or a POJO
- Java API for RESTful Web Services (JAX-RS)
 - Provides support in creating web services according to the Representational State Transfer (REST) architectural style.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Java Web Services are web-based applications that use open, XML-based standards and transport protocols to exchange data with clients. Web services are developed using Java Technology APIs and tools provided by an integrated Web Services Stack called Metro. The Metro stack consisting of JAX-WS, JAXB, and WSIT, enable you to create and deploy secure, reliable, transactional, interoperable web services and clients. The Metro stack is part of Project Metro and as part of GlassFish, Java Platform, Enterprise Edition (Java EE), and partially in Java PlatForm, Standard Edition (Java SE). GlassFish and Java EE also support the legacy JAX-RPC APIs.

The Metro Web Services stack delivers secure, reliable, transactional interoperability between Java EE and .Net 3.0 to help you build, deploy, and maintain Composite Applications for your Service Oriented Architecture. Metro provides ease-of-development features, support for W3C and WS-I standards such as SOAP and WSDL, asynchronous client and server, and databinding through JAXB 2.0.

WSIT is an implementation of a number of open web services specifications to support enterprise features. In addition to message optimization, reliable messaging, and security, WSIT includes a bootstrapping and configuration technology.

Java API for XML-based Web Services (JAX-WS)

- One of the primary goals of JAX-WS is to simplify the implementation of Web service endpoints and clients using new features, such as annotations.
- JAX-WS relies on the Java API for XML Binding (JAXB) 2.2 for data binding between XML and Java objects.
- Finally, JAX-WS supports the raw XML over HTTP messaging feature that can be used to implement REST (REpresentational State Transfer)-style web services.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

JAX-WS defines a Java API for implementing and accessing SOAP-based remote procedure calls (RPC). JAX-WS hides the complexity of SOAP, because neither the web service implementation nor the web service client needs to be coded to manipulate SOAP messages. JAX-WS does not require the web service endpoint and client to be implemented in Java technology. The Java-WSDL/XML mapping allows a web service implemented in JAX-RPC to expose a WSDL to web service clients. The WSDL/XML-Java mapping allows a JAX-WS client to access any web service through the interface mapped from the web service's WSDL-based description.

Note: In Java EE, a stateless session bean can be deployed as a web service endpoint. This allows the web service implementation to declaratively utilize advanced services provided by the application server, such as transaction management.

Comparing Integration Technologies

	JMS/MDB	JCA	Web Services
Integration Style	Rely on a message-oriented middleware	Typically rely on vendor-provided resource adapters	Implemented as an adapter layer
Coupling	Loosely coupled between application components and EIS	Tightly-coupled	Loosely-coupled
Interaction Pattern	Asynchronous	Both synchronous and asynchronous (vendor dependent)	Synchronous or Asynchronous



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

JMS, JCA and web services are all viable options for system integrations. It is possible that a solution based on any one of these technologies can solve the same problem. In this case, which one is the most appropriate option depends on the following factors:

Integration Style

Many organizations have adopted message-oriented middleware (MOM) products for EIS integration. JMS provides the most straightforward integration solution with enterprise Java applications if the MOM product is JMS compliant. For EIS resources that do not support the message-based integration mechanism, you can rely on a resource adapter based on JCA or web services. If the vendor of the product provides such an adapter, the interrogation is relatively easy. Otherwise, it typically requires significant efforts to develop such an adapter.

Coupling Between Application Components and the Resource

The coupling is the tightest for JCA-based solutions, due to its API specifications. Because of this, JCA is typically used as a mechanism to implement application components in the business tier. On the other hand, JMS and web services both support a loosely-coupled approach. In message-based implementation, the MOM product separates application components and the resource, and the dependency is typically at the message format and

Comparing JMS, JCA, and Web Services

	JMS/MDB	JCA	Web Services
Transaction	MDB can use declarative transaction features, and initiate a distributed transaction	Available, depending on the adapter implementation	Implementation of the Web service endpoint can be transactional. Additional transaction features are vendor-specific
Security	Vendor-specific	Resource adapter implementation allows transparent integration	Depends on vendor's implementation of Web service security features for additional security requirements



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

type level. Web services can also be used to design a loosely-coupled solution, because it removes the platform and programming dependency using XML-based standards.

The Interaction Pattern

The interaction between an application component and the EIS resource can be synchronous or asynchronous. JMS is the obvious choice for asynchronous interactions between application components and the EIS resource using a message-driven middleware. Depending on the implementation of the JCA adapter, it can also support asynchronous messaging. Although you can build asynchronous web services using one-way calls, it is not directly supported in Java EE. For synchronous interactions, web services and synchronous JCA adapters are readily applicable.

Transaction Support

Distributed transaction features can be built into the implementation of the JMS provider and JCA resource adapters, and the JMS provider and JCS resource adapter can enlist them as transactional resource managers to the application container. This allows application components to be designed and implemented with a consistent transaction model either programmatically or declaratively.

Support levels for transaction support in web services is vendor specific. In a Java EE application, web service endpoint implementation can be transactional. However, the distributed transaction support depends heavily on the vendor's implementation of the web service transaction standards, such as Web Service Coordination (WS-Coordination) and Web Services Atomic Transactions (WS-AT) specifications defined by the Web Service Transaction (WS-Transaction) committee of the Organization for the Advancement of Structured Information Standards (OASIS).

Note: The Web Service Interoperability Technology (WSIT) developed by the open source Project Tango implements both WS-Coordination and WS-AT specifications. For more information on WSIT, visit the web site <https://wsit.dev.java.net>.

Security

Security is one of the least standardized areas in Java EE. For example, there are no specific security requirements for a JMS provider, and it is up to the JMS provider to implement features, such as controlled access to JMS resources and message security. The security contract in JCA provides a standard interface for the resource adapter vendor to implement the security capability. However, the implemented security level depends on EIS resource types. In Java EE, the security of web services is typically supported by the security services provided with the application server, such as authentication, and HTTPS for encryption. However, besides these essential security services, web services security can be more involved. For example, when a client invokes a web service, the request message can pass through multiple intermediate nodes before it reaches the endpoint. To protect the portion of the message that is intended for the endpoint, you must use some message-level security mechanism to implement an end-to-end solution. The support for message-level security in web services depends on whether the application server vendor implements the features defined by specifications such as Web Services Security (WS-Security).

Lesson Agenda

- Examining Enterprise Information System integration
- Reviewing Java integration technologies
- **Applying Integration tier patterns**
- Examining Service-Oriented Architecture (SOA)



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Applying Integration Tier Patterns

Integration tier patterns allow you to achieve the following goals:

- Reduce the coupling between application components and resources.
- Simplify the access to the resources.
- Gregor Hohpe, lists 65 different enterprise integration (EI) patterns.



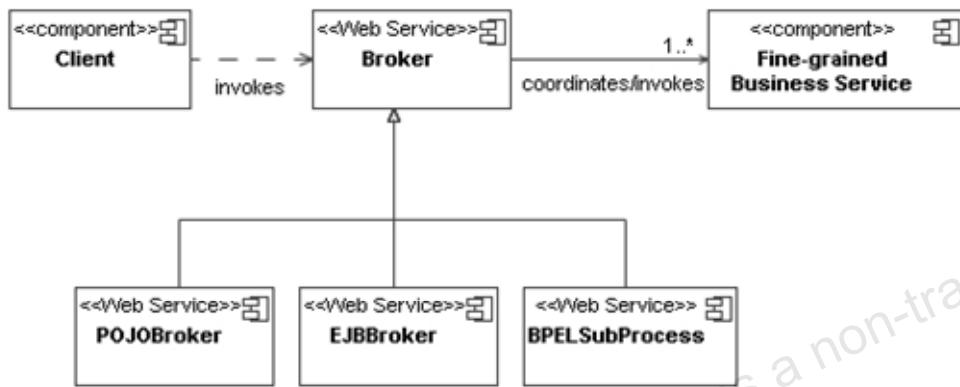
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Gregor Hohpe, in his book Enterprise Integration Patterns, lists 65 different integration patterns. These fall into the following categories:

- Integration Styles describe different ways applications can be integrated.
- Channel patterns describe the basic attributes of a messaging system.
- Message Construction pattern describes the intent, form and content of messages that travel across the messaging system.
- Routing patterns discuss mechanisms to direct messages from a sender to the correct receiver.
- Transformation patterns change the information content of a message.
- Endpoint patterns describe the behavior of messaging system clients.
- System Management patterns provide the tools to keep a complex message-based system running.

Integration Tier Patterns

Web Service Broker



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Similar to the Session Façade pattern, a web service broker, as shown in the slide, provides a coarse-grained interface to access high-level business functionality. The web service broker receives the request message sent from the client, performs necessary parsing and processing to coordinate the execution of distinct business service components, and sends the response back to the client. The web service broker can also work as a mediator to enforce controlled access to business service components.

The Web Services Business Process Execution Language (BPE, WS-BPEL, or BPEL4WS) allows you to define a business process that can be used to orchestrate the invocation of Web services. For more information on the BPEL specification, visit the website <http://www.oasis-open.org/specs/index.php#wsbpelv2.0>.

Also, the NetBeans Enterprise Pack allows you to design BPEL business processes graphically.

Aspect Oriented Programming

- Aspect-Oriented Programming (AOP) provides another way of thinking about program structure.
- AOP introduces aspects. Aspects support modularization of concerns that cut across multiple types and objects called “crosscutting concerns.”
- Transactions, security, and logging are examples of cross-cutting concerns.
- Weavers combine the core classes and crosscutting aspects into the final system.
- AspectJ is a seamless aspect-oriented extension to the Java programming language.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

In computing, aspect-oriented programming (AOP) is the realization that there are issues or concerns that are not well captured by traditional programming methods. Problems such as logging and security cut across the applications. Rather than repeat support and coding for these concerns, AOP is one way to address these issues.

AOP is driven by separation of concerns. It enables abstraction and modularization that is orthogonal to what OOP provides. AOP is not a style, it is a specific mechanism used to provide support for aspects.

AOP is a methodology that provides separation of crosscutting concerns by introducing a new unit of modularization—an aspect. And for an aspect weaver that composes the final system by combining the core classes and crosscutting aspects through a process called weaving.

AOP requires preprocess/pre-compiling.

XML Services

- Java API for XML Processing (JAXP) enables applications to parse, transform, validate and query XML documents using an API that is independent of a particular XML processor implementation.
- Java Architecture for XML Binding (JAXB) provides a convenient way to process XML content using Java objects by binding its XML schema to Java representation.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The Java API for XML Processing, or JAXP for short, enables applications to parse and transform XML documents using an API that is independent of a particular XML processor implementation. JAXP also provides a pluggability feature which enables applications to easily switch between particular XML processor implementations. To achieve the goal of XML processor independence, an application should limit itself to the JAXP API and avoid implementation-dependent APIs and behavior. This may or may not be easy depending on the application. The reason for the existence of JAXP is to facilitate the use of XML on the Java platform. For example, APIs such as DOM Level 2 do not provide a method to bootstrap a DOM Document object from an XML input document, whereas JAXP does. Other parts of JAXP such as the javax.xml.transform portion do not have any other equivalent XSLT processor-independent APIs.

JAXB is an acronym derived from Java Architecture for XML Binding. It constitutes a convenient framework for processing XML documents, providing significant benefits as compared to previously available methods such as the one following the Document Object Model (DOM). In the DOM approach, the parser creates a tree of objects that represents the content and organization of data in the document. The application can then navigate through the tree in memory to access the data it needs. DOM data, however, is contained in objects of a single type, linked according to the XML document's structure, with individual node objects containing an element, an attribute, a CDATA section, etc. Values are invariably provided as strings

Unmarshalling an XML document with the appropriate JAXB method also results in a tree of objects, with the significant difference being that the nodes in this tree correspond to XML elements, which contain attributes and the content as instance variables and refer to child elements by object references.

The JAXB runtime library, supported with the code generated by the Binding Compiler, provides methods for unmarshalling a document from various sources as well as for marshalling a content tree to various destinations. JAXB also supports marshalling and unmarshalling for SAX, the Simple API for XML.

JAXB uses Java's annotations for augmenting the generated classes with additional information that bridges the gap between what is described by an XML schema and the information available (via Java's reflection mechanisms) from a set of Java class definitions. Adding such annotations to existing Java classes prepares them for being used by JAXB's runtime.

Lesson Agenda

- Examining Enterprise Information System integration
- Reviewing Java integration technologies
- Applying Integration tier patterns
- Examining Service-Oriented Architecture (SOA)



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Service-Oriented Architecture (SOA)

SOA is defined as:

- A “paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains”—*Organization for the Advancement of Structured Information Standards (OASIS)* definition
- An IT strategy that organizes the discrete functions contained in enterprise applications into interoperable, standards-based services to be combined and reused quickly to meet business requirements

In computing terms, SOA is a standards-based method of systems development and integration.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

As stated in the slide points, SOA is a strategy for developing and integrating systems through interoperable standards-based services.

The benefits of following an SOA approach include:

- Reuse of functionality across different business organizations and processes
- Integration between different business systems by following a standards-based approach
- Agile development that adapts business changes through discrete functions and standardized interfaces

In summary, in computing terminology, SOA is a system integration process, based on using services developed according to standards that make the process integration easier. The idea of SOA does not prescribe a specific technological implementation. However, industry has embraced the standards, flexibility, and reusability of web service style technology implementations.

Why Is an SOA Approach Required?

An SOA approach of software systems that is structured around the concept of services enables:

- A consumer of the service to be decoupled from the service provided (producer)
- A business process to be decomposed into discrete, reusable functional components (or services)
- The following benefits to be realized:
 - Improve business agility.
 - Align IT with the business.
 - Remove barriers between business units and business partners.
 - Lower the cost of maintaining IT systems.
 - Speed up delivery of applications to meet business demands.
 - Protect IT investments by reusing the existing infrastructure.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

An SOA approach provides guidelines around structuring software systems, based on the idea of a service as the building block. Services provide discrete, decoupled business functionality to be assembled and reused to create applications that support changing business processes.

Aligning IT with discrete business functions can result in rapid development and more reliable delivery of business services. Other benefits of SOA are:

- **Reusability:** Existing business functionality in an application can be reused to meet new business requirements. One of the goals of an SOA approach is to design new services with reusability in mind as determined by its usage patterns within and outside the business domain. However, not all services, such as a large-grained application, are required to be reusable, nor should they be depending on the business requirements.
- **Interoperability:** Communication between services and the business process is not dependent on the platform and are standards enabled. The services are also not tightly coupled to the application.
- **Scalability:** Applications are flexible to the changing business requirements.
- **Cost Efficiency:** This is highly cost efficient as integrating the business resources is standards based.

Ways to Implement Services

Process and service integration can be done in many ways:

- Proprietary implementations: Typically integrate isolated systems with a point-to-point approach
- Vendor-specific implementations: Technology such as Tuxedo enables integration using a consistent approach
- Common Object Request Broker Architecture (CORBA) implementations: Programmatically intensive, widely supported, and not widely used because of implementation costs
- Web Service and now SCA style implementations: Using XML and Web standards to integrate systems

Note: Not everyone agrees on the way to do SOA-enabled implementations.



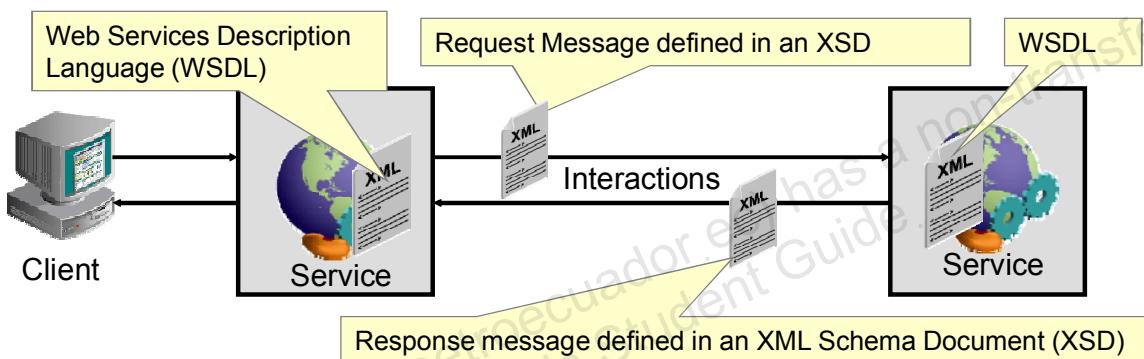
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The points stated in the slide illustrate that integration can be done in many ways and forms, some of which are proprietary approaches that have enabled silo (stand-alone) systems to be integrated, typically in a point-to-point manner. This approach is less flexible and hard to maintain over time. Technology such as Tuxedo, a vendor-specific integration technology among others, enable systems to standardize the way integration is done with their own implementation. Tuxedo does not provide native support for Web services, keeping the technology contained within an organization that embraced it. The computing industry worked to create a standardized protocol and software integration method called Common Object Request Broker Architecture (CORBA) that enabled systems to integrate by using a standard (non-XML-based) interface definition language for generating code templates that could be distributed. Although it is entirely possible to apply an SOA approach with CORBA, the lack of its wide use and accessibility in addition to the remote procedure call mechanisms make it costly to implement and less friendly to use across an Intranet or Internet. With the advent of Web Services and common application of XML, WSDL, and XSD standards, SOA-based approaches have naturally embrace these standards. This results in easier interoperability across Intranet and Internet networks. Keep in mind, not every one agrees on the same implementation styles for web services style SOA implementations. For example, not all Enterprise Service Bus implementations are alike, nor does everyone choose to implement BPEL for process orchestration. However, as we already know time changes many things.

What Are Services?

Services are:

- Building blocks for an SOA-enabled application
- Functionality-described standard interface and message structure definitions
- Accessed using standard protocols (the glue) to enable interoperability from decoupled functions



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Services are business functions that form the foundation for building applications by using an SOA approach. SOA applications can be thought of as functionality that aggregates a collection of related services, by reusing other services to complete automation of a business process.

Service functionality must be described by using standard interface and message structures to make them highly accessible and reusable. While the IT industry has created many ways to do this, each of which is a viable means to accomplish an SOA approach, the most widely embraced are the Web Services standards that include:

- Extensible Markup Language (XML) for metadata
- Web Services Description Language (WSDL) for service interface definition
- XML Schema Documents (XSD) for message structure definition

The benefit of using Web Services artifacts such as WSDL and XSD, which are XML-based documents, is that they are “accepted standards.” These XML document structures are easily exchanged using the standard Internet (Web Services) protocols such as Hypertext Transfer Protocol (HTTP) among others.

The ability to access information and service using standard documents structures, message, and protocols is that they form a consistent layer or glue between decoupled systems that enable high degree of interoperability regardless of the choice of operating system and computer languages used for service implementation.

Adopting Standards for an SOA Approach

Standards are:

- Adopted and used extensively
- Created to enable a high degree of interoperability between systems across heterogeneous environments
- Defined as formalized agreements between different stakeholders of specific implementations
- Required for SOA-enabled systems to be implemented easily and successfully
- Applied to different layers of implementation, such as:
 - Transport protocols
 - Message formats
 - Discovery of services
 - Description of services



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

For something to be considered a standard, it must be widely adopted and used. Then usually a collection of organizations collaborate to formalize the standard. The benefit is that the standards enable vendors to provide tools and products that can interact with each other more easily across heterogeneous environments. Standards enable services to interoperate and integrate functionality across different environments regardless of vendor implementation, operating system, and languages.

For SOA-enabled implementations, standards become even more important to enable a high degree of seamless integration to be possible. Seamless integration is enabled by employing a range of standards applied to communication and application layers, such as:

- Transport protocols, for example HTTP, Simple Message Transport Protocol (SMTP), among others
- Message formats, such as XML and SOAP
- Service discovery, such as Universal Description, Discovery, and Integration (UDDI)
- Service description, such as WSDL

Without various standards, an SOA-enabled approach might not be easily and successfully implemented.

Note: Open Service Oriented Architecture (www.osoa.com) owns the SOA specs.

Standards That Enable SOA

	Current and Emerging Standards	Category
Management WS-Policy WS-Security	Specifications forming a collection of standards	Service Component Architecture (SCA)
		Assembly Model
		Service Data Objects (SDO)
	Orchestration: Business Process Execution Language (BPEL4WS)	Business processes
	WS-Reliable Messaging	Quality of service
	Universal Description, Discovery, and Integration (UDDI)	Discovery
	Web Services Description Language (WSDL)	Description
	SOAP	Message
	XML	Transport
	HTTP(S), IIOP, JMS, SMTP	

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The graphic illustrates the many existing World Wide Web Consortium (W3C) standards upon which an SOA-enabled approach can be implemented. The Service Component Architecture (SCA) and Service Data Objects (SDO) specifications are a work in progress guided by the OASIS organization that may lead to a new collection of standards with an SOA-enabled approach in mind.

The collection of standards listed work together and build on simple, stand-alone standards (such as XML and XPath) to enable an SOA approach (that is, using the Web service foundations of SOAP, WSDL, and UDDI). The idea of an SOA is elevated to a higher level through process orchestration and integration.

Note: Embracing a set of standards for an SOA-enabled approach enables independence of hardware, operating systems, and languages used for application implementation for the business process and service functionality.

All the standards above the transport category, such as Business Process Execution Language (BPEL) often known as “BPEL for Web Services” (BPEL4WS for the BPEL V1.1 standards) and more recently (circa 2007) known as BPEL Process Execution Language Version 2.0 (WS-BPEL), SCA and SDO are all XML-based implementations. As many may have heard before: “XML is the lingua franca of the Internet” and more so in the case of SOA-enabled approaches today.

Designing with an SOA Approach

Designing with an SOA approach involves:

- Aligning IT with the business requirements—design requires to be driven by the business process.
- Identifying services that perform functionality as part of a business process—building a service portfolio
 - Designing standard message structures in XML format using XML Schema Documents (XSD)
 - Specifying service interfaces in Web Services Definition Language (WSDL) format
 - Selecting choice of service implementation
 - Java Web Services and J2CA Adapters
 - BPEL
 - SCA Composite Applications
 - .NET, among others



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

When designing systems integration approaches with an SOA-style implementation, the focus is always to be aligned with the business requirements. The reason for this is the business defines and changes their processes as required to meet with changing forces in their industry domain. Therefore it is the job of IT to be agile enough to change as the business requires.

When designing a service it should be evaluated in terms of its role in the business and how it can be reused (if at all) subject to changes in the business requirements.

Note: Some services by nature may not be reusable and would have to undergo maintenance and changes to be aligned with the business.

Once the business process flow and requirements are understood, when it comes to designing each service, the key artifacts that require careful design are:

- Service message structures that must be exchanged between service consumer and clients. Design of message structures is a form of data modeling with respect to identifying what information is required by a service to perform its function.

- Service interface definitions that describe their operations and associated input and output messages. Defining a service interface, in Web Services Definition Language (WSDL), specifies if operations are synchronous or asynchronous for their intended usage.
- Service functionality can be designed based on choice of implementation language or type.

Creating Service Portfolios

Created with reusability in mind, a service portfolio:

- Is identified from a set of services required for implementing one or more business processes within a business domain
- Can be stored in an Enterprise Repository for design-time and SOA governance purposes
- Can be accessed from a Service Registry at design time and run time for location independence and governance
- Can be realized by:
 - Reusing functionality by wrapping as services, and using adapter technology
 - Creating new services with SOA-enabled technology, such as Web Services, SCA Composite Applications, BPEL, and a Service Bus



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

When embarking on an SOA approach to systems integration, systems integrators begin by examining business process requirements and the business domain to identify functionality to be used as a “service” unit or work to help complete a business process. The aim of building a system portfolio is to identify common functionality to be decoupled and exposed or used as a service within a business domain. While the business process drives the identification of services, the analysts should always keep the goal of reusability in mind within the business domain.

Note: Not all services or service-enabled applications should be reusable. Experience helps to find the right balance.

After a collection of services has been identified to serve the business process and can be applied to a business domain, it is recommended to store information about services, their interfaces, and message structure in a common location such as an Enterprise Repository. This facilitates and promotes sharing and reusability from design through to production. Coupled with a Service Registry, an Enterprise Repository can migrate service run-time information into development, test, and production operational environments. The power of coupling an Enterprise Repository and Service Registry enable an organization to implement strong SOA governance strategies for services throughout a service life cycle from design-time to run-time environment. A Service Registry enables applications to look up service using a service key for location independence.

SOA, Web Services and Java EE

- SOA by itself does not imply any particular technology.
- Web services provide an ideal technical foundation to build SOA-based systems.
- Java EE provides a sound platform to implement a web service-based SOA.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Web services typically represent self-contained functionality of a distributed application. This leads to a level of abstraction comparable to services in SOA.

Web service technologies can be used to implement loosely-coupled business services. For example, WSDL provides a language and platform independent approach to describe service interfaces, and a UDDI registry allows services to be dynamically published and located.

Web services allow existing information systems to be leveraged and reused. Web services allows EIS resources to be integrated into the SOA-based system to build composite applications.

Industry-wide efforts in web service interoperability increases the ease of integration between systems or business organizations. For example, the profiles defined by the Web Service Interoperability Organization (WS-I) provide clarification, constraints, and guidelines for using standard specifications to implement interoperable Web services.

Java EE 6 relies on protocols specified in WS-I Basic Profile 1.1. This allows web services developed in Java EE to not only take advantages of services provided by the application server, but also maintain their level of interoperability.

Business Process Execution Language (BPEL)

- Short for Web Services Business Process Execution Language (WS-BPEL), BPEL is an OASIS standard executable language for specifying interactions with web services.
- Processes in Business Process Execution Language export and import information by using web service interfaces exclusively.

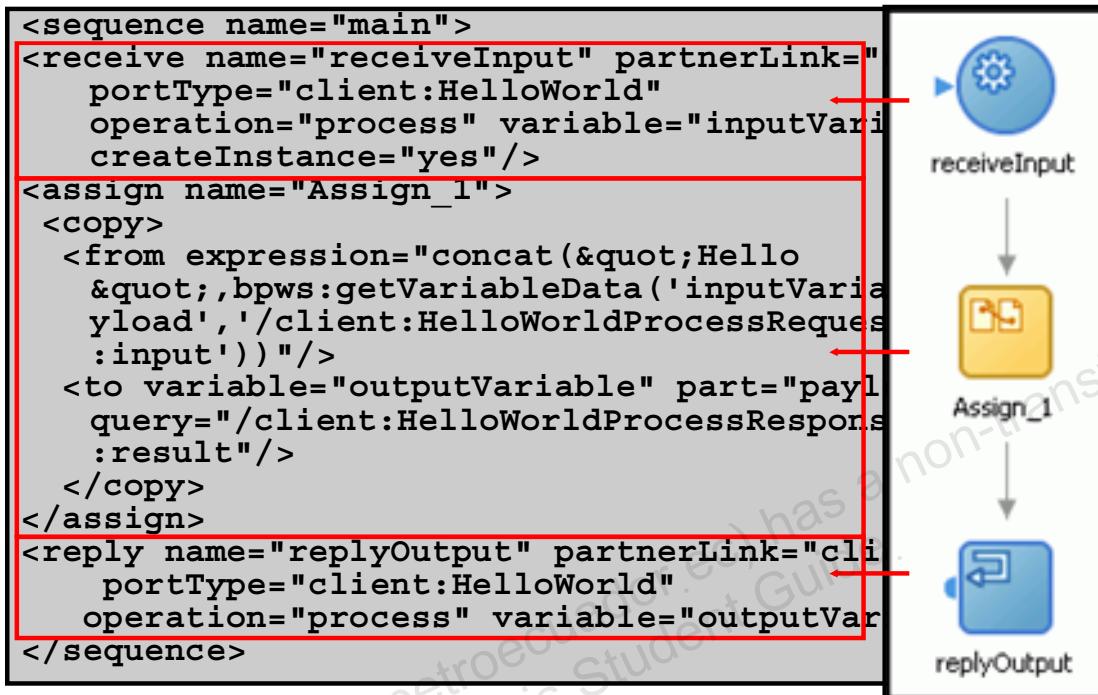


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

BPEL is an XML-based language for enabling task sharing across multiple enterprises using a combination of web services. BPEL is based on the XML schema, simple object access protocol (SOAP), and web services description language (WSDL). BPEL provides enterprises with an industry standard for business process orchestration and execution. Using BPEL, you design a business process that integrates a series of discrete services into an end-to-end process flow. This integration reduces process cost and complexity. The BPEL language enables you to define how to:

- Send XML messages to, and asynchronously receive XML messages from, remote services
- Manipulate XML data structures
- Manage events and exceptions
- Design parallel flows of process execution Undo portions of processes when exceptions occur

Introducing Business Process Execution Language (BPEL)



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The graphic illustrates a notation for specifying business process behavior and interaction with services, web services, or other services exposed through WSDL. The notation used is called Business Process Execution Language, or BPEL. BPEL is a markup language for composing a set of discrete services into an end-to-end process flow. A BPEL process uses service interfaces described in its WSDL for exposing its functionality, and invoking operations described in interfaces in the WSDL of other services.

A business process can be described in two ways:

- An executable process of the actual behavior of a participant in a business interaction
- An abstract process describing a business protocol

BPEL models the behavior of both executable and abstract processes. A business protocol uses process descriptions that specify the shared message structures exchanged between parties involved in the protocol, without exposing their internal behavior. BPEL:

- Is a language for formal specification of business processes and interaction protocols
- Extends the web services interaction model and enables support for business transactions

- Defines an interoperable integration model facilitating the expansion of automated process integration of internal and external services
- Is graphically modeled to be processed and executed by a BPEL process engine

BPEL Depends on WSDL

BPEL depends on the following XML-based specifications:

- WSDL 1.1
- XML Schema 1.0
- XPath 1.0
- WS-Addressing

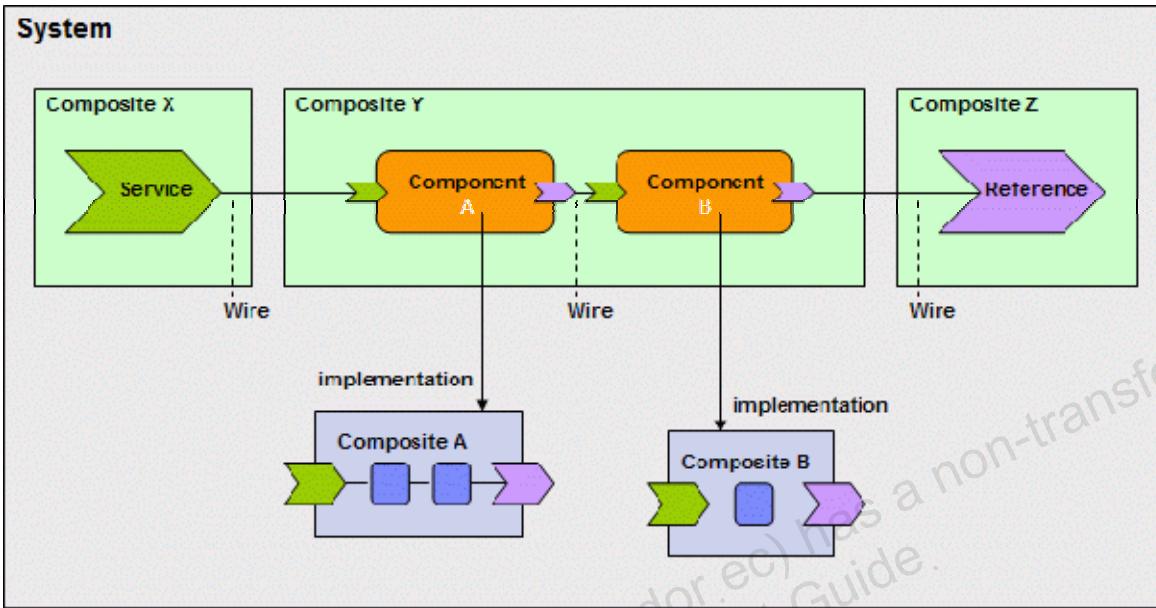
Among these, WSDL has the most influence on BPEL whose process model is layered on top of the service model defined by WSDL. The BPEL process model is based on the concept of peer-to-peer interaction between services (or partners) described in WSDL.

The BPEL process coordinates interactions between a process instance and its partners. Both the BPEL process and its partners are modeled as WSDL services. Therefore, the BPEL process definition:

- Interacts with one or more WSDL services
- Provides the description of the behavior and interactions of a process instance through WSDL. That is, the definition of a BPEL business process follows the WSDL model of separation between the message content and deployment information (messages and port type versus binding and address information)

In short, a BPEL process is a web service.

Service Component Architecture (SCA) and Service Data Objects (SDO)



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Service Component Architecture (SCA) and Service Data Objects (SDO) are new standards that simplify service-oriented architecture (SOA) programming. Specifically, they are technologies designed to:

Simplify the representation of service-oriented business logic: SCA gives developers and architects the ability to represent business logic as reusable components that can be easily integrated into any SCA-compliant application or solution. The resulting application is known as a composite application.

Simplify the representation of associated way to access data and can be used to modify business data regardless of how it is physically accessed. Developers and architects do not need to know the technical details of how to access a particular back-end data source in order to use SDO in their composite applications. Consequently, they can use static or dynamic programming styles and obtain connected as well as disconnected access.

SCA and SDOs are designed to be used together with the SOA programming model. In that model, business components are represented as SCA components and the data used between components are represented as SDOs.

SCA and SDO specifications are being jointly developed within the Open SOA Collaboration by Oracle and its partners.

Service Component Architecture (SCA) is a set of specifications which describe a model for building applications and systems using a Service-Oriented Architecture. SCA extends and complements prior approaches to implementing services, and SCA builds on open standards such as web services.

SCA encourages an SOA organization of business application code based on components that implement business logic, which offer their capabilities through service-oriented interfaces called services and which consume functions offered by other components through service-oriented interfaces, called references. SCA divides up the steps in building a service-oriented application into two major parts:

The implementation of components which provide services and consume other services

The assembly of sets of components to build business applications, through the wiring of references to services.

SCA emphasizes the decoupling of service implementation and of service assembly from the details of infrastructure capabilities and from the details of the access methods used to invoke services. SCA components operate at a business level and use a minimum of middleware APIs.

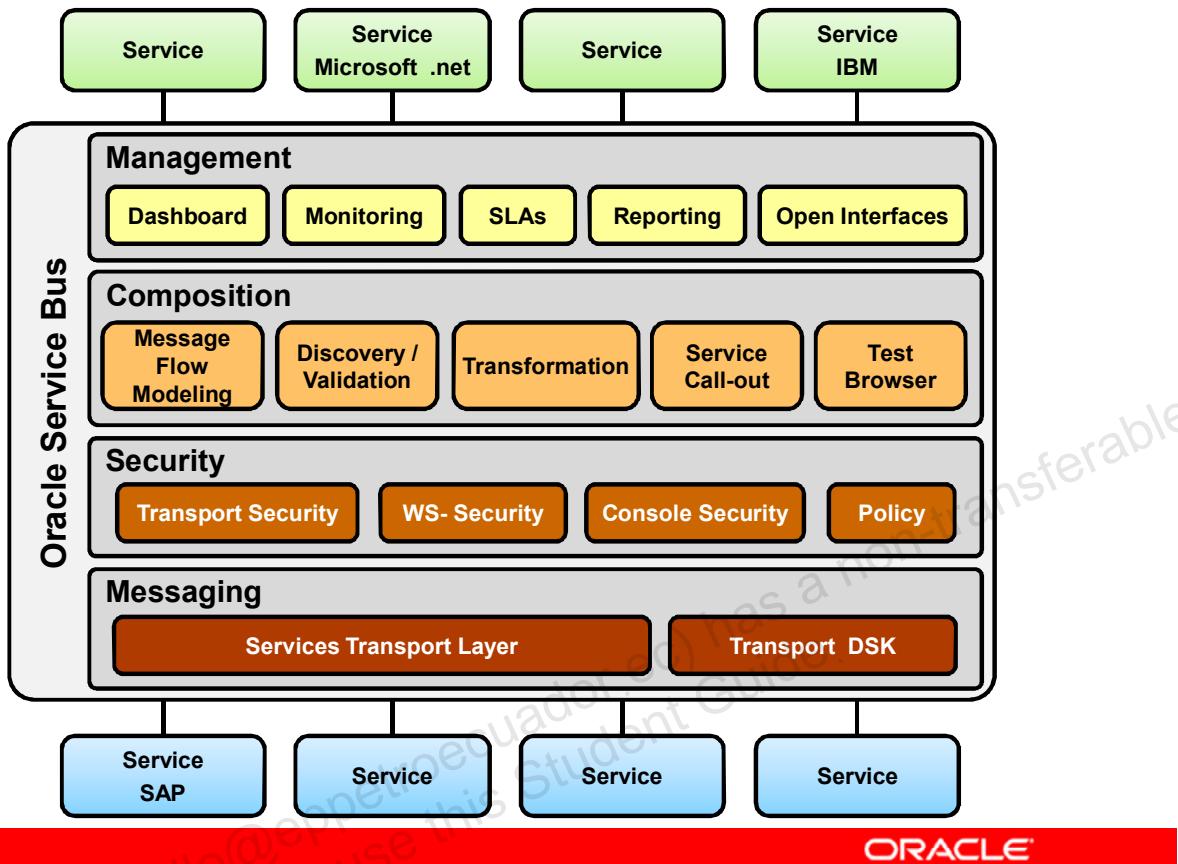
SCA supports service implementations written using any one of many programming languages, both including conventional object-oriented and procedural languages such as Java™, PHP, C++, COBOL; XML-centric languages such as BPEL and XSLT; also declarative languages such as SQL and XQuery. SCA also supports a range of programming styles, including asynchronous and message-oriented styles, in addition to the synchronous call-and-return style.

SCA supports bindings to a wide range of access mechanisms used to invoke services. These include Web services, Messaging systems and CORBA IIOP. Bindings are handled declaratively and are independent of the implementation code. Infrastructure capabilities, such as Security, Transactions and the use of Reliable Messaging are also handled declaratively and are separated from the implementation code. SCA defines the usage of infrastructure capabilities through the use of Policies and Profiles, which are designed to simplify the mechanism by which the capabilities are applied to business systems.

SCA also promotes the use of Service Data Objects to represent the business data that forms the parameters and return values of services, providing uniform access to business data to complement the uniform access to business services offered by SCA itself.

The SCA specification is divided into a number of documents, each dealing with a different aspect of SCA. The Assembly Model deals with the aggregation of components and the linking of components through wiring using composites. The Assembly Model is independent of implementation language. The Client and Implementation specifications deal with the implementation of services and of service clients—each implementation language has its own Client and Implementation specification, which describes the SCA model for that language.

ESB Features and Functions



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

An ESB provides an efficient way to build and deploy enterprise SOA. ESB is a concept that has gained the attention of architects and developers, as it provides an effective approach to solving common SOA hurdles associated with service orchestration, application data synchronization, and business activity monitoring. In its most basic form, an ESB offers the following key features:

- **Web services:** support for SOAP, WSDL and UDDI, as well as emerging standards such as WS-Reliable Messaging and WS-Security
- **Messaging:** asynchronous store-and-forward delivery with multiple qualities of service
- **Data transformation:** XML to XML
- **Content-based routing:** publish and subscribe routing across multiple types of sources and destinations
- **Platform-neutral:** connect to any technology in the enterprise, for example, Java, .Net, mainframes, and databases

Adaptive Service Messaging Layer

Existing middleware, applications, and data sources become first-class citizens of SOA initiatives, protecting existing IT investments and enabling IT to connect, mediate, and manage services using heterogeneous end points, formats, and protocols.

Oracle Service Bus works with diverse Web Services transports including the following:

- HTTP/SOAP
- WS-I, WS-Security, WS-Policy, WS-Addressing
- SOAP v1.1 and v1.2
- EJB/RMI on WebSphere

Optimized Pluggable Security Layer

A comprehensive set of components for built-in security give customers significant flexibility and choice. Users can also plug in home-grown or third-party security components. Built-in capabilities allow flexibility in implementation by enabling security at the following levels:

- Transport Security - SSL/Basic Auth
- Message Security - support for WS-Policy/WS-Security, SAML, UserID/Password, X509, Signing & Encryption, and Custom security credentials
- Console Security - support for user Web Single-Sign-On and role based access
- Policy - leverages WS-Security and WS-Policy

Rich Service Composition Layer

Oracle Service Bus provides a rich environment for service configuration, modeling, discovery and message validation. It also supports message transformations, service callouts to external service providers, and a test browser.

Oracle Service Bus configuration-driven composition environment, with Oracle Service Bus plug-ins for Eclipse and the Oracle Service Bus Console, is designed with a no coding approach, allows message flow modeling using versatile graphical modeling tools. The modeling capabilities allow for configuration of dynamic content-based routing, message validation and exception handling.

The composition environment includes a service discovery and validation capabilities for automatic import and synchronization of services with UDDI registries. This functionality allows for validation to ensure service integrity and reconciliation of conflicts before deployment.

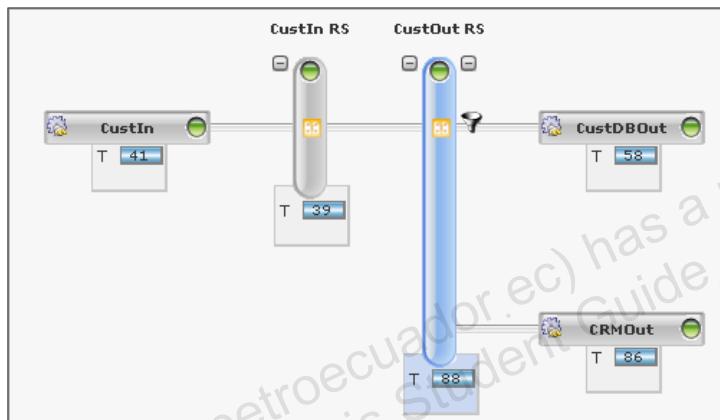
Embedded Service Management Layer

Oracle Service Bus offers embedded service management capabilities that provide optimized governance of all messaging. Its preemptive support ensures that mission-critical business processes continue to serve customer needs, even as business demands, requirements, and workloads change. Oracle Service Bus Dashboard provides an unified data service interface for all application development and maintenance, service monitoring and management and improved operations support. The dashboard allows for monitoring of fault and performance metrics, viewing of summaries for aggregated ESB. It allows for dynamically defining and managing routing relationships, transformations, and policies.

Oracle Service Bus has built-in optimizations for performance and monitoring, guarantee quality of service through alert monitoring on single node or entire server, and configurable Service Level Agreements (SLAs) for messages. It also supports viewing and managing SLA alerts on operation metrics and message pipelines.

Canonical Model

- Defines a common format to describe business entities within the enterprise
- Reduces interface maintenance and encapsulates business logic in one central place
- Each application produces messages in this common format



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The canonical model supports integration of applications that have different data formats. Each application produces and consumes messages in this format, thus reducing or eliminating the burden of transforming messages between applications. ESBs can also do this to messages as they come through the bus.

The canonical model provides an additional level of indirection between individual data formats. If a new application is added to the solution, only transformations between the canonical model has to be created, regardless of the number of applications that participate in the solution.

Use the 80/20 rule to define the canonical model. Start with all unique identifiers and a superset of the most used and relevant data. Changes are made to the model when 80% of the consumers need new attributes. For schema versioning, the Format Indicator pattern is useful. Use declarative namespaces to manage data domains in order to have a generic enterprise wide data definition strategy. Use generic XML types instead of database specific types for schemas.

A Canonical Model has both a Logical and Physical Representation, just as a Data Model does. The logical model represents your enterprise entities in application neutral terms. The names are expressed in Business terms. The Physical model is in the form of an XSD (by entity or by Enterprise context), defining the elements and lightweight properties, datatypes, lengths, etc.

SOA Best Practices

Guidelines for identifying and designing services:

- Define self-contained service interfaces to reflect the business functionality first: Make them independent of particular applications and implementation technologies
- Map service interfaces to modularized technology-oriented artifacts.
- Provide consistent implementation guidelines and technologies to address the quality requirements.
- Over time, categorize these services into coherent business service portfolios to improve reusability.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

SOA essentially presents a paradigm shift in designing information systems. It proposes a composite approach to integrate existing information systems and services to deliver values to customers and business partners. Because of this, adopting the SOA approach should be an evolutionary process, and designing a system based on SOA requires considerations at different levels, including design, implementation, and organization-level governance.

When you identify business functionality as services and design a blueprint to guide their implementation, consider the following guidelines:

- Define self-contained service interfaces to reflect the business functionality first. The initial definition of these service interfaces should not depend on particular applications and implementation technologies. Typically, a service interface should be self-contained to allow the invocation of a complete use case.
- Map these service interfaces to modularized technology-oriented artifacts, such as WSDL and XML Schema documents.

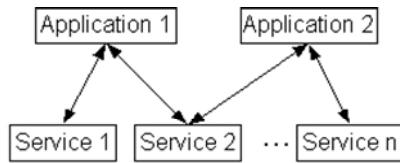
Provide consistent implementation guidelines and technologies to address the quality-of-service requirements. An SOA-based solution typically implies a composition of application components and fine-grained services, you should apply appropriate technologies at different granularity levels. For example, if the implementation of a service requires distributed transactions, you should consider using a component model, such as Enterprise JavaBeans as a façade to manage the transactional execution. On the other hand, if the service is a composition of other service components, you should consider using an orchestration mechanism, such as BPEL to manage their invocation.

Categorize these services into coherent business service portfolios to improve reusability. Over time, these portfolios allows you to create architectural templates that can be reused to solve similar problems.

Moving to SOA

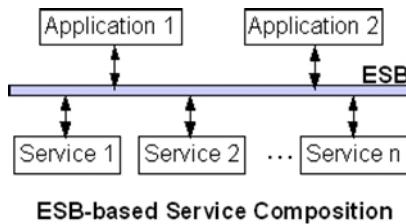
Integration Level Guidelines

- From



Point-to-Point Service Composition

- To



ESB-based Service Composition

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

In large scale SOA-based systems, new services and applications are built as a composition of existing services. To reduce the complexity of the point-to-point communication path among services, an enterprise service bus (ESB) approach is used to provide capabilities, such as intelligent routing. The point-to-point and ESB-based approaches are illustrated in the slide.

SOA Governance

- To maximize the value of SOA-based systems, services must have a high level of reusability and agility.
- To be adaptive to the constantly changing business operations and technology landscape, building SOA-based systems requires strong and comprehensive governance that goes beyond information technologies.
- SOA Governance refers to the process of enforcing organizational policies and standards throughout the development and deployment of an SOA system.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

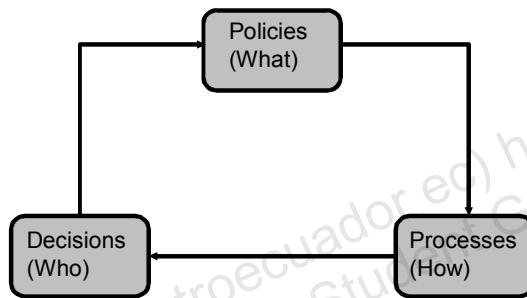
The goal of SOA governance is to ensure that the overall goal of SOA is achieved through consistent service planning, design, development, integration, change, deployment, and operation. Its scope ranges from business-level strategy planning to more specific decisions on architecture, products, and the evolution of the system.

One of the main technical components required for effective SOA governance is a registry and a repository. The registry allows the service interface descriptions to be categorized and dynamically published or discovered. On the other hand, the repository provides capabilities to store and manage additional artifacts for SOA governance.

Define SOA Governance

SOA governance can be defined as:

- A set of solutions, policies, and practices that enable organizations to implement and manage an enterprise SOA
- An application of IT governance specifically focused on the life cycle of services and composite applications in an organization's service-oriented architecture



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Governance is about ensuring that business is conducted properly. It is less about control and strict adherence to rules, and more about guidance and the effective and equitable use of resources to ensure sustainability of an organization's strategic objectives. Governance is a process that is influenced through organizational behavior and the establishment of structured processes. Technology is there to help automate the governance process as much as possible. In other words, it is a framework for managing SOA assets in compliance with a company's standards, policies, and business strategies specifically focused on the life cycle of services, policies, practices, metadata, and composite applications.

Identifying the Need of SOA Governance

- Business value
- Alignment
- Business agility
- Risk reduction
- Cost savings



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

SOA governance offers the following benefits:

- **Business value:** Ensures that project investments yield business value
- **Alignment:** Keeps SOA aligned with the business and architecture and in compliance with business and IT policies
- **Business agility:** Gains visibility into your SOA for more rapid decision making
- **Risk reduction:** Controls dependencies, manages the impact of change, enforces policies
- **Cost savings:** Promotes consolidation, standardization, and reuse

In the context of web services-based SOA, web service registries are typically based on UDDI, which focuses on registration of WSDL service descriptions. While this is an essential function, large SOA projects can rely on additional metadata and artifacts than only those of WSDL. Examples of these artifacts include XML schemas and WS-BPEL descriptions. Such artifacts also need to be centrally accessible to promote the benefits of reuse and control, and standard ways of storing and retrieving them, capabilities that are not addressed by UDDI.

Another standard in this area that accommodates these needs is the ebXML Registry. It not only supports web service registry functions, but also provides a tightly integrated repository and functions for the organization, storage, and control of any kind of service metadata or artifact. Functions include those for federated Web service asset management across multiple repositories.

The combined registry and repository capabilities make for a more flexible and complete service metadata and artifact management solution for large-scale SOA deployments. These capabilities also provide core infrastructure support not only for service discovery, but also for life-cycle management and SOA governance.

What Do You Think

- What is the relationship between SOA, EA and BPM?
- Is one dependent on another?
- Is EA basically SOA? BPM?



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Additional Resources

The following references provide additional information on the topics described in this lesson:

- Hohpe, Gregor, and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston: Addison-Wesley, 2003.
- Erl, Thomas. *SOA Design Patterns (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*, Prentice Hall, 2009.
- Josuttis, Nicolai M. SOA in Practice: *The Art of Distributed System Design (Theory in Practice)*, O'Reilly Media, 2007.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Raj, Gopalan Suresh, Binod PG, Keith Babo, and Rick Palkovic. *Implementing Service-Oriented Architectures (SOA) with the Java EE 6 SDK*.

[<http://java.sun.com/developer/technicalArticles/WebServices/soa3>], accessed February 27, 2007.

Hohpe, Gregor, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional. 2003.

Organization for the Advancement of Structured Information Standards (OASIS). [<http://www.oasis-open.org>], accessed February 27, 2007.

JSR 208, *Java Business Integration*. [<http://www.jcp.org/en/jsr/detail?id=208>], accessed February 27, 2007.

Coqueret, Regis and Marc Fiammante. *Choosing among JCA, JMS, and Web services for EAI: Positioning alternative interface technologies for enterprise integration*. [<http://www-128.ibm.com/developerworks/Webservices/library/ws-jcajms.html>], accessed February 27, 2007.

Quiz

An integration Tier is less distinguishable, but still as important for the following reasons: (Choose all that apply.)

- a. It facilitates communication with the database
- b. It reduces dependency between application components and EISR
- c. It increases the reusability of the integration components
- d. It is provided in Java EE6 through the Java Integration API



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: b, c

Quiz

The resource tier encloses all of the external EIS resources. Which of the following are NOT typically part of the Resource Tier?

- a. Relational Databases
- b. Legacy applications
- c. Data feeds
- d. Enterprise JavaBeans



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: d

Quiz

Given a situation in which a large, complex data store is in place with limited (and questionable) documentation, what process will help with the integration into the new architecture?

- a. Use data Mining
- b. Create UML diagrams of existing systems
- c. Perform system testing
- d. Observe and record user interactions



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: a

Quiz

Given an architectural problem that requires a loosely-coupled solution, which of the following are good choices for integrating EIS resources? (Choose all that apply.)

- a. Java Messaging Service (JMS)
- b. Java Connector Architecture (JCA)
- c. Java Web Services
- d. Enterprise JavaBeans (EJB)



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: a, c

Summary

In this lesson, you should have learned how to:

- Describe the challenges in Enterprise Information System (EIS) integration
- Describe the roles of the integration tier
- Describe the EIS resource tier
- Review Java integration technologies and best practices
- Apply integration-tier patterns
- Describe how Service-Oriented Architecture (SOA) facilitates system integration
- Describe SOA best practices



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 11 Overview: Refining the Integration Tier Architecture

This practice covers the following topics:

- Interacting with external vendors and service providers.
- Presenting the system as a set of services.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

In this practice you will add architectural elements to support interaction with remote external vendors services.

12

Evaluating the Software Architecture

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

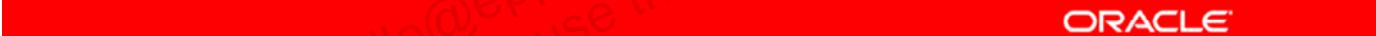
Objectives

After completing this lesson, you should be able to:

- Describe architecture evaluation guidelines
- Evaluate Java EE technologies and their applicability
- Create system prototypes
- List and define application server selection criteria

Discussion Questions

- How do you evaluate an architecture?
- What Java EE technologies are available for addressing common programming needs of an application?
- How do you choose an appropriate architecture from multiple candidates?
- How can you use the prototypes to assess the expected performance or to assess other QoS requirements of the final system?



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

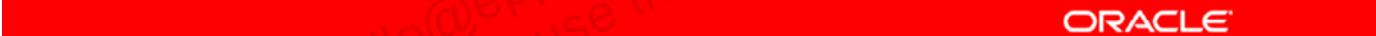
- Evaluating software architectures
- Evaluating Java EE technologies
- Creating system prototypes
- Selecting servers and frameworks

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Evaluating Software Architectures

- Architectures capture the essential decisions that will be applied throughout the product design and implementation.
- It is essential to evaluate the architecture, potentially multiple candidate architectures, to assure that the architecture indeed provides a sound blueprint.
- Effective evaluation can improve the architecture, hence the product being developed.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Characteristics of a Good Architecture

- Conceptual integrity:
 - An architecture should provide a consistent guideline to the system development.
 - Conceptual integrity helps all the stakeholders to easily understand and envision the system being developed.
- Correctness and completeness:
 - An architecture should include all the high-level decisions.
 - The architecture should clearly state trade-off decisions.
- Buildability:
 - The architecture should present a blueprint that allows the system to be completed.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The goal of the architecture is to ensure the product under development can meet the functional and quality requirements. Typically, a sound architecture that facilitates this goal has the following characteristics:

Conceptual integrity

An architecture should provide a consistent guideline to the system development. Conceptual integrity helps all the stakeholders to understand and envision the system being developed. This allows the detailed design and implementation to be carried out in a predictable manner. Common practices to achieve conceptual integrity include applying proven patterns and maintaining a reference architecture as the IT asset of an organization.

Correctness and completeness

An architecture should include all the high-level decisions made to address the systemic quality requirements. If some quality requirements cannot be met, the architecture should state trade-off decisions that need to be made. In practice, the most effective way to ensure the correctness and completeness is through formal architectural evaluation.

Buildability

The architecture should present a blueprint that allows the system to be completed by the available team in a timely manner. It should also be open to necessary changes as development progresses. To achieve buildability, the architecture should incorporate the time, cost and technology constraints.

Architecture Evaluation Guidelines

Architecture evaluation can take many different forms.

Typically, however, an effective architecture evaluation should incorporate the following best practices:

- Treat architecture evaluation as an activity of the development process.
- Conduct architecture evaluation at the most appropriate time.
- Select appropriate evaluation techniques.
- Manage the scope and expectation of the evaluation activity.
- Produce and maintain evaluation deliverables.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Evaluating software architectures
- Evaluating Java EE technologies
- Creating system prototypes
- Selecting servers and frameworks



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Java EE Technologies and Architectural Objectives

Architectural Objectives	JPA Entity	Stateful Session Bean	Stateless Session Bean	Message-Driven Bean	Servlets	JSF/JSP	Managed Beans	Custom Tags
Long-term Application State	Yes	Maybe	Maybe	Maybe	Unlikely	No	Maybe	Maybe
Client Session State	No	Yes	No	No	Maybe	No	Maybe	Maybe
Business Process and Workflow Control	Maybe	Yes	Yes	Yes	Maybe	No	Maybe	Maybe
Presentation Process and Workflow Control	No	Maybe	No	No	Yes	Maybe	Maybe	Yes
Presentation Layout	Unlikely	Unlikely	No	No	Maybe	Yes	Maybe	Yes
Asynchronous communication	Maybe	Yes	Yes	Yes	Yes	No	Maybe	Maybe

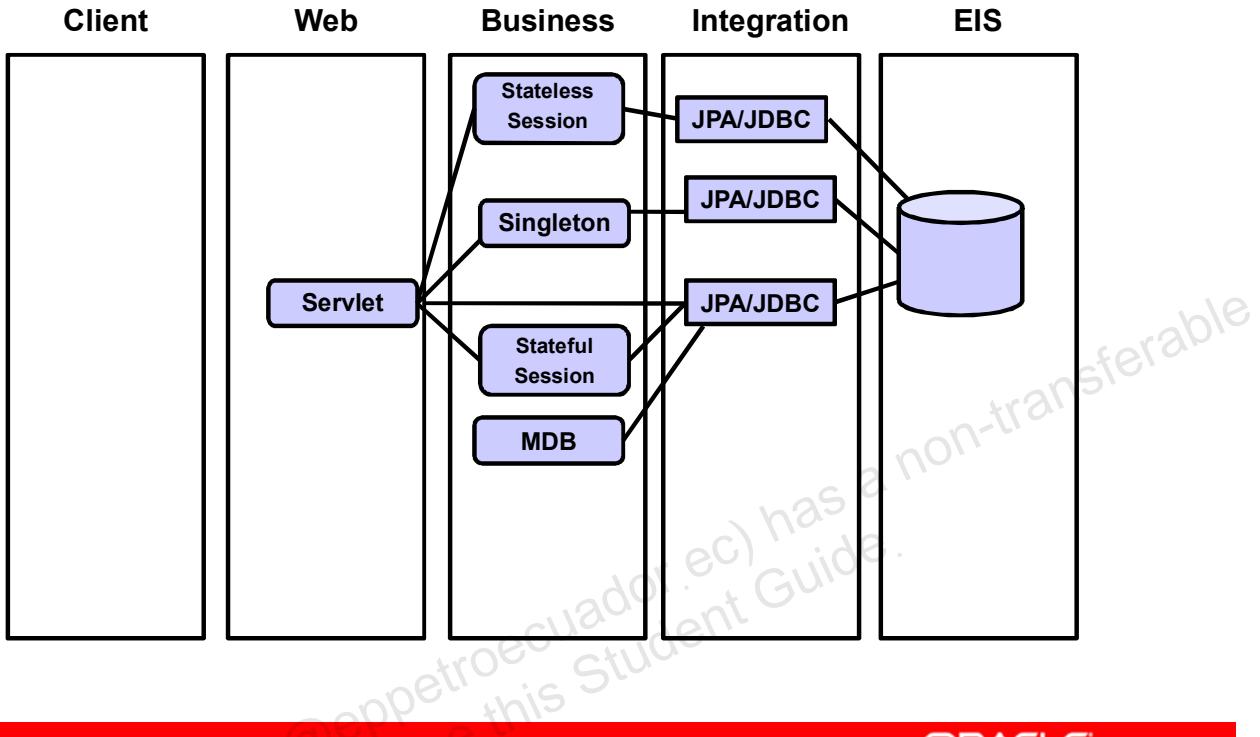


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The Java EE technology provides a variety of component models that can be used to solve different problems. Understanding the applicability of these components is essential to creating a solid architecture.

Custom tags are used by JSP technology and can do almost anything in that context. For example, custom tags can contain SQL code to connect to a database. This might be appropriate if the system is small and is an unlikely candidate for future scaling. This lesson identifies custom tags as particularly relevant in two specific applications, but does not otherwise mention their general applicability.

Designing for Long-Term Application State



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

One of the main functions of an application is to modify and maintain data. That data is stored in objects, and the state of the application is reflected in the state of those same objects. The business model has several options for managing the persistence of the application state.

The slide shows the basic layout of the main techniques that are used for managing application state. The MDB typically uses a JPA to handle the actual updates, as is typically the case for a session bean. Because the graphics in this module are designed to show the differences between components, they do not use UML notation.

JPA Entities

The primary design goal of entities is to provide an object mapping of persistent storage. Because entity classes are OO views into a persistent store, they represent the following items:

- Data from the persistent store
- Operations on the data from the persistent store

As a result, you should ask whether the appropriate operations are being defined in the entity class.

Operations on entity classes focus on two functions:

- Maintaining data integrity
- Moving data between the application and the persistent store in meaningful chunks

However, the data must be intrinsic to the abstraction that the entity class defines.

Consequently, entity classes do not usually contain operations, except for those that describe constraints on the data themselves. For example, if a field in a database cannot meaningfully be negative, then it would be reasonable to encode that rule in the entity class itself, or use bean validation.

Session Beans

You might consider using either stateful or stateless session beans to interface with the long-term persistence mechanism of the application. As a general rule, use JPA to decouple the beans from the persistent store. This avoids an inappropriately close dependency of the bean on the data store. In addition, stateful session beans can implement the SessionSynchronization interface if the design requires transactional control of the session bean. However, there is a serious drawback to using stateful session beans to represent application data. A stateful session bean is uniquely owned by its client for the duration of its use. This means there could be potentially thousands of copies of the same data in the application at the same time, especially if this data is meant to be shared.

Message-Driven Beans

Messaging systems are sometimes used as an interfacing layer to legacy systems. For example, some systems use IBM MQ Series to access CICS or IMS. Asynchronous data store access has the following advantages:

- Provides scalability because the data store requests do not block the requestor, which frees server resources
- Supports batch requests
- Allows for messages that are transactional

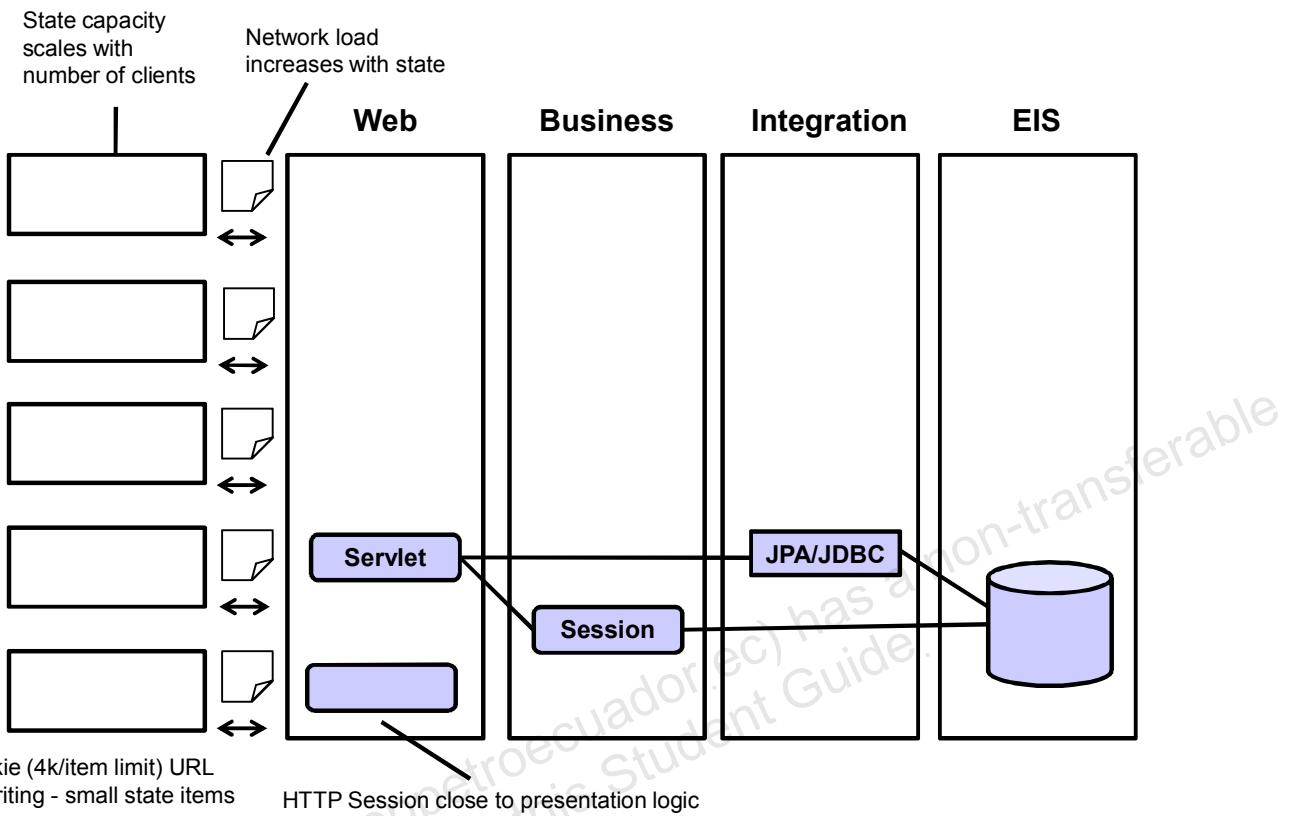
Servlets and JSF/JSP Technology

Servlets and JSF/JSP technology components are not well suited for managing application state, much less for managing the storage and retrieval of data. Therefore, as a general rule, do not use these technologies for that purpose. Data access functions are part of a business model, and helper beans are better suited to providing support to servlets and JSP technology components.

Note: Compared to JSF/JSP technology components, servlets are better suited to implementing complex choices.

There is a circumstance in which it might be appropriate to handle application state with the use of web-tier components. A JSF/JSP technology component would not be suitable because it does not handle complex logic well, but a servlet, helper bean, or custom-tag would be appropriate. If you have an application that has no other need of an EJB technology container, then you might reasonably handle persistence directly from the web-tier components. If you do not use an EJB technology container, you imply that you have no need of the transaction and concurrency management features of the EJB technology container and that you are confident that scalability requirements are not an issue. In this situation, it is still wise to create a JPA entity to separate the persistence logic from the other aspects of the application.

Managing Client Session State



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

As a general rule, do not persist client session state. Client session state exists for the duration of the client's connection and is abandoned when the connection is closed. Consequently, the selection of components for managing client conversation information typically does not need to support data persistence.

Also, the client's presentation content does not always require client state. Instead, you might be using only portions of the client state data or taking different views on that data. Consequently, although they are capable of managing client state, presentation-oriented components, such as JSP technology components and custom tags, are not useful for this application. The slide shows the options that you can use to handle client conversational state. The Client tier can store state with cookies or URL rewriting. URL rewriting can store only enough state to tie a browser to a particular client session, while cookies can hold up to 4 Kbytes each. State stored in cookies must pass from the client to the presentation tier before it can be used. This places a limit on the number of concurrent clients because of the network bandwidth used.

You can hold state on the presentation tier with the `HttpSession` object, which is a collection that can hold any non-primitive type. The Web container ties the `HttpSession` object to the client either with the use of cookies or URL rewriting, so no programming effort is required to do this function.

You can also hold state with stateful session beans or entity classes. Entity classes allow state to be persisted to the database.

JPA Entities

You might reasonably use entities to hold client session state if you intend to have long-lived state. For example, if you want to provide a shopping cart that remembers all of the items that the customer has considered buying, and then display those items when the customer returns to your site three weeks later, then an entity class might be appropriate. In this case, the real issue is whether you feel that this data belongs in a database.

Stateful Session Beans

Stateful session beans are designed to handle client session state. They do this efficiently in most cases. This efficiency results from the container's ability to swap out a bean if memory becomes short. Remember that the commonly-heard mantra, "Stateful session beans do not scale," tends to be misguided. If your application needs to maintain conversational state, then this state must be stored somewhere and must be communicated from that storage location to the appropriate part of the system when needed.

Instead of using a stateful session bean, consider some of the following alternatives:

- Store all state on the client, perhaps as cookies.

This approach scales infinitely from a memory point of view because each new client brings new storage for its own state. However, from a network bandwidth perspective, the story is not so good. Each time that the client makes a request, the client has to use additional bandwidth to send any state that the server might require. Increased bandwidth could be particularly problematic if the clients are running over slow connections. In addition, there is a cost in processing time to convert client data into application usable state objects.

- Store all state in the web tier.

This approach is a reasonable alternative if the web tier is the main user of the state. If, however, the business tier is the main user of the state, then you can consider using sessions bean local to (collocated in) the web tier.

- Store all state in the database.

This approach is likely to generate a large additional load on the database. Moreover, this approach creates additional network traffic between the database and whatever tier is using the state information. If the state must be long-term persistent for other reasons, however, this approach becomes a reasonable option.

Stateless Session Beans

Unlike stateful session beans, stateless session beans do not maintain client conversational state. This makes them inappropriate for solving state-related issues.

Message-Driven Beans

Message-driven beans are closely related to stateless session beans. Because of this, it is not practical to use them to handle client session state.

Servlets

As previously mentioned, servlets can function to track and manage client session state. This is valid only in two circumstances:

- Presentation data is client-specific

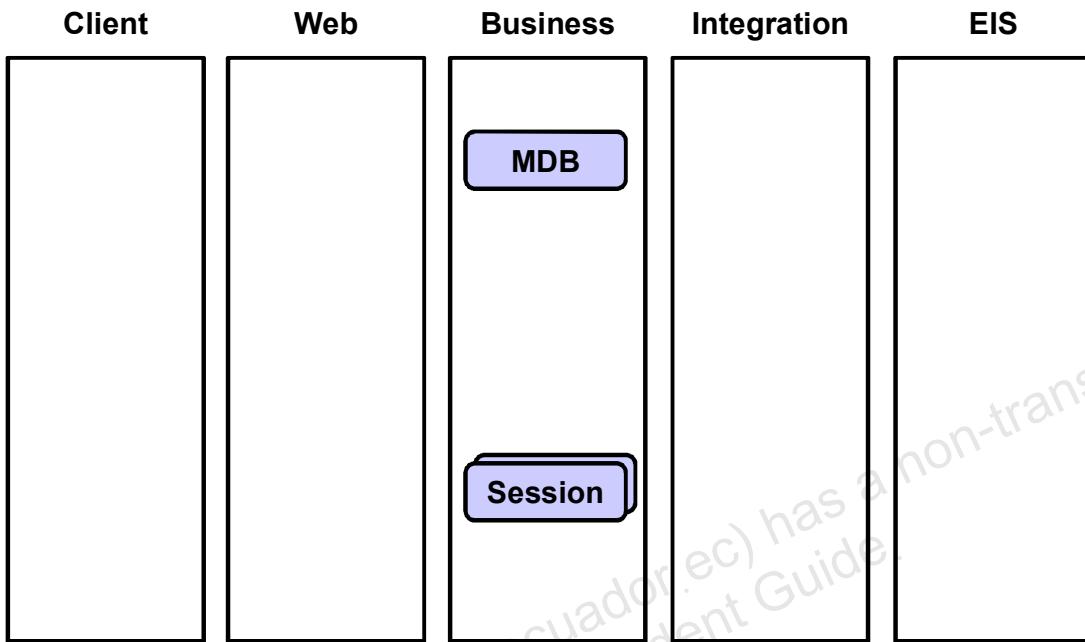
If the data is client-specific, there is a degree of control on when the data needs to be passed to and from the web tier.

- Business logic is completely stateless

Stateless business logic has no way to maintain state. It then falls to the web tier to manage stateless business logic.

Finally, if the web server or container is spending more of its time managing state than servicing clients (based on empirical data), then you should reconsider the use of servlets to track and manage client session state. In this case, it is usually preferred to store only enough data for session tracking and to store the larger portion of session data in a stateful session bean.

Enabling Business Process and Workflow Control



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Business process and workflow make up the functional aspects of the business model. They capture the use-case activities and map them onto the significant data-oriented tasks. You assign these functional aspects to components of the business model. Presentation technologies (such as JSF/JSP technology components) and persistence technologies (such as JPA entities) are not well suited to defining and managing business processes directly. The slide illustrates the main approach for handling business process, which is usually managed in the business tier.

Session Beans

Session beans, whether stateful or stateless, are designed primarily for modeling business processes and workflow. Session beans are a good way to provide application-level behavior to the web tier and work better than having the entities provide all of the behavior, or having the presentation tier model these processes. The presentation tier should be focused on presentation.

Message-Driven Beans

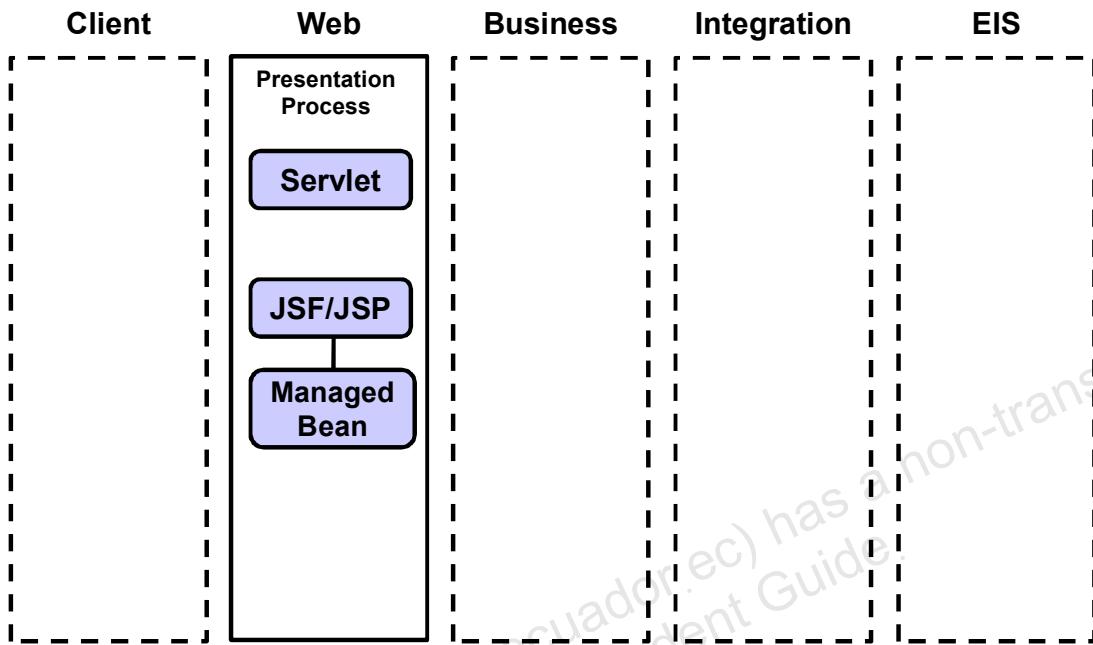
You sometimes use messaging to trigger changes in a state machine and thereby drive a process. If your system uses messages in this way, it is appropriate to use message-driven beans to implement this behavior. However, in the more general case, especially with new developments, it is more common to use some form of synchronous communication with a session bean to manage business processes.

Servlets

You can use servlets to perform business process control. However, it would be more usual to achieve this capability with the use of session beans. You might use servlets in this role if the following conditions exist:

- The system is small enough.
- The system is not expected to scale.
- There is no need for a middle tier.

Enabling Presentation Process and Workflow Control



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Presentation processing and workflow handle client request processing and routing and transform data sent from the business model into a format suitable for rendering at the client. This part of the application should not perform any processing directly related to the business model or the persistence of data. As a result, EJB architecture components are a poorly-suited technology for these tasks. The figure above shows the main approach for handling presentation process, which is usually managed in the presentation tier.

JPA entities

Again, entities are intended as an OO view into a persistent store. Therefore, entities are not a likely contender for handling the following operations:

- Presentation processes
- Presentation layout
- Workflow issues

Session Beans

You can use session beans to handle presentation logic, but presentation logic is usually best handled by the presentation-tier components as a separate issue from business processes.

However, if your system has simple and essentially static presentation requirements, and does not have a web interface, then it might be appropriate to use a session bean to handle presentation logic. If you take this route, use one session bean to handle presentation logic and have that bean communicate with another session bean that is charged with handling the business processes. This approach still provides the flexibility that comes from separating the unrelated concerns of presentation and process.

Presentation process and workflow depend on the client conversational state. Because stateless session beans do not maintain client conversational state, they are inappropriate for handling presentation process and workflow.

Servlets

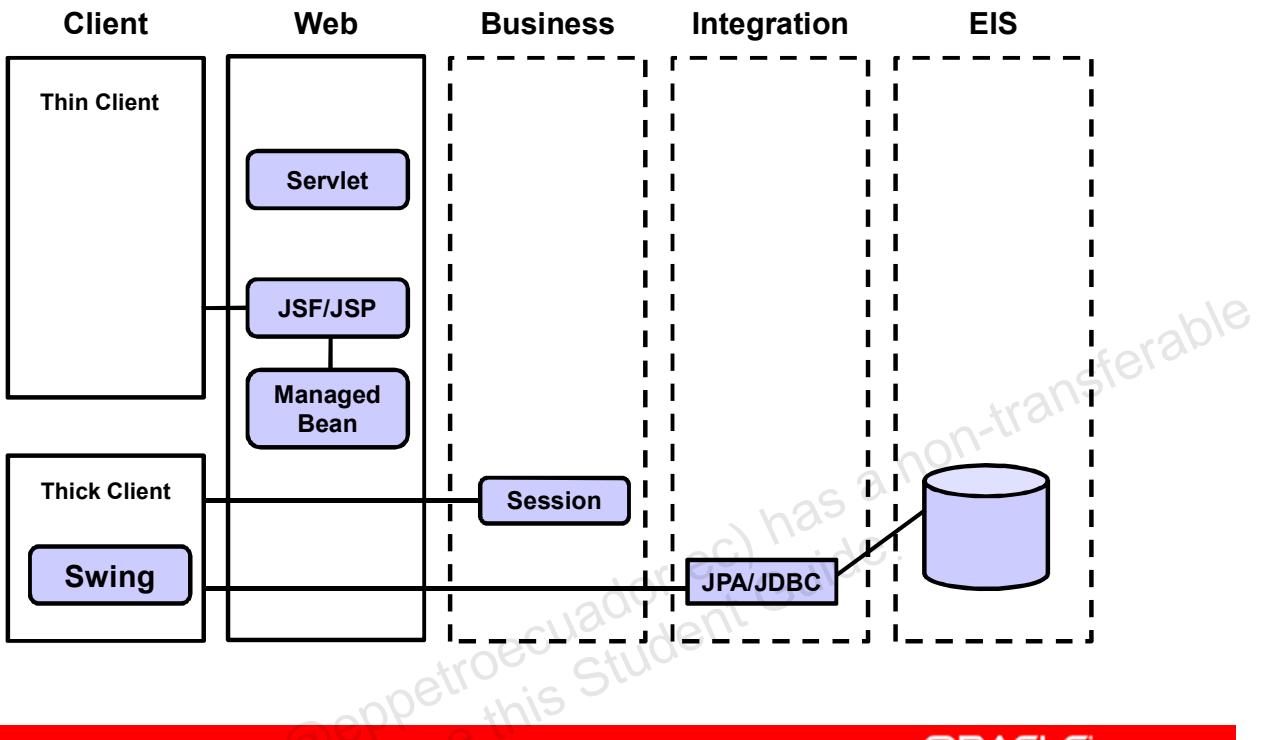
Servlets provide a way to perform complex logic programming immediately behind a web page. This means that servlets are primarily suited to presentation process and workflow control.

JSF/JSP Technology

The use of JSF/JSP technology to manage presentation process and workflow control is not uncommon. Many application developers use JSP technology components to handle large portions of the logic that is needed to process client requests and to trigger behavior in a business model. In fact, this is exactly the Model 1 web-tier architecture. However, the Model 1 web-tier architecture is considered to be a model made out of necessity, time-to-market requirements, or inexperience.

Note: For information on Model 1 and Model 2 architectures, see “Understanding JavaServer Pages Model 2 Architecture” at <http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmv.c.html>. Avoid the use of JSP technology to define Java technology code that more correctly belongs in a class. Instead, define the code in a helper bean, a custom tag class, or a servlet. In the Model 2 web-tier architecture, an application reflects the more mature approach of code separation and uses JSP technology for layout only. Model 2 architecture structure is the preferred model.

Managing Presentation Layout



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Presentation layout focuses on the client, the particular information that the client wants to view, and how clients have chosen to have that information formatted and presented to them. Presentation layout is often a highly dynamic and customized layer of the application, and it rarely has any aspect related to the business model.

Business model components, such as EJB technology components, are highly inappropriate for managing presentation layout. The presentation tasks are better suited to presentation-tier components, which are represented by the Web component technologies or by the components that support standalone application clients.

However, if you have a thick client for an application that has no web-based clients at all, then it might be reasonable to use session beans to provide an element of layout control. In most cases, these session beans should not be the same as the session bean that provides the main business processes. Moreover, these session beans should have substantial help from the client in performing the real layout. However, a form of prelayout might be of value. For example, consider a prelayout that gets data into a form that is both efficient for network transfer and more closely related to the expected visual layout.

An entity class might take some part in the layout, particularly in the thick client or no Web-tier scenario and if the layout is somehow determined dynamically according to persistent information in the database. This situation might arise in portal-like systems. It shows these options.

Message-Driven Beans

Message-driven beans are essentially undedicated. That is, a message-driven bean does not receive messages from a specific client, but it does receive any message on a particular topic or from a particular queue. This makes message-driven beans inappropriate for handling presentation elements of a system.

JSF/JSP Technology

JSF/JSP technology is primarily intended to provide a mechanism for nonprogrammers to build custom utilities into potentially complex layouts. These utilities are usually embedded in custom tags or managed beans. Without complex processing, JSF/JSP technology is not suited to any presentation layout work other than presentation management.

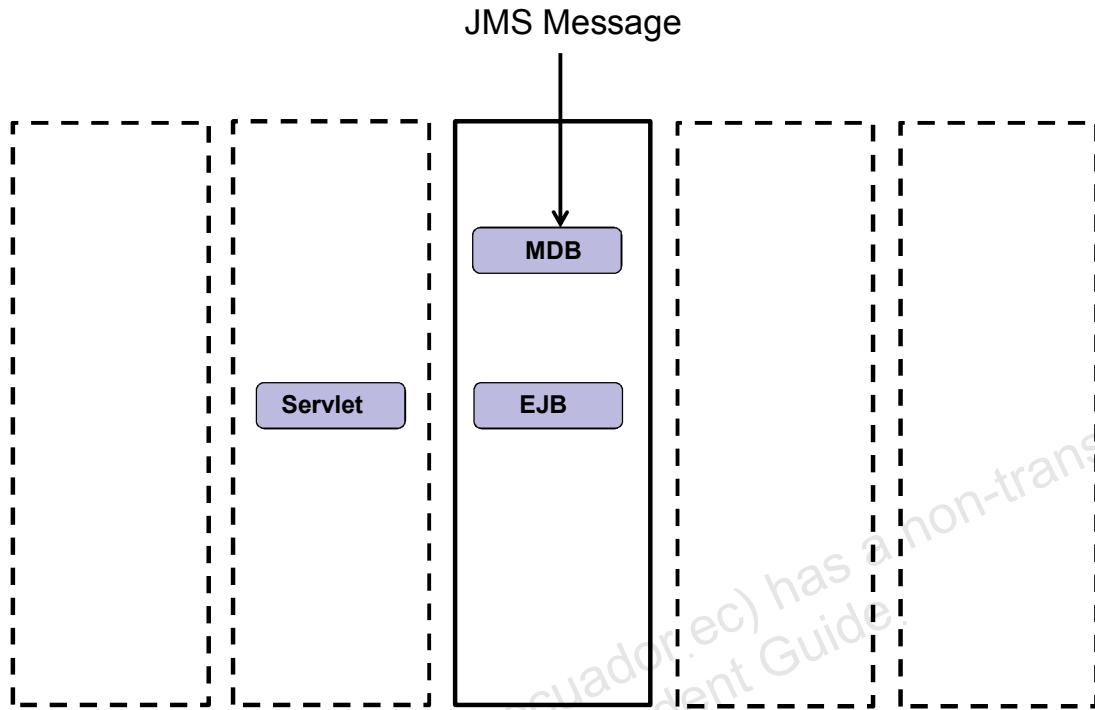
Custom Tags

Custom tags provide pre-prepared, but custom-written support to JSP technology components. You can only implement custom tags in the presentation tier. The primary role of custom tags is to provide support to JSP technology components, which subsequently provide support to presentation issues.

Servlets

You can use servlets for presentation layout, although JSF/JSP technology components are a more suitable choice from a technical point of view.

Designing for Asynchronous Communication



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Sometimes synchronous operations are unsuitable for the needs of an application. In these cases, a message-oriented middleware can provide the component integration capability that you need.

Message-Driven Beans

MDBs are intended to handle the integration with JMS. With Java EE 6, if you have need for “light” asynchrony (that is, not using JMS) then EJBs and Servlets can be called asynchronously.

Java EE Technologies and Architectural Objectives

Architectural Objectives	JPA Entity	Stateful Session Bean	Stateless Session Bean	Message-Driven Bean	Servlets	JSF/JSP	Managed Beans	Custom Tags
Long-term Application State	Yes	Maybe	Maybe	Maybe	Unlikely	No	Maybe	Maybe
Client Session State	No	Yes	No	No	Maybe	No	Maybe	Maybe
Business Process and Workflow Control	Maybe	Yes	Yes	Yes	Maybe	No	Maybe	Maybe
Presentation Process and Workflow Control	No	Maybe	No	No	Yes	Maybe	Maybe	Yes
Presentation Layout	Unlikely	Unlikely	No	No	Maybe	Yes	Maybe	Yes
Asynchronous communication	Maybe	Yes	Yes	Yes	Yes	No	Maybe	Maybe

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using the table above, how do these technologies fit?

- CDI
- JAX-WS
- JAX-RS

Lesson Agenda

- Evaluating software architectures
- Evaluating Java EE technologies
- **Creating system prototypes**
- Selecting servers and frameworks



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating System Prototypes

One of the deliverables of the architectural process is the evolutionary prototype, which is a prototype of the system that demonstrates typical use cases and conformance to nonfunctional requirements.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The evolutionary prototype has the following characteristics:

- The prototype is the basis for the production system.
- Development of the prototype requires sound design principles.
- The use of patterns in prototype development is advised.

With patterns, the architect can rely on known solutions to problems.

- Proper testing of the prototype is required.

Proper testing ensures that the prototype exhibits the characteristics that are expected during prototype execution.

Prototypes Based on Patterns

- Sound prototypes are based on prior experience and experimentation.
- When you use patterns, you gain industry experience and avoid the problems and delays that can result from experimentation.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Select patterns based on appropriateness rather than familiarity. That is, architects frequently make errors in pattern selection by choosing a pattern that they know when another pattern might have been more effective. By selecting a familiar pattern, the architect might make a decision that is detrimental to the overall quality of the system.

Prototype Validation for Standards Conformance

- After you have constructed a prototype, you must validate it against existing structures for conformance to standards.
- Prototypes that use standards in their implementation are inherently more flexible and maintainable because of the common knowledge implied by standards.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

For example, it is far easier to find a software designer who understands CORBA than it is to find a designer who understands a proprietary procedure call mechanism that is based on sockets and command constructs.

Prototype Testing

- You can use a throwaway prototype to test ideas and validate variations on implementation before you integrate these ideas or variations into the evolutionary prototype.
- Proof of concept is a critical part of developing a prototype.
- The evolutionary prototype is subject to a more rigorous testing structure.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Proof of concept is a critical part of developing a prototype. As the development process progresses and new components are introduced, the component implementations can be refined, validated, and adjusted to fit the evolutionary prototype. Then the realizations can be tested for correct functional behavior and profiled for performance characteristics. In addition, before you add new constructs to the evolutionary prototype for further development, use preliminary load testing to expose potential barriers to further development and integration.

The evolutionary prototype is subject to a more rigorous testing structure. This more rigorous testing ensures that each increment does not compromise the quality of service of the overall application. Extensive load testing and system profiling expose the current capabilities of the system. When you include alternative strategies for implementing prototypes, you can evaluate variations under similar conditions. Moreover, you can measure performance profiles and change efforts.

Measuring QoS Capabilities of System Prototypes

- While building prototypes validate a proof of concept or contribute to the evolution of the final product, testing prototypes validate the system capabilities.
- Specifically, load testing and stress testing give insight into how well the system might perform after it is brought online.
- Record the data and plot the results. The following values help to quantify the system's runtime characteristics:
 - The maximum hit rate of the prototype system
 - The maximum number of concurrent users the prototype system supports
 - The maximum allowable response time

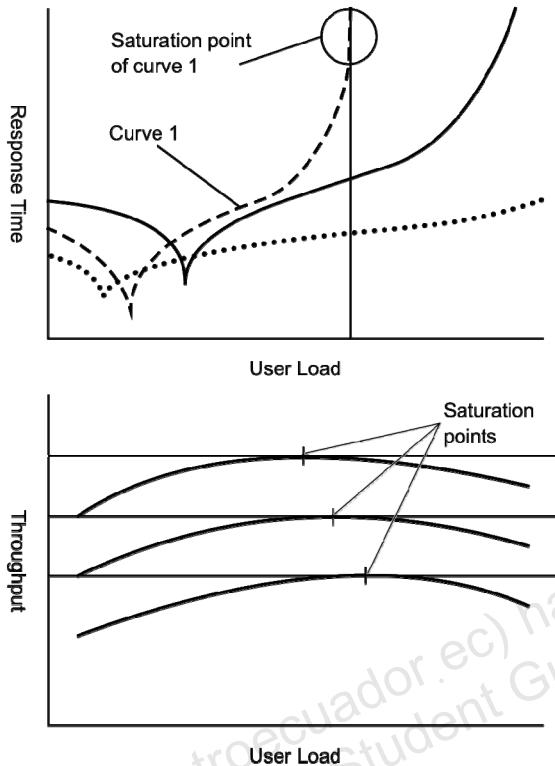


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Obtain measurements of all of the important runtime characteristics under various load scenarios, and plot graphs of the system's response over various load conditions. The graphs on the slide titled "Saturation Points" show typical loading or performance curves.

After you implement the prototypes, perform load testing of the various configurations.

Saturation Points



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Another key value is the point at which the system saturates. Saturation occurs when the performance curves begin to show a *downturn* of performance and throughput as additional load is added to the system (see). The reason for this downturn is the amount of processing effort that is used to handle resource contention and the fact that this effort draws away from the ability to process requests.

You should also track how much effort is required to modify or replace components during the testing phase. To estimate quality metrics, such as flexibility and maintainability, you can use the measured values of staff hours to complete changes and the number of lines of code needed to integrate new components into the prototypes. Use these values to gauge the complexity of the system to be implemented so that not all of the system metrics are based on runtime profiles.

Some general heuristics can help improve service levels for system designs. Use the following techniques to achieve favorable metrics:

- Try to keep the transaction load ratio low (number of transactions to number of requests)
- Make more CPUs available for processing
- Scale vertically before horizontally

Look at saturation characteristics. The ability of the system to deliver performance and throughput depends on its saturation point. If a requirement allows for a 10-second response time, but systems saturate at a response time of 5 seconds, then the prototypes must be designed with the 5-second response time.

Note: Although they are not part of the Java EE specification, a number of tools are available that allow the instrumentation of Java EE systems. These tools facilitate saturation measurements. Additionally, a considerable amount of useful data might be extracted directly from the host operating system.

Judging the Prototypes Against Architectural Goals

- After you have gathered data for system profiles and plotted the results, you must evaluate the prototypes for effectiveness and how well each prototype fulfills the architectural goals.
- The objective is to select a system that has the following characteristics:
 - Delivers the desired runtime characteristics
 - Shows growth potential
 - Is not overly complex

ORACLE

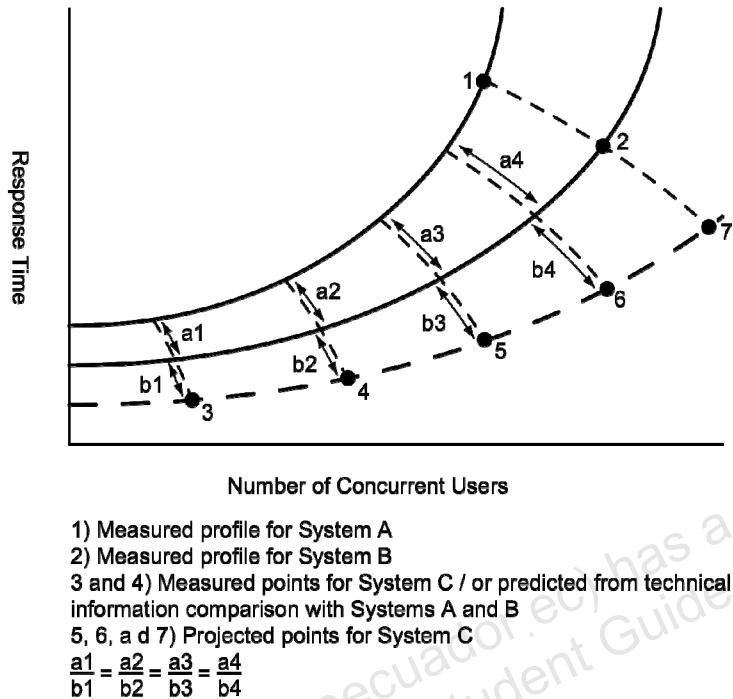
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Sometimes the graphs show a system prototype that delivers all of these characteristics, but prototypes tend to be much smaller and less expensive than the actual deployment systems. In addition, the load tests of the prototypes are completed at levels that are below the actual load levels that the production system experiences. As a result, it is difficult to estimate the performance of the production system. You can improve the estimates by using curve-fitting techniques and trend analysis.

Curve Fitting

Curve fitting matches the data points to a continuous plot. There are many techniques for selecting a best-fit curve for a set of data points. As an example, you could use quadratic expressions to generate response-time curves that are similar to the curves shown on the slide titled “Extrapolations or Trend Curves.” If the operation points that you expect are not on the load curves of the tested prototypes, you can extrapolate the data for good estimates on the performance of the system at the operation point.

Extrapolations or Trend Curves



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

To create the extrapolations, use curves that cross the performance curves at a perpendicular angle of intersection, as shown in the graph on this slide. These added curves show the trends of the performance curves. You can use the trend curves to estimate the performance curve of the production system. While this does not yield exact results, the trends shown by the curves tend to be good. Moreover, if you need a better estimate, you can gather additional data to get a more fine-grained estimate on the trend. For example, if you used 1000 user increments to gather load data, then test again with 500 or even 250 user increments to get a better view on trends and improve the extrapolation estimates.

Draw the estimated performance curve on the graph by following the curve out to the desired operating point, as shown in the slide. This allows you to estimate the performance of the production system with a reasonable amount of confidence.

What Do You Think?

What is the relationship between a Proof of Concept, a Prototype and a Reference Implementation?

Lesson Agenda

- Evaluating software architectures
- Evaluating Java EE technologies
- Creating system prototypes
- Selecting servers and frameworks



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Defining Application Server Selection Criteria

- The current market offers a wide variety of application server products from a wide range of vendors.
- Each of these products is designed to offer a common set of services to application components.
- Services can include transaction and concurrency management for clients and life cycle management for applications, to name a few.
- In addition, many of these products offer proprietary services in an effort to garner more market share by making their product more attractive than the competitor's product.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

You do not need to be familiar with the entire market of application server products.

It is better to have a set of generic selection criteria that you can use again and again to evaluate products. Then you can more closely scrutinize the feature comparisons of the products that pass this initial set of criteria to find the server that best fits the needs of your project.

While no list is comprehensive, you can begin with the following questions:

- What is the server license cost structure?
- Does the application server support load balancing, failover, and clustering?
- Can you run multiple instances on a single server system (virtual servers)?
- What software models does it support (CORBA, Java EE, .NET, or others)?
- Does the product allow for component infrastructure plug-ins?
 - Can you change the Java 2 Runtime Environment, Standard Edition (JRE), the Java DataBase Connectivity (JDBC) technology drivers, and the Java Naming and Directory Interface (JNDI) API service providers?
 - Does the server support the Java EE Connector Architecture?
- Does the server support remote application monitoring? If so, does the server provide Java Management extensions (JMX) APIs for writing your own monitoring applications?

- Does the product have a flexible memory management system?
 - Can you select and tune the number of user threads available?
 - Does the product allow for lazy data loading and cached database writes?
- What is the product's quality-of-service capability? What are the published values for scalability, availability, and security?

There are many other selection criteria that you can use to further refine the list of available server products into a list of products that are right for your application. Preliminary data for application server selection is available online in the Application Server Comparison Matrix (see <http://www.flashline.com/components/appservermatrix.jsp>). In addition, you can use benchmark results to get a starting point to evaluate the performance and throughput of the application server and other infrastructure components used in system under development.

The Java EE technology specification deliberately leaves many issues that relate to systemic qualities unspecified. This is so that different vendors can gain competitive advantage by implementing better algorithms for certain types of situations, while other vendors might consider that they have an advantage by not implementing complex algorithms because they can sell their solution at a lower price. However, because the functional issues are specified, you can always change containers if needed.

Evaluating Application Server Selection Criteria

- Evaluating an application server is a subtler prospect than just outlining a few yes-or-no questions.
- Several products might comply with your initial selection criteria, but other issues, such as performance and manageability, now come into play.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The following is a basic evaluation procedure:

1. Narrow the list to a set of likely product candidates.
2. Examine more appropriate architectural criteria that affect the overall non-functional requirements of the project itself.
 - For example, you must weigh the ease-of-use needs, both from a deployment perspective and from an administrative perspective, against the performance, availability, and scalability needs. This comes back to creating a comparison of architectural concerns based on cost.
3. Download demonstration versions of the servers that you are considering, and rate them on the following items:
 - What is your evaluation of each server's ease-of-use?
 - How much time is required to deploy a sample application on each server?
 - How adaptable is each server to changing environments?
4. Create cost estimates that contribute to the TCO of the product, based on the flexibility and manageability tests on the product.

5. Use the demonstration versions of the products to conduct performance profiling of transactions, user requests, and failure recovery time.

You can also run load testing and product burn-in cycles to profile the long-term behavior of the product. This will give you insight into the scalability and reliability of the products.

Selecting Frameworks & Libraries

Considerations:

- General
- Technical
- Managerial
- Architectural
- Integration

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Considerations:

General

- Developer skill level and spread
- Integration points (Microsoft only?)
- Exampleware available?
- Healthy online community?
- All-or-nothing adoption?

Technical

- Model-driven or UI-only? Automation possibilities?
- Configuration versus conformance (ex: XML setup versus RubyOnRails)
- Single UI (not good)
- Design lock-in - can you use an alternate
 - transport layers?
 - logging systems
 - reporting systems
- Open or closed DB schema?

- Scalability
 - architectural: large model or discrete components?
 - loading, failover
- Granularity: security per field? UI-only?
- Suitable abstractions?
- Does it have the hooks you need? can you transform or tap data at critical pts?
- Tuning, introspection support? Isolating problems by tagging events
- Threading - look for tough examples

Managerial

- Niche product? Hard-to-hire?
- How does it evolve?

Go deep, then wide

- Fail fast
- Try to identify the hardest/riskiest things, try those first

Access to the source?

- Determine its quality, smoke tests/static analysis

Architectural Layers

- Not enough or too many?
- Constraints, where are they?

Integration

- Able to wrap existing stuff?

Authentication does equate to Authorization

Just because you know who the user is doesn't mean she is allowed to do X with Y. Look for systems that vet user actions against a security module. Having to add that in after the fact is suboptimal. The system should make it easy to associate user roles to actions and specific data elements.

Additional Resources

The following references provide additional information on the topics described in this lesson:

- Bass, Len, Paul Clements, and Rick Kazman. *Software Architecture in Practice, Second Edition*. Boston: Addison-Wesley, 2003.
- Abowd, Gregory, Lan Bass, Paul Clements, Rick Kazman, Linda Northrop, and Ami Zaremski. *Recommended Best Industrial Practice for Software Architecture Evaluation*. Technical Report CMU/SEI-96-TR-025. Software Engineering Institute, Carnegie Mellon University, 1996. Available at [<http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.025.html>].



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Quiz

The goal of a good architecture is to ensure that the product under development can meet the functional and quality goals. What characteristics does a sound architecture have? (Choose all that apply.)

- a. Correctness and completeness
- b. Loosely coupled component models
- c. Conceptual Integrity
- d. Buildability



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: a, d

Quiz

Given an application requirement to keep client purchase history indefinitely, what model would provide good long-term state management?

- a. JSF components > Servlets
- b. JSF components > Servlets > Session Beans
- c. JSP/JSF -> Session Beans > JPA
- d. Swing > JMS

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: c

Quiz

Given an application requirement to hold onto the shopping cart (before purchase) of a user and the ability to scale with seasonal purchasing trends, what model will provide good client session state management?

- a. JSP > Session Beans
- b. JSP > Servlet > JPA
- c. JSP > EJB
- d. Servlet > JMS > JPA



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: b

Quiz

Creating an evolutionary prototype demonstrates typical uses cases and NFRs. What are key characteristics of this prototype? (Choose all that apply.)

- a. Is the basis for the production system
- b. If done correctly, can be used directly by users
- c. Requires sound design principles
- d. Demonstrates the performance of the system



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Answer: a, c

Summary

In this lesson, you should have learned how to:

- Describe architecture evaluation guidelines
- Evaluate Java EE technologies and their applicability
- Create system prototypes
- List and define application server selection criteria



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 12 Overview

In this practice you will:

- Discuss solutions to two real-world architectural problems.
- Practice solving a small capstone case-study.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2015, Oracle and/or its affiliates.

Iveth Tello (iveth.tello@eppetroecuador.ec) has a non-transferable
license to use this Student Guide.

Appendix A

UML 2 Quick Reference

Iveth Tello (iveth.tello@eppetroecuador.ec) has a non-transferable
license to use this Student Guide.

UML 2 Quick Reference

This appendix provides a reference to the Unified Modeling Languages (UML) 2.1.1 specification, released 02-05-2007, along with supporting material from that release.

What Is New in UML 2

For those familiar with previous versions of the UML, here is a brief summary of some of the most important changes that appear in version 2.x, often referred to as UML 2:

- Package diagrams have become official UML diagrams.
- Collaboration diagrams are now known as Communication diagrams.
- Activity diagrams have several new notations for modeling more complex activities.
- UML Use Case diagrams allow guard conditions on extends relationships.
- Interfaces use a new ball-and-socket notation.
- Sequence diagrams use interaction frames to show conditionals, loops, and parallel logic.
- A new diagram type, the Timing diagram, is available to show time-dependent changes in state of an object or a component.
- A new diagram type, the Composite Structure diagram, is available to show objects collaborating to accomplish a task.
- A new diagram type, the Interaction Overview diagram, is available to show how flow moves between interaction frames.
- Diagrams can be placed within labeled frames.

UML Basics

The Unified Modeling Language (UML) is a graphical language for modeling software systems. The UML is not:

- A programming language: It is a set of diagrams that can be used to specify, construct, visualize, and document software designs. Software engineers use UML diagrams to construct and explain their software designs just as building architects use blueprints to construct and explain their building designs. UML has diagrams to assist in every part of the application development process from requirements gathering through design, and into coding, testing, and deployment.
- A process for analysis and design: Its diagrams must be used with a process.

The UML was developed in the early 1990s by three leaders in the object modeling world: Grady Booch, James Rumbaugh, and Ivar Jacobson. Their goal was to unify the major methods that they had previously developed to create a new standard for software modeling. UML is now the most commonly used modeling language. The UML specification is currently maintained by the Object Management Group (OMG), and is available on the OMG web site at <http://www.omg.org/uml/>.

General Elements

In general, UML diagrams represent:

- Concepts, which are depicted as symbols (also called nodes)
- Relationships among those concepts, which depicted as paths (also called links) that connect the symbols

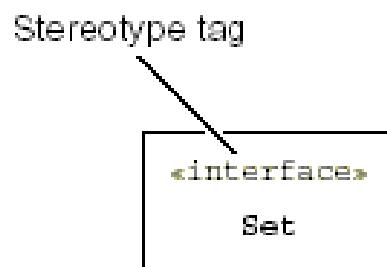
These nodes and links are specialized for each particular diagram. For example, in Class diagrams, the nodes represent object classes and the links represent associations between classes and generalization (inheritance) relationships.

Classifiers

A classifier, according to the UML 2.1.1 specification, is “an abstract metaclass” that can contain attributes and operations. A classifier is a generalization that defines a type. Classes, interfaces, use cases, and actors are all classifiers.

Stereotypes

Stereotypes enable modelers to create their own semantics for model elements (nodes and links). A stereotype is shown in a node as a word above the node name, enclosed in guillemets «». If the guillemet character is not available, use double angle brackets (<>>). shows the use of the stereotype tag <<interface>> to declare that the class node Set is a Java technology interface declaration. Stereotype tags can adorn relationships as well as nodes. There are over a hundred standard stereotypes. You can also create your own stereotypes to model your own semantics.



Some Standard Stereotypes from the UML

Stereotype	Meaning
<<create>>	A usage dependency denoting that the client classifier creates the instances of the supplier classifier.
<<document>>	A generic file that is not a source file or executable file.
<<entity>>	A persistent information component representing a business concept.
<<executable>>	An executable program file.
<<instantiate>>	A usage dependency indicating that operations on the client create instances of the supplier.
<<library>>	A static or dynamic library file.
<<script>>	A script file that can be interpreted by a computer system.
<<source>>	A source file that can be compiled into an executable file
<<utility>>	A class that has no instances. It holds a collection of class-scoped attributes and operations.

More than one stereotype can be applied to the same model element. All stereotypes are placed in a single pair of guillemets, separated by commas.

Keywords

Keywords are reserved words in the UML. They are used:

- To provide information to distinguish among different concepts that are graphically the same. For example, using the 'interface^a' keyword distinguishes that element from a simple class, although they appear in similar rectangles.
- To distinguish relationships that are graphically the same, such as lines To specify the value of a modifier such as an attribute
- To indicate a standard stereotype

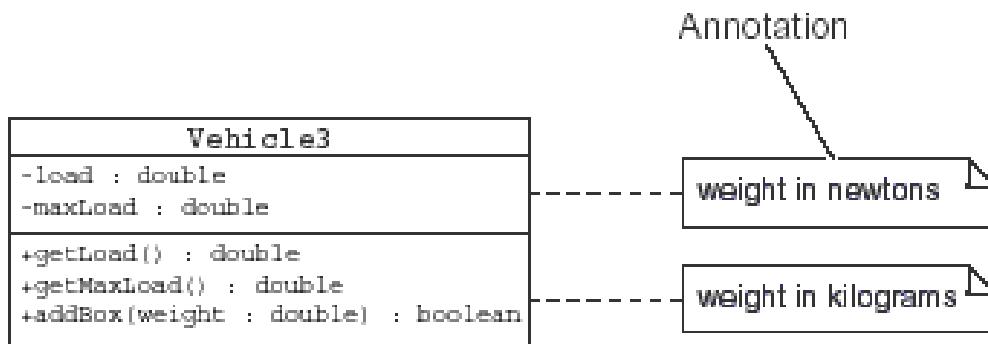
Keywords are enclosed in guillemets. However, not all words in guillemets are keywords.

Some Common Keywords

Keyword	Meaning
<<actor>>	Classifier is an actor in a use case.
<<artifact>>	Classifier is an artifact.
<<enumeration>>	Classifier is an enumeration.
<<extend>>	Relationship between use cases in which one case may optionally include the other.
<<include>>	Relationship between use cases in which one case must always include the other.
<<signal>>	Classifier is a signal.
<<statemachine>>	Classifier changes behavior when its state changes.

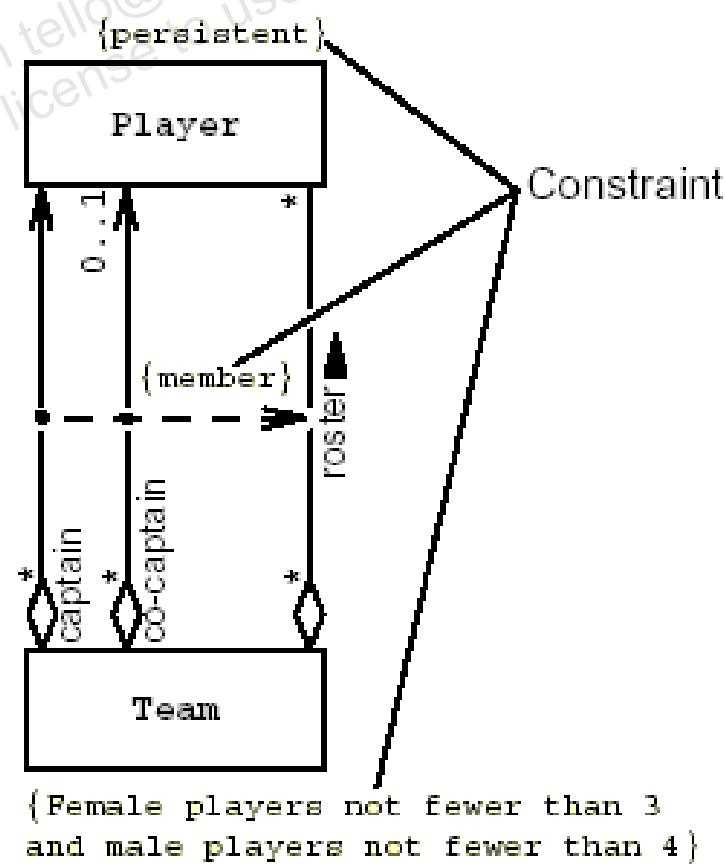
Annotations

Annotations can contain notes about the diagram as a whole, notes about a particular node, or even notes about an element within a node. The dotted link from the annotation points to the element being annotated. If there is no link from the annotation, then the note applies to the whole diagram.



Constraints

Constraints enable you to model certain conditions that apply to a node or link. The topmost constraint specifies that the `Player` objects must be stored in a persistent database. The middle constraint specifies that the captain and co-captain must also be members of the team's roster. The constraint on the bottom specifies the minimum number of players by gender.



Object Constraint Language (OCL)

The OCL is a formal notation that can be used to express constraints very precisely. OCL expressions relevant to a particular class UML diagram are usually recorded either as comments or in a separate document.

An OCL expression is like a Boolean expression in that it is an expression with operants and operands which returns true or false. When listed in a document, an OCL expression looks like:

Context Book

inv: pages > 0

An OCL Expression

The first word in the OCL below, *Context*, is followed by the name of the class to which the following constraints apply. Any number of constraints can follow. Each of those constraints begins with inv, pre, or post, to indicate which of the 3 standard defined OCL constraint types it is:

- inv: invariant: A constraint that must always be met by all instances of the class
- pre: precondition: A constraint that must always be true before the execution of the operation
- post: postcondition: A constraint that must always be true *after the execution of the operation*

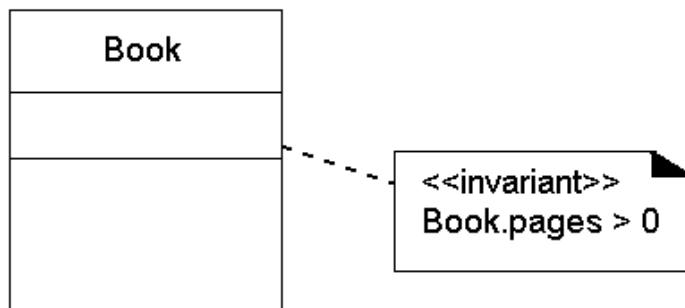
The type is followed by a colon, and then by the constraint expression. So the OCL constraint means that for any instance of a Book object, the pages attribute must be greater than zero.

Context Book

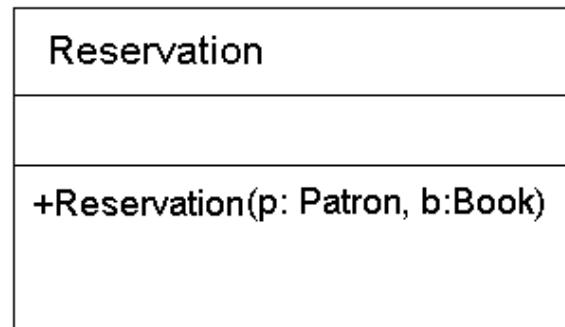
inv: pages > 0

An Invariant Constraint

Alternatively, a constraint can be represented as a comment. In this notation, the constraint type is shown as a stereotype, and the class and attribute are expressed in dot notation with the name of the class followed by a dot followed by the attribute to which the constraint applies. The comment shown indicates that the Book class has an invariant constraint that for any instance of the Book class, the attribute pages of that instance must always have a value greater than 0.



OCL can also be used in guard conditions or for results of operations. In these cases, OCL is used to indicate pre/post conditions. Here is an example of text OCL for a precondition and a postcondition:



```
context Reservation::Reservation(p: Patron, b:Book)
pre: not (b.reserved = true )
-- book cannot already be on reserve
context Reservation::Reservation(p: Patron, b:Book)
post: result->date > today
```

The OCL in the example above provides the following information:

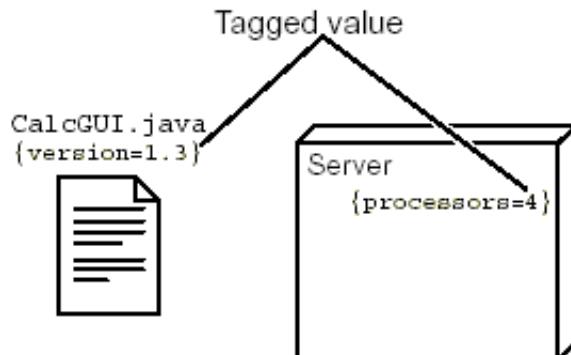
- In the context of a Reservation class, there is an overloaded constructor called Reservation with two parameters, the first of which is of type Patron and the second of type Book. The double colons show that the Reservation method belongs to the Reservation class.
- This method has a precondition which must be satisfied before the method is called. If it is not satisfied, then the result of the method call is indeterminate. This precondition is that the Book parameter's reserved attribute is not true.
- A comment, which is preceded by two dashes and ends at the end of the line, explains the precondition further.
- The method also has a postcondition. A postcondition must be true after the method completes.
- In a postcondition, the keyword result refers to the result of the operation. The \rightarrow pointer notation is used to refer to attributes of the result. Thus the last line of the constraint indicates that the date attribute of the Reservation object returned by the Reservation constructor must have a value greater than today's date after the constructor has completed.

The Full OCL Specification

The full syntax of the OCL language is not presented here. The complete OCL standard is available at <http://www.omg.org>, as a document separate from the UML specification.

Tagged Values

Tagged values are simple name=values strings that can be attached to model elements. In effect, they can be used to add new properties to nodes in a diagram. This shows examples of tagged values.



Profiles

As new software technologies emerge, the UML must provide appropriate elements to model them in order to continue to be a useful tool. A profile, which is a specification of a set of customized model elements, can be used to extend the UML to add new modeling constructs in a relatively simple manner, without requiring changes to the official UML standard.

A profile is designed to create elements appropriate to modeling a specific area such as CORBA, testing, or voice-based applications. It can use various extension mechanisms including stereotypes, tagged values, and constraints, to create an appropriate set of model elements for this purpose.

The UML metamodel is an underlying model that defines the language for defining the UML itself. Therefore, adding to the metamodel is another way to extend the UML.

OMG provides prewritten profiles for several purposes at:
http://www.omg.org/technology/documents/profile_catalog.htm.

Frame notation

UML 2 introduces a diagram frame that can be placed around a diagram. In the upper left hand corner, a label identifies the type of the diagram in the frame. The label is rectangular in shape, with a cut off lower right hand corner, as shown on page A-11 .

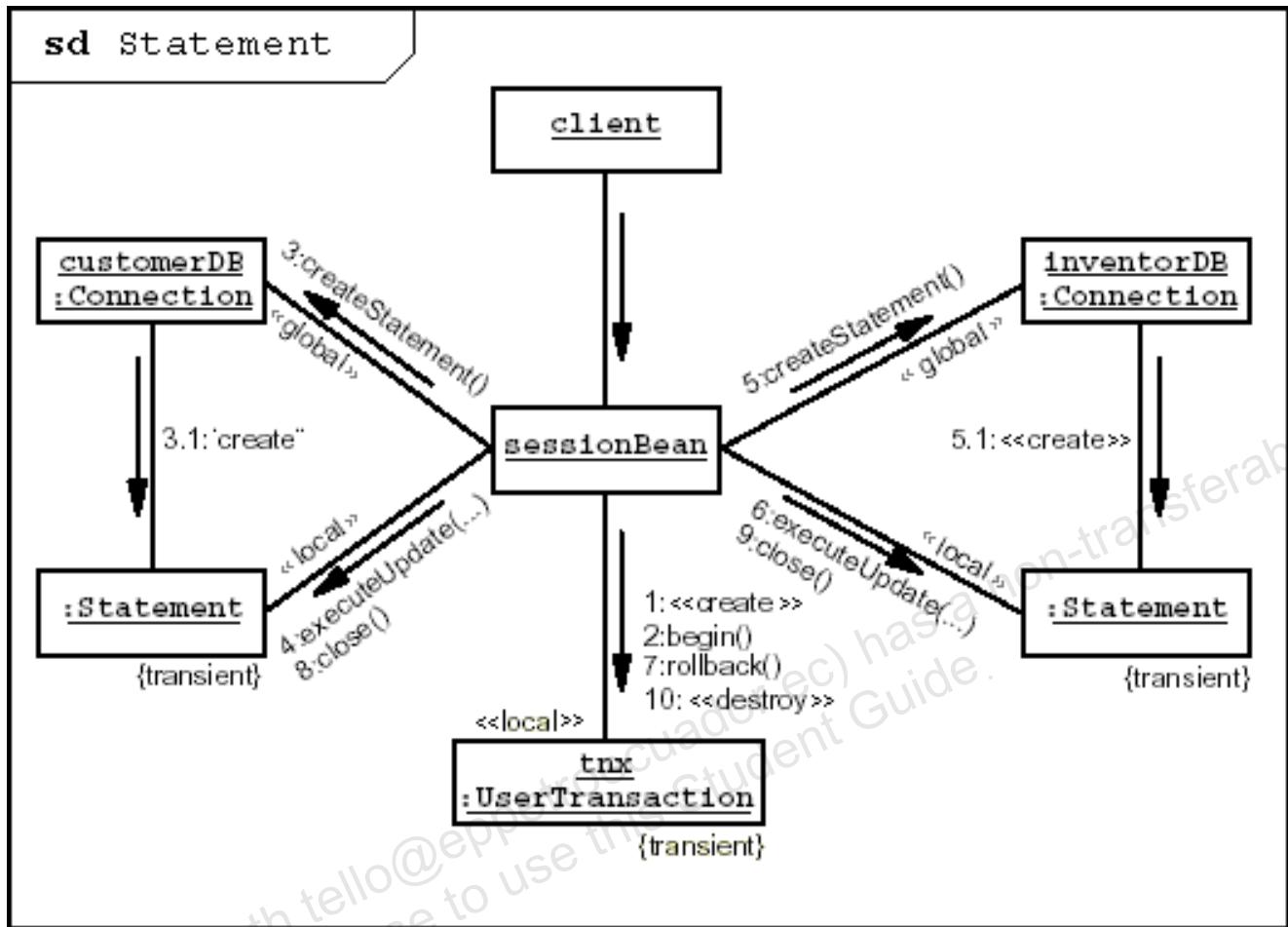
The labels provided in the UML specification are shown in the table on the next page.

Although the specification uses sd for any of the Interaction diagram types, some UML users prefer to use cd for Communication diagrams, td for Timing diagrams, and iod for Interaction Overview diagrams.

Frame Labels

Label	Frame Type
activity or act	Activity
class	Class
component or cmp	Component
interaction or sd	Interaction, which includes Sequence, Communication, Interaction Overview, and Timing
package or pkg	Package
state machine or stm	State Machine
use case or uc	Use Case

An Interaction diagram frame used with a Communication diagram.



Other diagram frame types are component, package, and use case.

UML 2 Diagrams

Classifying UML 2 Diagrams.

Diagrams in the UML 2 are classified as either structure diagrams or behavior diagrams.

- Structure Diagrams

The following diagrams present a static picture of the system or a part of the system at a point in time.

- Class Diagrams
- Component Diagrams
- Object Diagrams
- Composite Structure Diagrams
- Deployment Diagrams
- Package Diagrams

- Behavior Diagrams

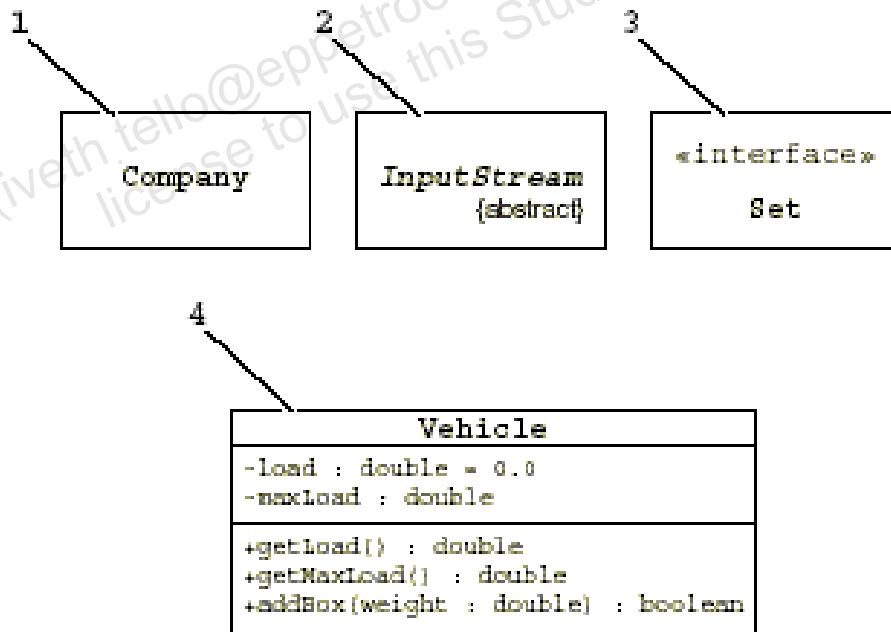
The following diagrams depict the dynamic flow of the system or a part of the system operating over time.

- Activity Diagrams
- Use Case Diagrams
- State Machine Diagrams
- Interaction Diagrams: These are a subset of behavior diagrams which include
 - Communication Diagrams
 - Sequence Diagrams
 - Interaction Overview Diagrams
 - Timing Diagrams

Class Diagram

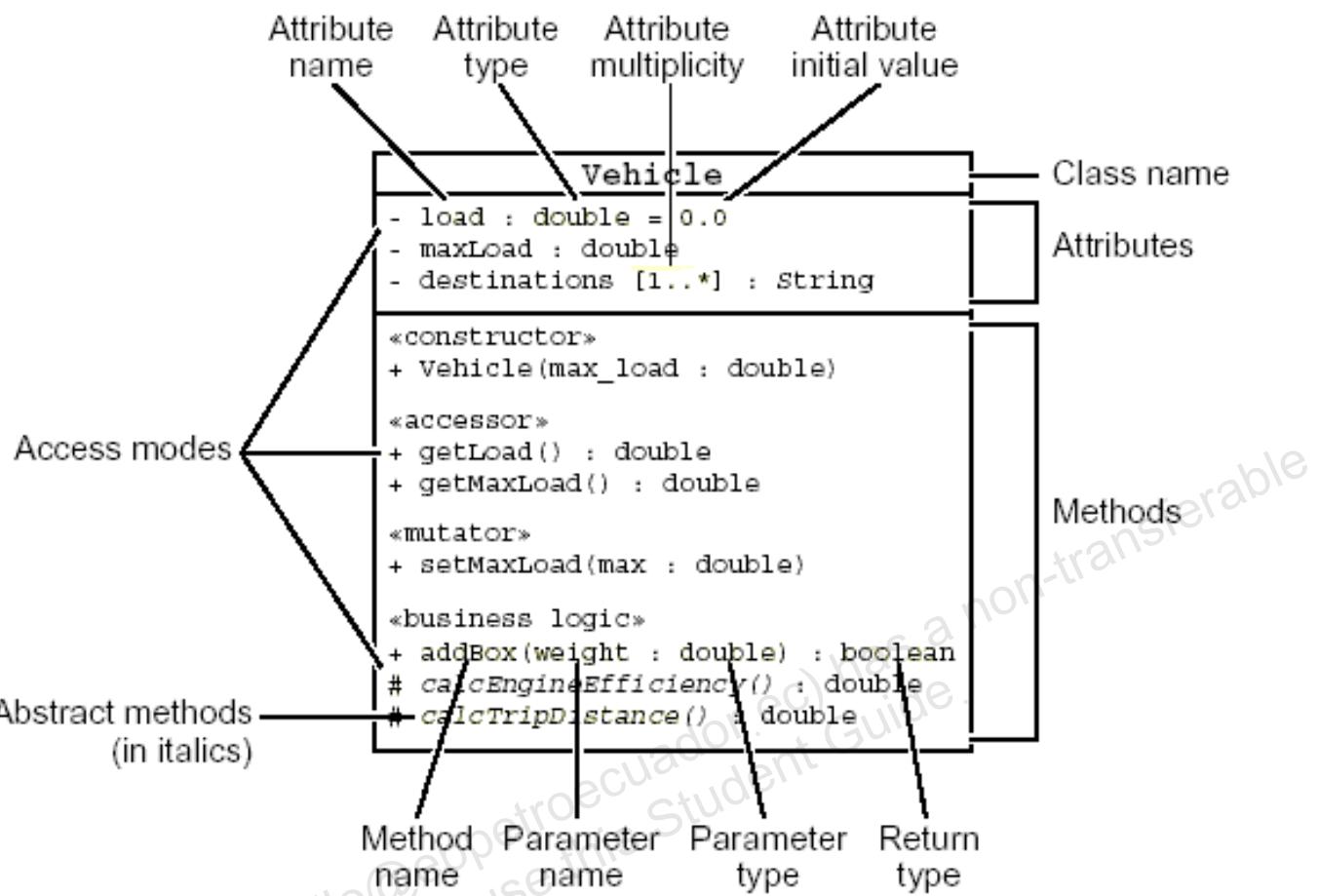
A Class diagram represents the static structure of a system. These diagrams are composed of classes and their relationships.

Class Nodes



You do not have to model every aspect of a class every time that class is used in a diagram. A class node can just be the name of the class, as in Examples 1, 2, and 3. Example 1 is a concrete class, where no members are modeled. Example 2 is an abstract class, as indicated by the constraint, and by the italic font used for the class name. Example 3 is an interface.

a concrete class, where members are fully modeled.



A fully specified class node has three compartments separated by horizontal lines:

A top compartment containing the name of the class and a possible stereotype

A middle compartment containing a set of attributes: An attribute is specified by five elements, including access mode, name, multiplicity, data type, and initial value. Only the name is required.

A bottom compartment containing a set of methods: A method is specified by four elements, including access mode, name, parameter list (a comma-delimited list of parameter name and type), and the return type. Only the name is required. If the return value is not specified, then no value is returned. The name of an abstract method is shown in italics. **You can use stereotypes to group attributes or methods together.**

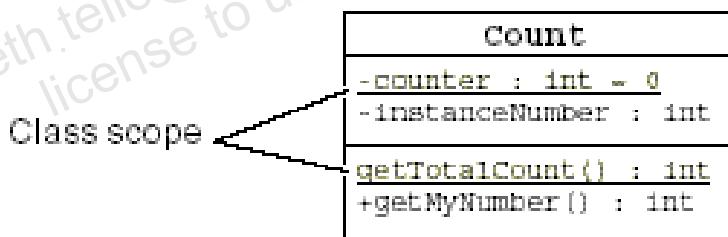
For example, you can separate accessor, mutator, and business logic methods from each other for clarity. And because there is no UML-specific notation for constructors, you can use the “constructor” stereotype to mark the constructors in the method compartment.

UML Defined Access Modes and Their Symbols

Access Mode	Symbol
private	-
package	~
protected	#
public	+

Class Scope

The figure shows an example class node with elements that have class (or static) scope. This is denoted by the underline under the element. For example, counter is a static data attribute and getTotalCount is a static method.

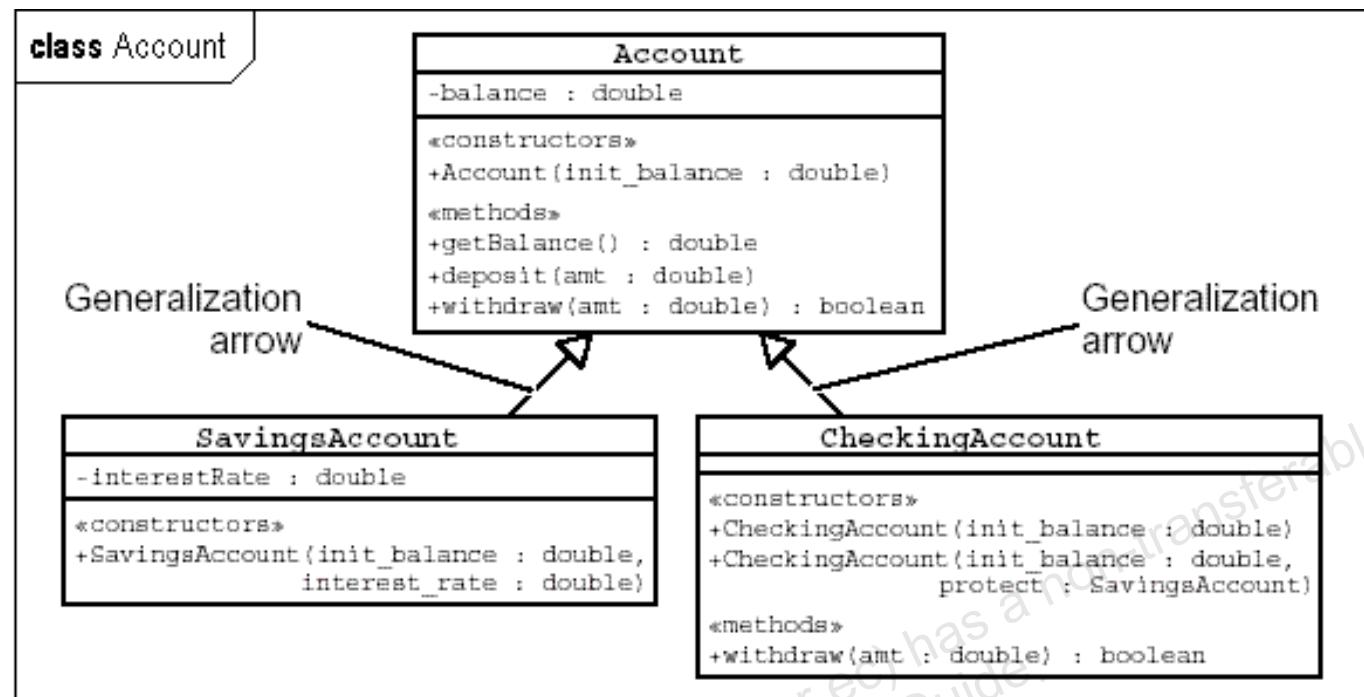


You could write the Count class from the Java programming language as shown in:

```
public class Count {  
    private static int counter = 0;  
    private int instanceNumber;  
    public static int getTotalCount() {  
        return counter;    }  
    public int getMyNumber() {  
        return instanceNumber;  
    }  
}
```

Inheritance

The figure shows class inheritance through the generalization relationship arrow.



Class inheritance is implemented in the Java programming language with the `extends` keyword. For example, the classes would be coded as shown:

```
public class Account {  
    // members  
}  
  
public class SavingsAccount extends Account {  
    // members  
}  
  
public class CheckingAccount extends Account {  
    // members  
}
```

Interface Implementation

The figure shows how to use the realization arrow to model a class that is implementing an interface.

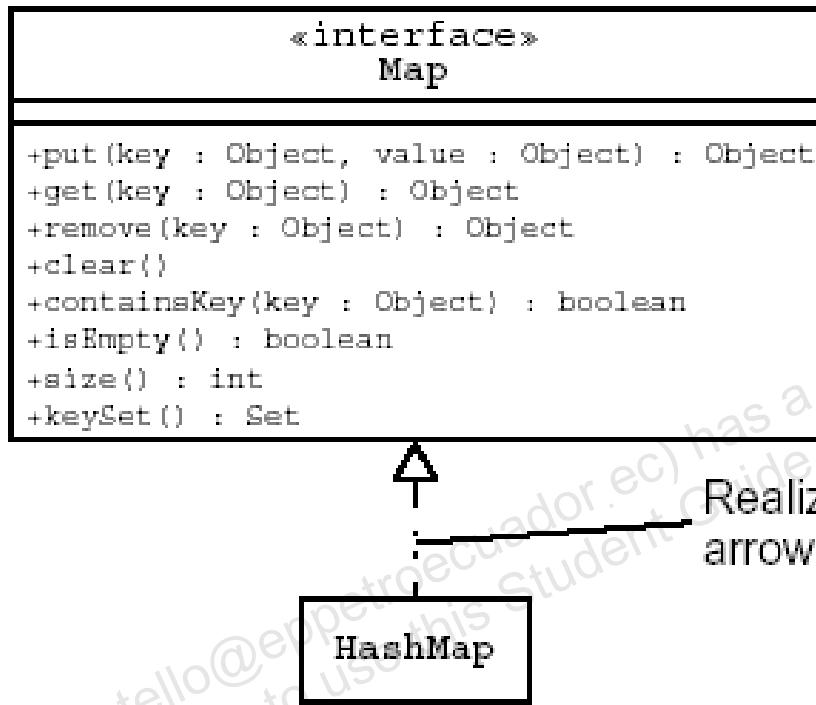
An interface is implemented in the Java programming language with the `implements` keyword.

```

public interface Map {
    // declaration here
}

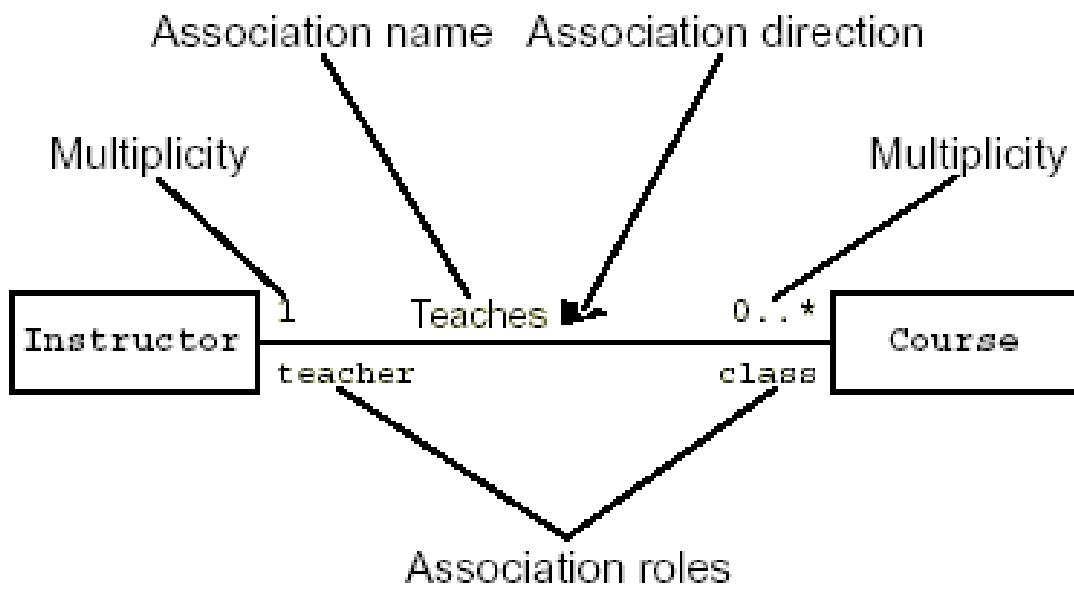
public class HashMap implements Map {
    // definitions here
}

```



Association, Roles, and Multiplicity

The figure shows an example association. An association is a link between two types of objects and implies the ability for the code to navigate from one object to other.



In the previous figure, Teaches is the name of the association with a directional arrow pointing to the right. This association can be read as “an instructor teaches a course.” You can also attach roles to each end of the association. In the figure, the teacher role indicates that the instructor is the teacher for a given course. All of these elements are optional if the association is obvious.

Multiplicity

The figure also demonstrates how many objects are involved in each role of the association. This is called multiplicity. In this example, there is only one teacher for every class, which is denoted by the 1 on the Instructor side of the association. Also, any given teacher might teach zero or more courses, which is denoted by the 0..* on the Course side. You can leave out the multiplicity for a given role if it is always one. You can also abbreviate zero or more as just *.

You can express multiplicity values as:

- A range of values: For example, 2...10 means at least 2 and up to 10.
- A disjoint set of values: For example, 2, 4, 6, 8, 10.
- A disjoint set of values or ranges: For example, 1...3, 6, 8...11.

However, the most common values for multiplicity are exactly one (1 or left blank), zero or one (0..1), zero or more (*), or one or more (1..*). Associations are typically represented in the Java programming language as an attribute in the class at the tail of the relationship as specified by the direction indicator. If the multiplicity is greater than one, then some sort of collection or array is necessary to hold the elements.

The association between an instructor and a course in might be represented in the Instructor class as shown in:

```
public class Instructor {  
    private Course[] classes = new Course[MAXIMUM];  
}
```

Representing a Multiplicity of Greater than One

Alternatively, the code below can be used:

```
public class Instructor {  
    private List classes = new ArrayList();  
}
```

An Alternate Representation of a Multiplicity of Greater than One

This code is preferable if you do not know the maximum number of courses any given instructor might teach.

Association

Association is the way in which objects interact. Objects are associated when one object uses the services or operations of another object.

It is the essence of object orientation that independent objects collaborate to accomplish larger purposes. Objects are intended to be relatively small, focused units of code that can be used by code in other objects. Commonly, one object will call a method of one or more other objects to obtain data or to perform operations. This kind of interaction is a form of association.

Aggregation

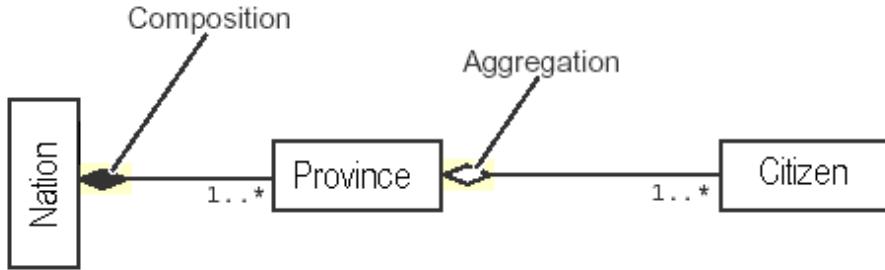
Aggregation is a type of association. Aggregation defines an object in terms of its components parts. For example, a car has an engine, wheels, seats and so on. These are all parts of the whole car. If you take any of these out, the car cannot function properly. When an object is made up of constituent parts, it is said to have an aggregate relationship with its parts. The aggregated object is a necessary part of the aggregator, even though the aggregated object can also exist independently of the aggregator.

A common way to identify classes which have an aggregation relationship is to apply the Has-A test. If the sentence “ClassA *has a* ClassB” makes sense, then the two classes are related by aggregation. For example, “A car *has an* engine.” is such a sentence. However, care must be taken when using this test. Although the sentence “A driver *has a* car.” makes sense, it is clear that a car is not a constituent part of a driver. This is a relationship of possession, not of aggregation.

Composition

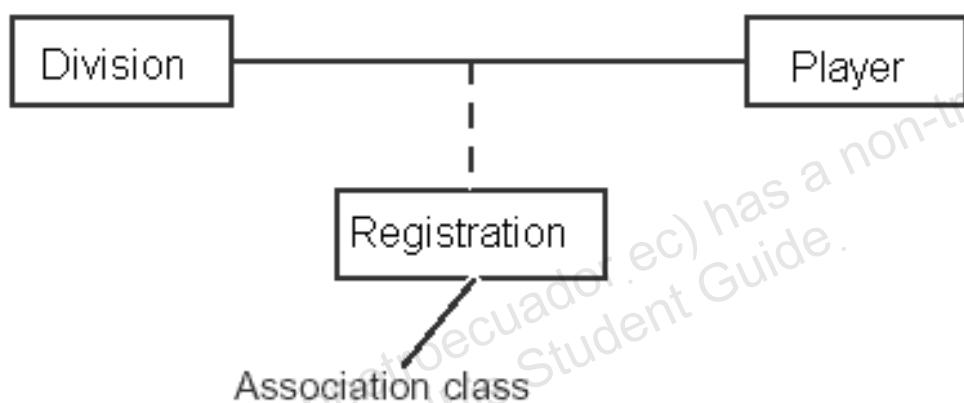
Composition is a stronger type of aggregation, so the “*has a*” test also identifies a composition relationship. However, composition differs from aggregation in that the constituent parts in composition cannot exist independently of the whole, nor can the whole exist without its constituent parts. Where this is true, composition is a more suitable relationship than aggregation. For example, a pencil contains a lead tip, and a wooden tube object. Without the lead, the pencil would not be a pencil. In programming, a composition relationship implies that a constituent part object is created with and destroyed with the containing object.

In the example below, a nation is a composition of provinces, and a province is an aggregation of citizens. A citizen might exist independently of a particular province. If a particular province were eliminated somehow, its citizens would still exist. However, if a nation ceased to exist, its provinces would no longer exist, either.

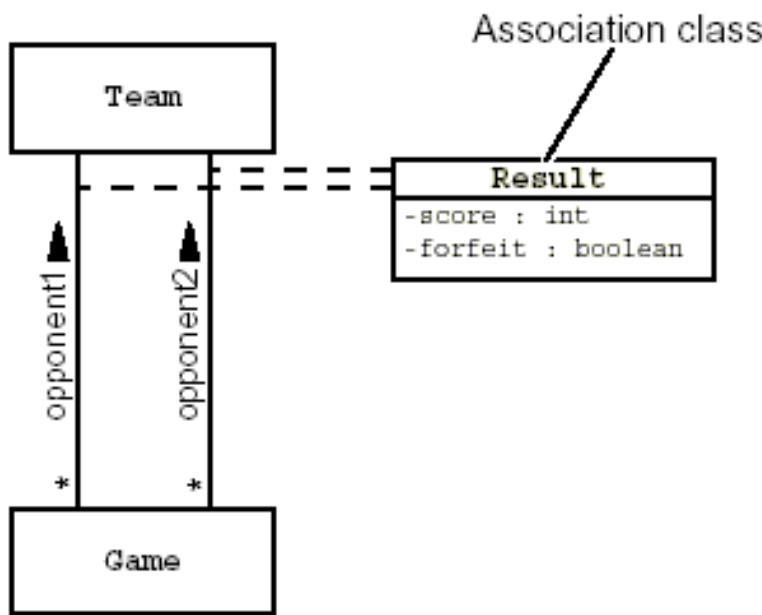


Association Classes

An association between two classes might have properties and behavior of its own. Association classes are used to model this characteristic. For example, players might be required to register for a particular division within a sports league, as this figure shows. The association class is attached to the association link by a dashed line.



The figure below shows an association class that is used by two associations and that has two private attributes. This example indicates that a Game object is associated with two opposing teams and each team has a score and a flag specifying whether that team forfeited the game.



The association class notation is used during analysis, to show a relationship that has attributes which belong to the relationship itself, rather than to either of the classes in the association. At design time, the information contained in the association class must be resolved. Two common methods of handling an association class are:

Creating a new class to hold the association class attributes

Moving the attributes from the association class into one of the other existing classes in the association

Creating a New Class from the Association Class

You can code the association class as a standard Java technology class. For example, a registration could be coded as shown in:

```
public class Registration {  
    private Division division;  
    private Player player;  
    // registration data  
    // methods...  
}  
  
public class Division {  
    // data  
    public Registration retrieveRegistration(Player p) {  
  
        // lookup registration info for player  
        return new Registration(this, p, ...);  
    }  
    // methods...  
}
```

Move the Association Class Information into an Existing Class

The association class attributes can be coded directly into one of the associated classes. For example, the Game class might include the score information as shown in:

```
public class Game {  
    // first opponent and score details  
    private Team opponent1;  
    private int opponent1_score;  
    private boolean opponent1_forfeit;
```

```

// second opponent and score details

    private Team opponent2;

    private int opponent2_score;

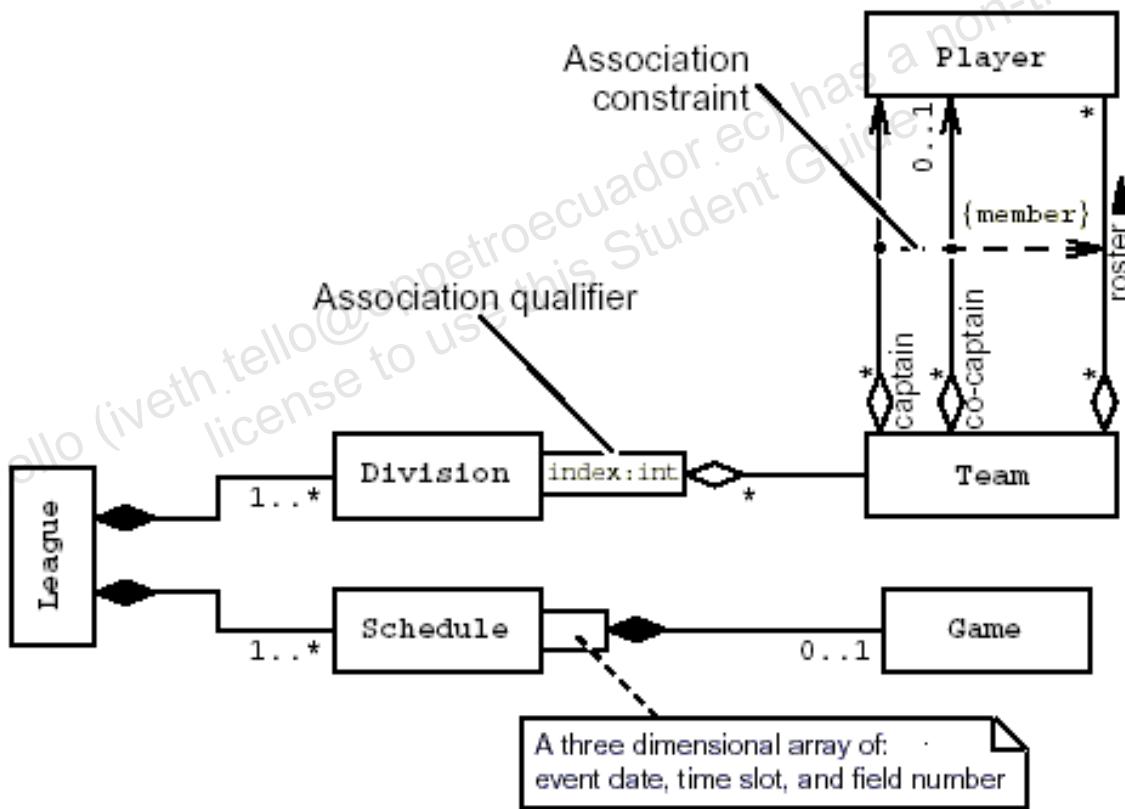
    private boolean opponent2_forfeit;

    // methods...
}

```

Association Constraints and Qualifiers

An association constraint enables you to augment the semantics of two or more associations by attaching a dependency arrow between them and tagging that dependency with a constraint. For example, the captain and co-captain of a team are also members of the team's roster.



An association qualifier provides a modeling mechanism to state that an object at one end of the association can look up another object at the other end. For example, a particular game in a schedule is uniquely identified by an event date, such as Saturday August 12, 2000, a time-slot, such as 11:00 a.m. to 12:30 p.m., and a particular field number, such as field #2.

One particular implementation might be a three dimension array. For example, in Game [] [] [], each qualifier element (date, time-slot, field#) is mapped to an integer index.

Component Diagram

A Component diagram represents the organization and dependencies among software implementation components.

The figure shows types of icons that can represent software components. The first type shows generic icons for a component. The stereotype “component” is used most commonly, either with or without a small component icon, based on the UML 1.x notation for a component, in the upper right corner. Although the specification does not explicitly permit it, the small icon is sometimes used alone, without the text stereotype.



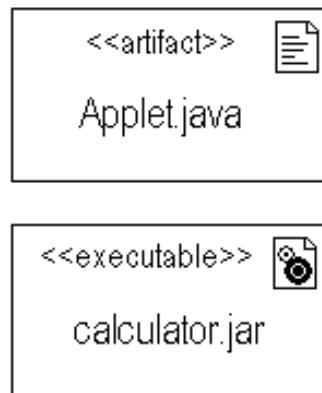
prior to UML 2



The notation for a component used prior to UML 2 is also shown in the figure. This notation is still supported in UML 2, along with the preferred newer notation.

Prior to UML 2, the definition of a component was broad enough to include items such as source code and documentation. Now components are defined in a way that is closer to the common understanding of that term: an encapsulated piece of modular software. Components are high-level, logical constructs used at design time. They interact using defined interfaces, so the actual implementation of the component is not defined and components implementing the same interface can be substituted for one another.

Physical items, which can include executable code, source code, documentation, scripts, or other such items, are now known as artifacts. The figure below shows how these can be represented with the artifact stereotype, and also shows the use of the executable stereotype.

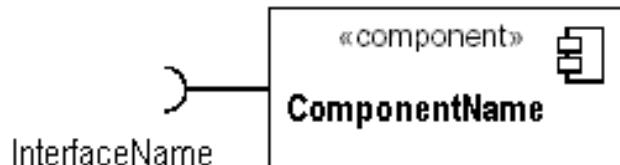


Interfaces and Components

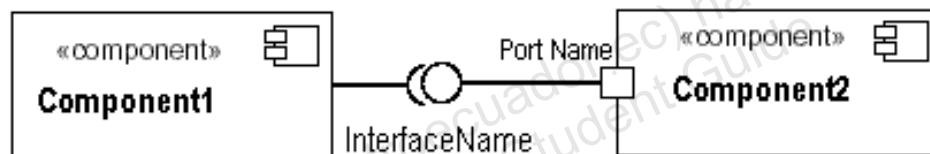
To show that a component provides (realizes or implements) an interface, use the notation shown in:



To show that a component uses or requires an interface, use the socket notation shown in:

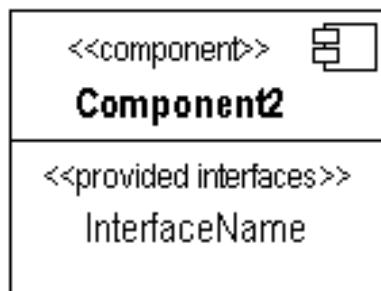
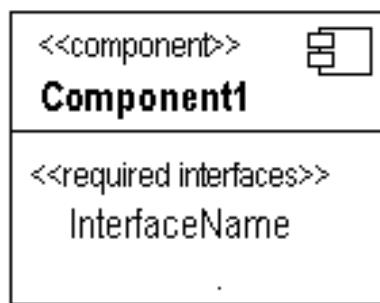


A component may provide a port, which means an interaction point through which the component communicates with the outside world, not a hardware port. The notation allows you to indicate that as shown in:



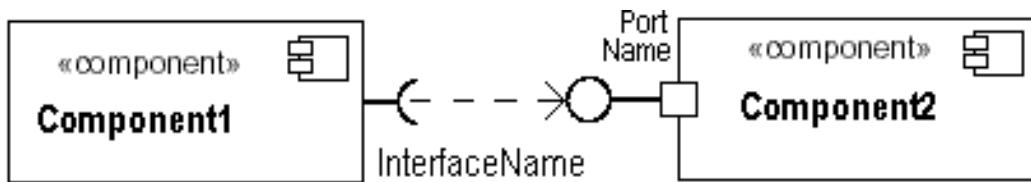
The small box to which the interface line attaches in the figure above indicates that an appropriate interface is provided at a port, which may be named.

An alternate notation for interfaces places the interface name in the component icon and uses a stereotype to show whether the interface is required or provided. This notation is shown in:

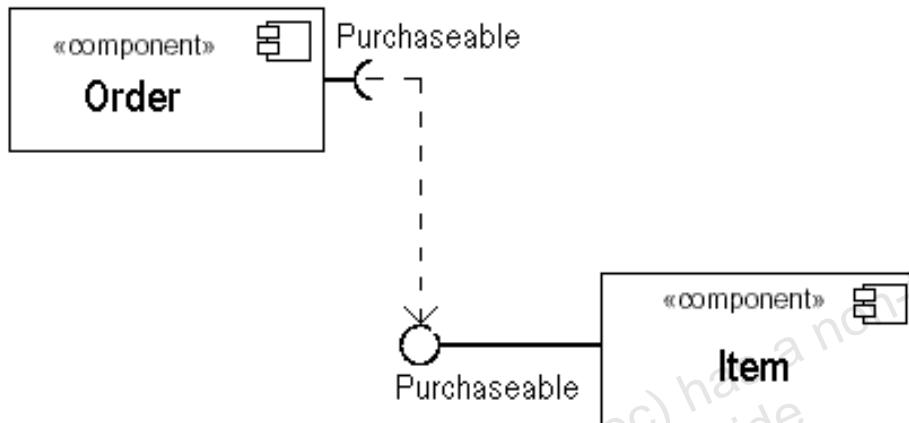


Unauthorized reproduction or distribution prohibited. Copyright© 2015, Oracle and/or its affiliates.

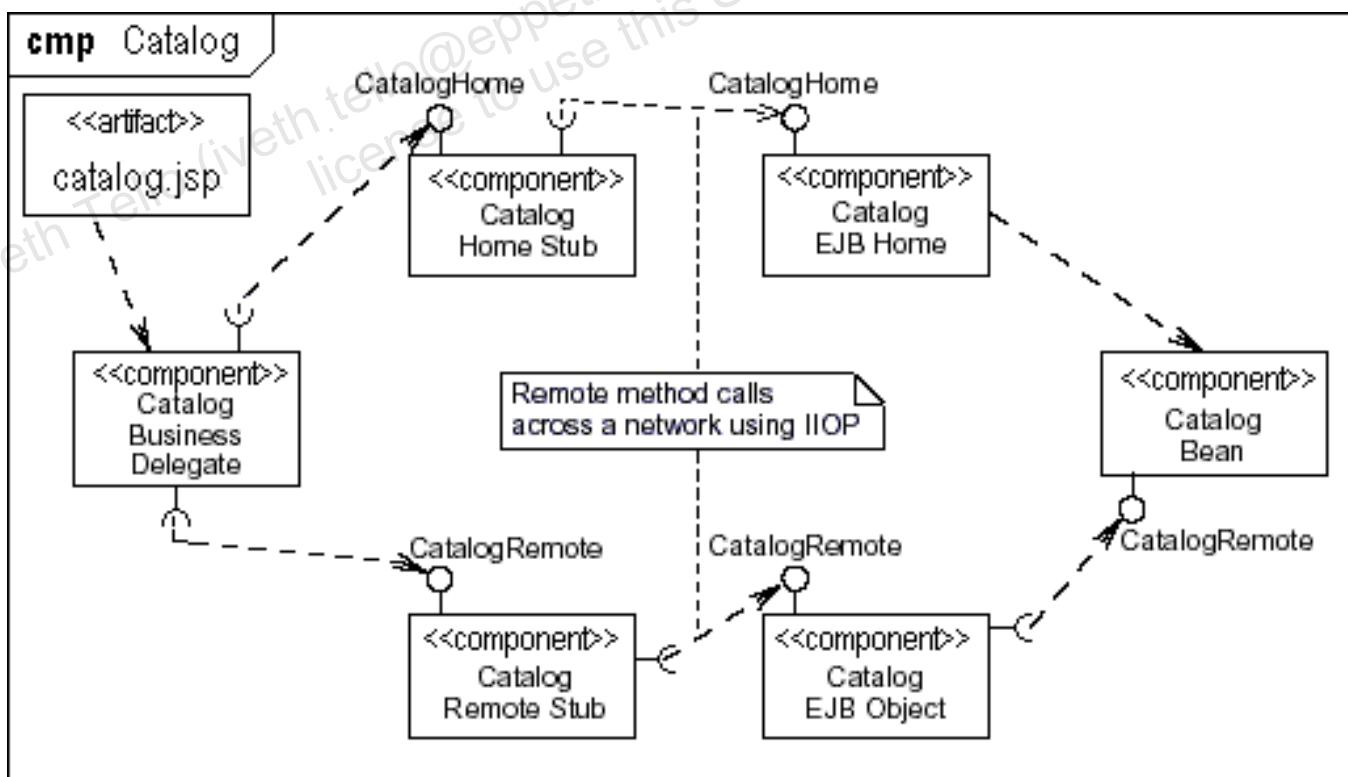
Interface notation can include dependency arrows, as shown below.



The figure below is identical to the one above, but uses actual values for the generic labels.



The figure below shows a more complex example of a Component diagram.

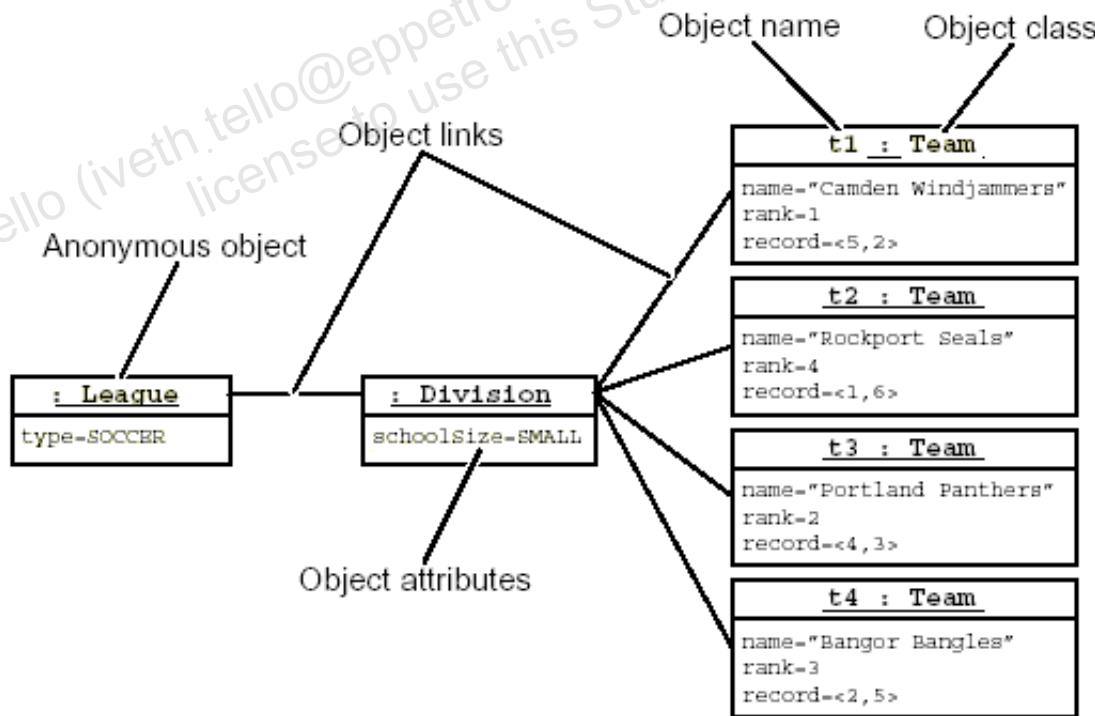


In the figure above, which is a Java Enterprise Edition (Java EE) technology example, the web tier includes a catalog JavaServer Page, which uses a catalog Business Delegate JavaBeans component to communicate with the Enterprise Java Bean (EJB) technology tier. Every enterprise bean must include two interfaces. The home interface enables the client to create new enterprise beans on the EJB server. The remote interface enables the client to call the business logic methods on the remote enterprise bean. The business delegate communicates with the catalog bean through local stub objects that implement the proper home and remote interfaces. These objects communicate over a network using the Internet Inter-ORB protocol with remote skeletons. In EJB technology terms, these objects are called EJBHome and EJBObject. These objects communicate directly with the catalog bean that implements the true business logic.

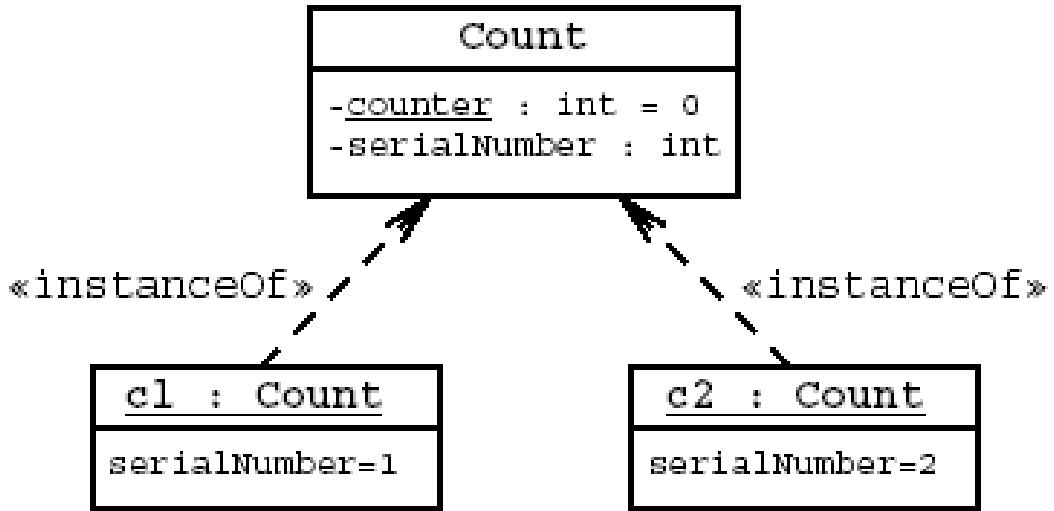
Object Diagram

An Object diagram represents the static structure of a system at a particular instance in time. These diagrams are composed of object nodes, associations, and sometimes class nodes.

The figure below shows a hierarchy of objects that represent a set of teams in a single division in a soccer sports league. This diagram shows one configuration of objects at a specific point of time in the system. Object nodes only show instance attributes because methods are elements of the class definition. Also, an Object diagram does need not to show every associated object. It just needs to be representative.



The figure below shows two objects, c1 and c2, with their instance data. They refer to the class node for Count, and the dependency arrow indicates that the object is an instance of the class Count. The objects do not include the counter attribute because it has class scope.



Composite Structure Diagram

This diagram shows a set of objects that work together for some purpose. This cooperative effort is known as a collaboration.

Note: This is not related to the diagram previously known as a Collaboration diagram, and now known as a Communication diagram.

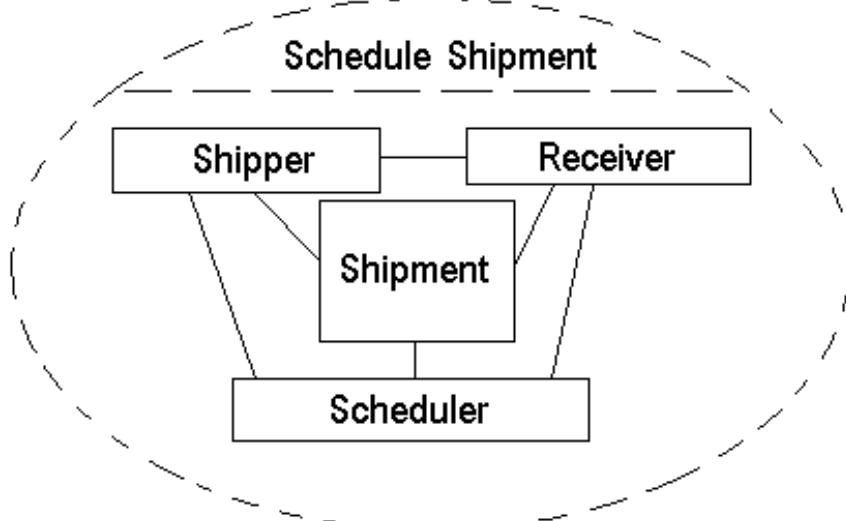
Collaboration

A collaboration specifies lifelines, which are classifiers, the communication between them, and the features they must have in order to accomplish the collaboration's purpose. This does not necessarily mean that all features of the lifelines in the collaboration must be represented.

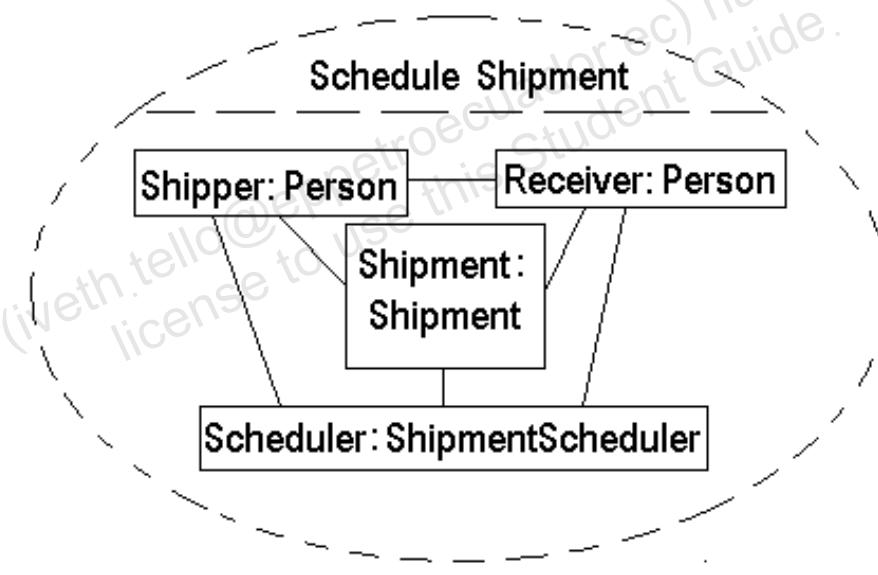
Collaborations are logical constructs. They are not instantiated, but do exist in a logical sense at runtime as actual instances playing designated roles interoperate at runtime as defined by the collaboration.

The collaboration is shown as an ellipse drawn with a dashed line. Inside the ellipse are placed the lifelines required to carry out the collaboration's purpose. Links between the lifelines, including multiplicity indicators, show how they communicate. The lifelines in the collaboration are often interface types, since it is the role, not the actual class that implements it, that is important in the collaboration.

The figure below shows a collaboration which schedules a shipment. It indicates that four lifelines are required to do this: a scheduler, a shipment, a shipper, and a receiver. Each lifeline is shown as a rectangle that contains the name of the lifeline.

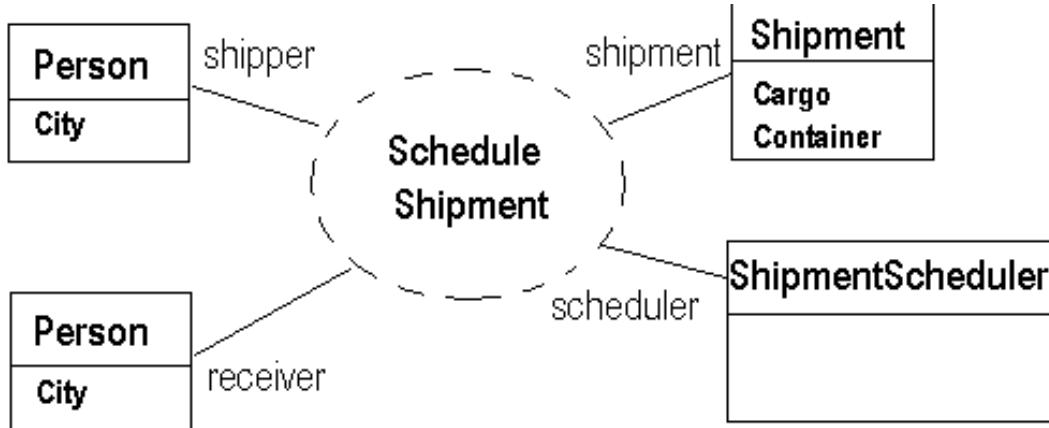


The figure below adds the names of specific interfaces that can fill these roles.



Composite Structure

Collaborations can be placed in a Composite Structure diagram. This diagram permits more information on the interfaces to be shown. It presents an example of the Composite Structure diagram, which is new in UML 2.

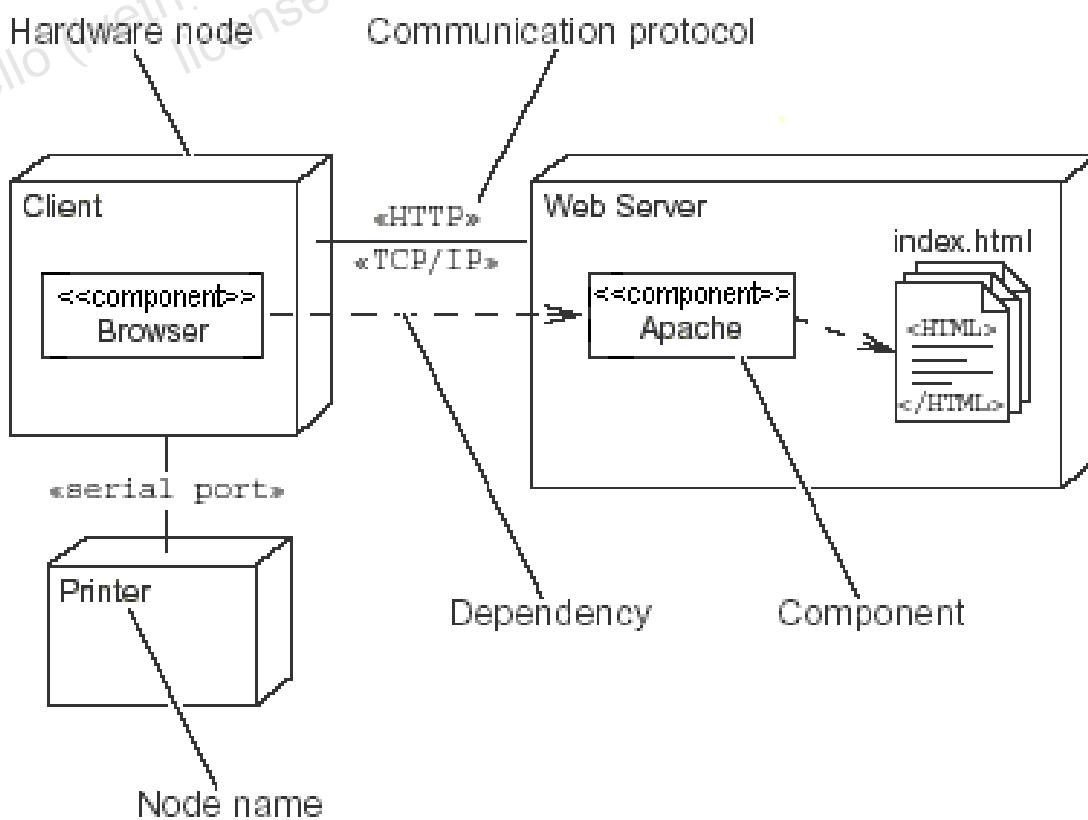


The figure above shows the collaboration name in the center of the diagram, with the cooperating elements around it and connected to it by lines labeled with lifeline names. The elements use notation which allows you to show which of the features of the lifeline are critical to the collaboration. These are not full Class diagrams, but show only those features that participate in the collaboration.

Deployment Diagram

A Deployment diagram represents the network of processing resource elements and the configuration of software components on each physical element.

A Deployment diagram is composed of hardware nodes, software components, software dependencies, and communication relationships. It shows an example in which the client machine communicates with a web server using HTTP over TCP/IP.

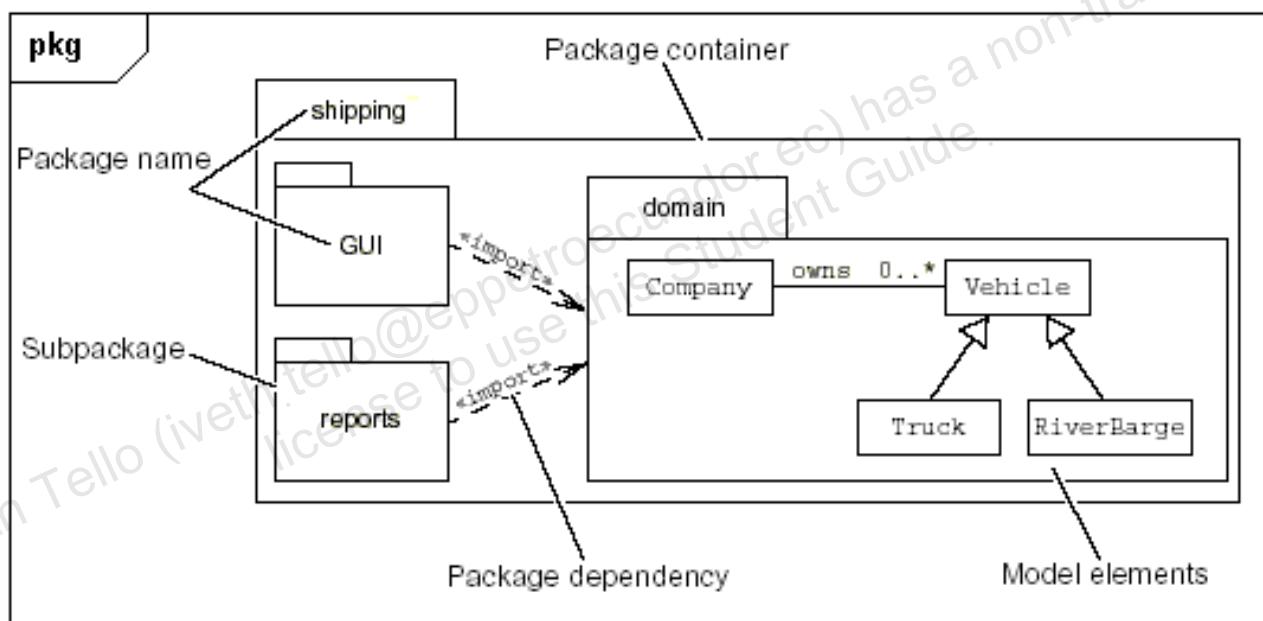


The client is running a web browser, which is communicating with an Apache web server via HTTP over TCP/IP. The browser component depends upon the Apache component. Similarly, the Apache application depends upon the HTML files that it serves. The client machine is also connected to local printer using a serial port.

You can use a Deployment diagram to show how the logical tiers of an application architecture are configured into a physical network.

Package Diagram

Packages enable you to arrange your modeling elements into groups. UML packages are a generic grouping mechanism and should not be directly associated with Java technology packages. However, you can use UML packages to model Java technology packages. The figure below demonstrates a Package diagram that contains a group of classes in a Class diagram. The shipping.GUI and shipping.reports packages have their names in the body of the package symbol rather than in the head, which can be done when the diagram does not expose any of the elements in that package.



The figure above also demonstrates a simple hierarchy of packages. The shipping package contains the GUI, reports, and domain subpackages. The dashed arrow from one package to another indicates that the package at the tail of the arrow uses (imports) elements in the package at the head of the arrow. For example, reports use elements in the domain package as shown in the figure above, and this can be shown in code as in:

```
package shipping.reports;  
import shipping.domain.*;  
public class VehicleCapacityReport {  
    // declarations  
}
```

Mapping to Java Technology Packages

The mapping of UML packages to Java technology packages implies that the classes would contain the package declaration of package `shipping.domain`. The file `Vehicle.java`, in the code below, shows:

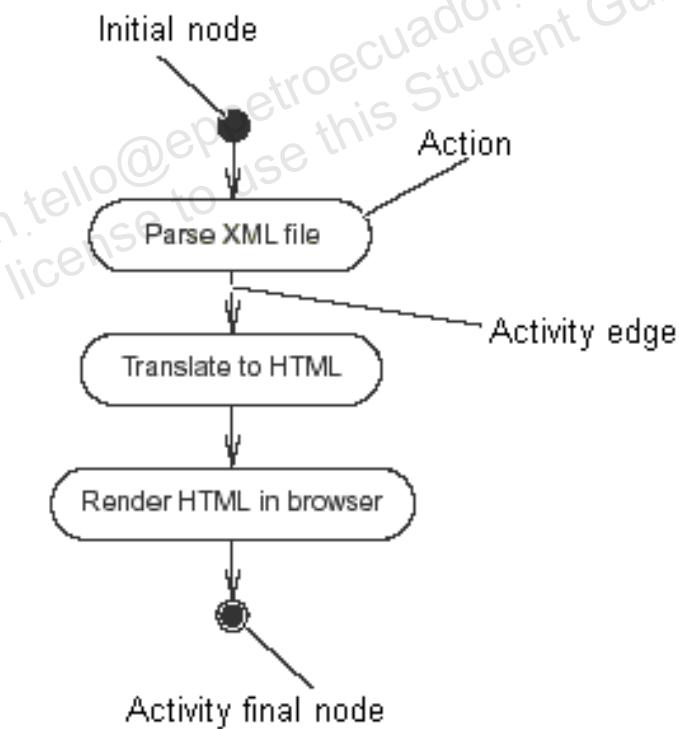
```
package shipping.domain;  
public class Vehicle {  
    // declarations  
}
```

Activity Diagram

An Activity diagram represents the activities or actions of a process without regard to the objects that perform these activities.

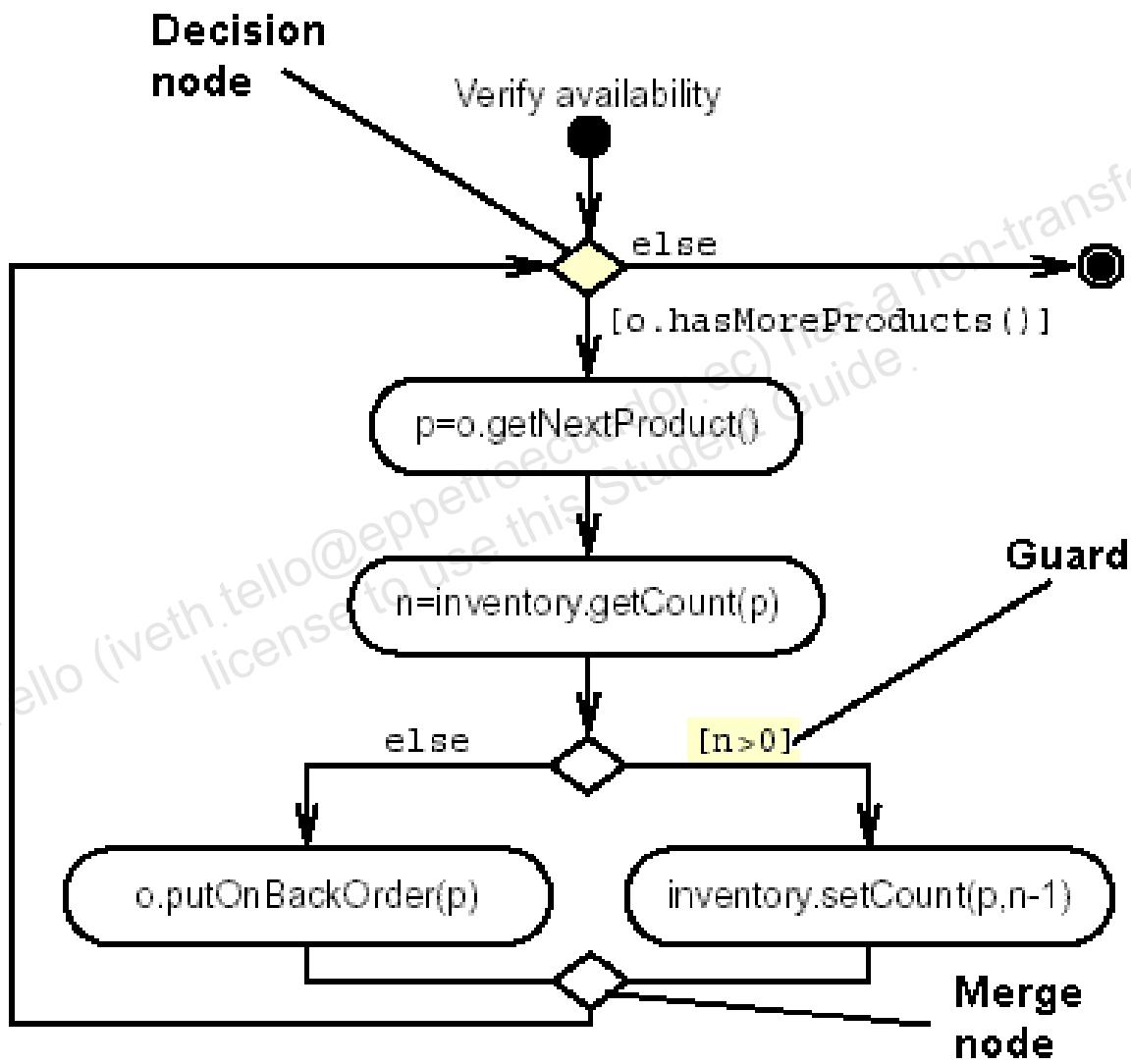
The figure below shows the elements of an Activity diagram. An Activity diagram is similar to a flowchart, depicting actions (previously known as activities) and transitions (known as activity edges) between them.

Every Activity diagram starts with a single start, known as the initial node. However, unlike flowcharts, Activity diagrams can have multiple stopping points, known as activity final nodes.

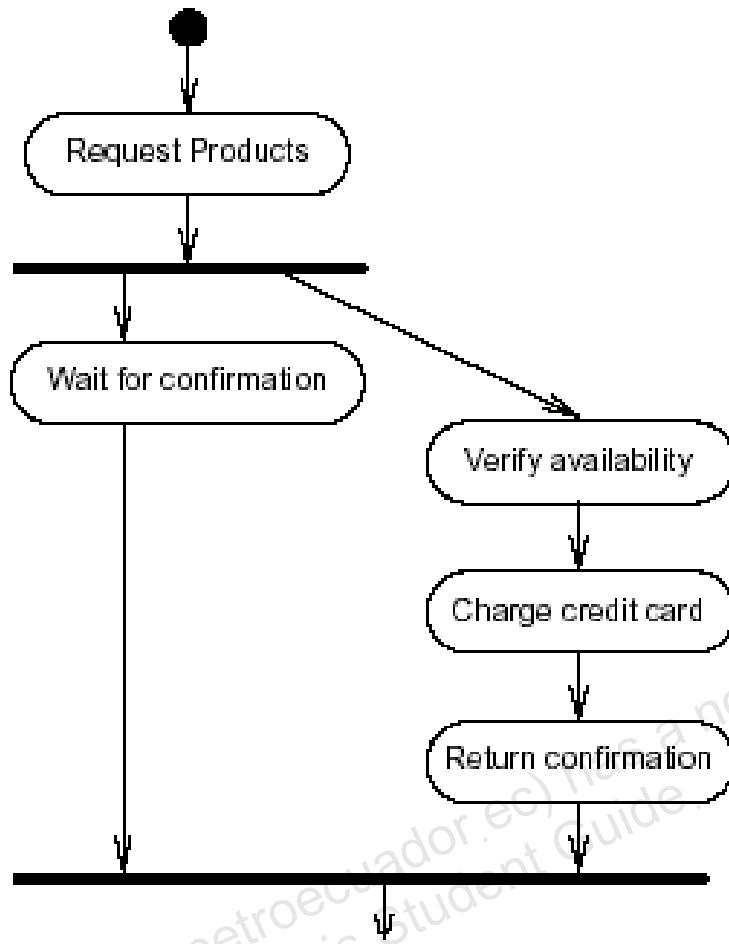


Like flowcharts, Activity diagrams can model conditionals (branching) and iterations (looping). A small diamond is used to indicate loop and branch control, in which case it is called a decision node, and also for indicating a merge that reunites two or more flows of control after a branch, in which case it is called a merge node.

The figure below demonstrates branching and looping in Activity diagrams. The diagram models the higher level activity of the use case Verify Availability of Products in a Purchase Order. On the second decision node, a guard condition indicates which transition is to be followed.

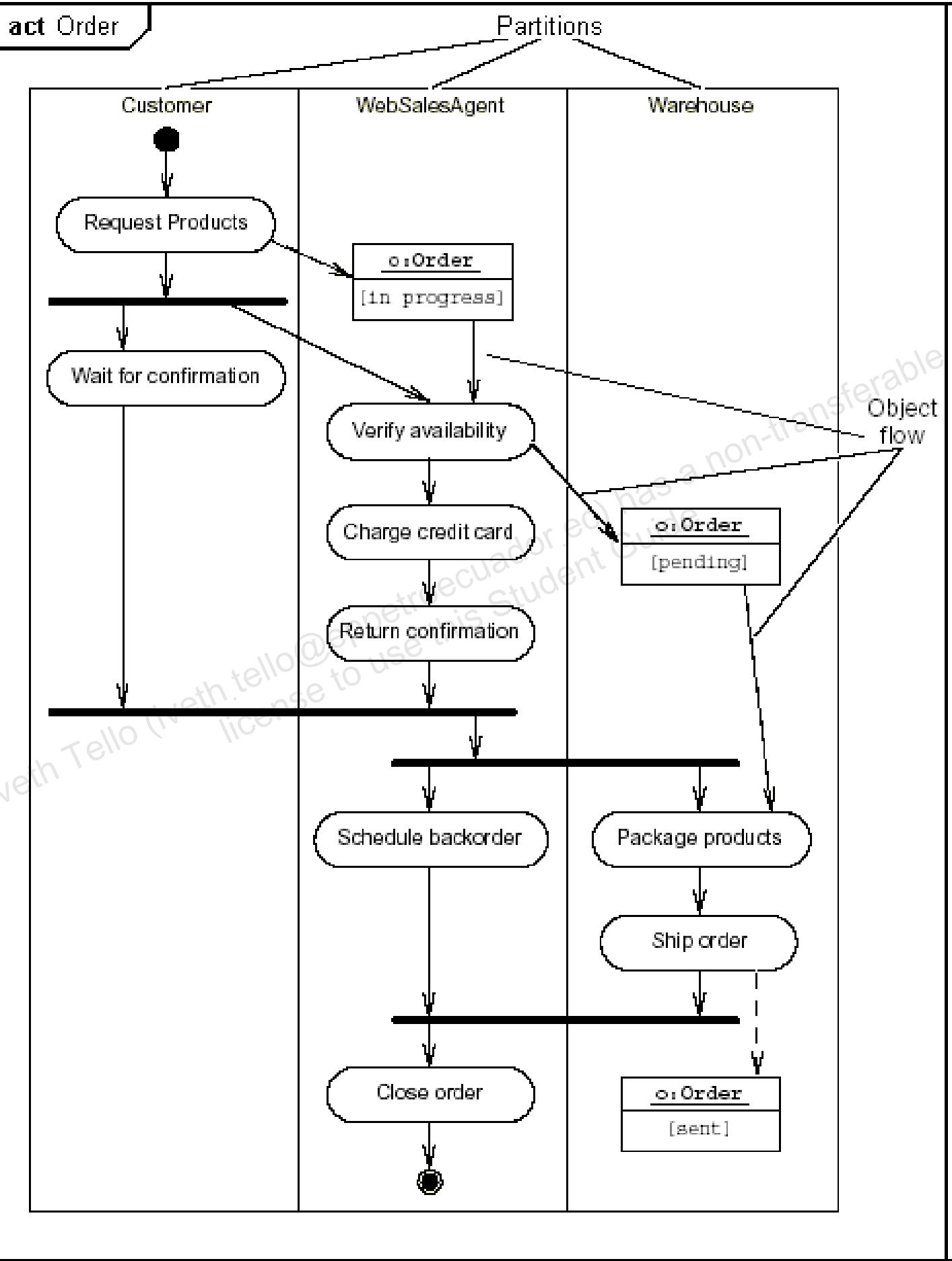


Activity diagrams can also model concurrency and synchronization of activities. For example, the Customer initiates the purchase of one or more products on the company's web site. The Customer then waits as the WebSalesAgent software begins to process the purchase order. The fork node splits a single transition into two or more transitions. The corresponding join node must contain the same number of inbound transitions. This is illustrated in the following figure:



An Activity diagram can be divided into logical partitions, also known as swim lanes. These can run either vertically or horizontally, and are used to logically group activities to show a particular view of how they can be organized. Partitions often correspond to business areas.

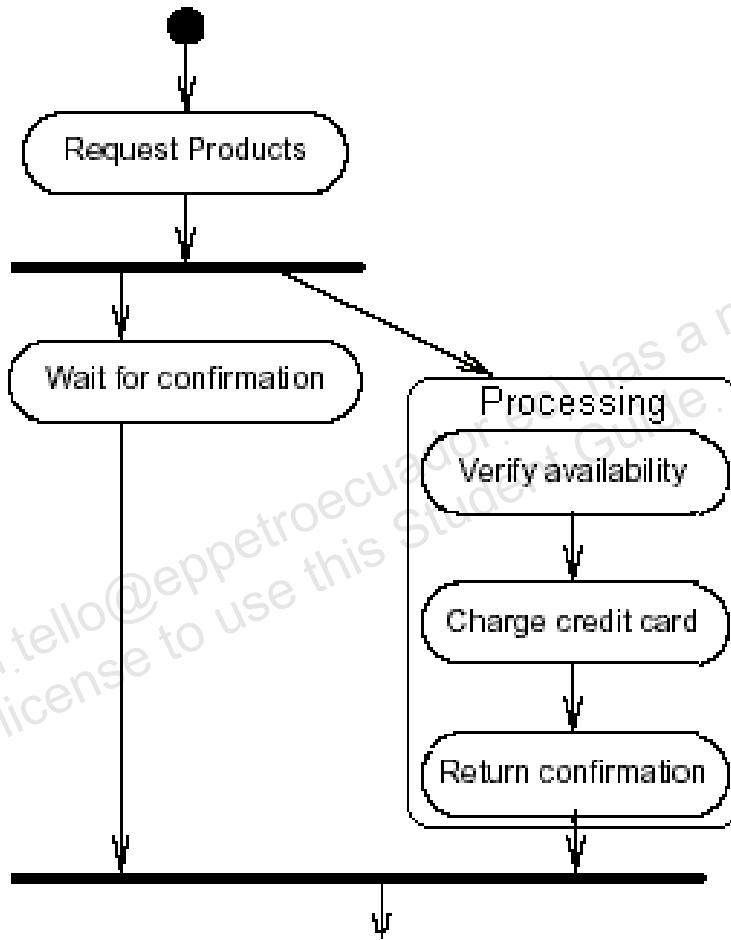
The following figure shows a more complex Activity diagram including partitions.



New in UML 2

In UML 1.x, what are now called actions were called activities. In UML 2, activities can refer either to action nodes, to object nodes, which represent individual data items or objects holding data, or to control nodes such as decision nodes or fork nodes.

Activity group refers to a grouping of actions or other activities along with sequencing. The group can have preconditions and postconditions, as can actions themselves, and can be iterated. Activities replace ActionState, CallState, and SubactivityState from previous versions of the UML. The following figure shows a structured activity group.

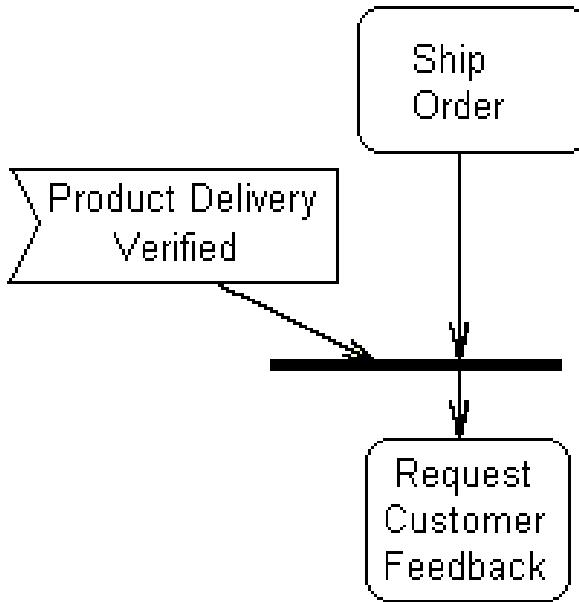


Other Activity Groups

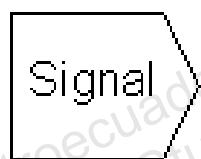
In addition to structured activity groups, loop nodes model looping behavior, expandable nodes model behavior that iterates over an input collection, and interruptible activity nodes model activities that can respond to signals.

Signal

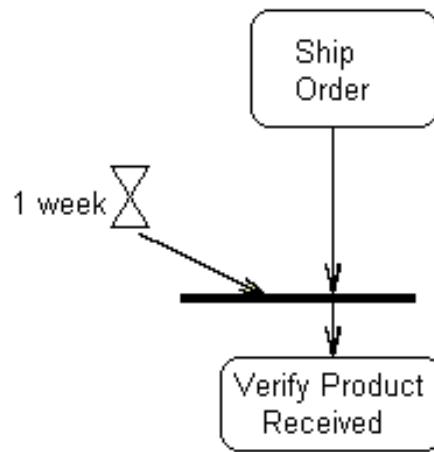
External signals, known as accept event actions, can be modeled with special icons. The banner-like icon shown in the figure below indicates an external signal.



The equivalent Send Signal action is shown in the following figure:



A time-based signal, known as an accept time event action can use a special hourglass-like icon instead. If a time signal is an input to an action, that action does not occur until the indicated amount of time has passed, or the indicated time (such as first of month) has arrived. This is illustrated below:

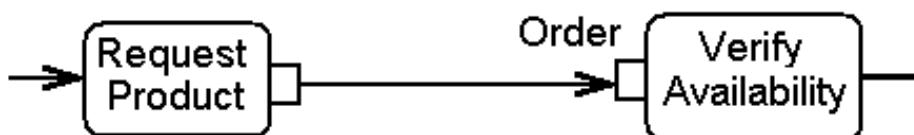


Object Flow

Object flows show the transfer of objects or data produced by one node to another node for its use. An object node, which is a rectangle without rounded corners, represents the object or data, as shown in figure below. The object flows are simple arrows.



An alternative way to notate the flow of an object is or data with a small rectangle called a pin, which is attached to the receiving action and sending action and labeled with the object or data (variable) name. The figure below illustrates:

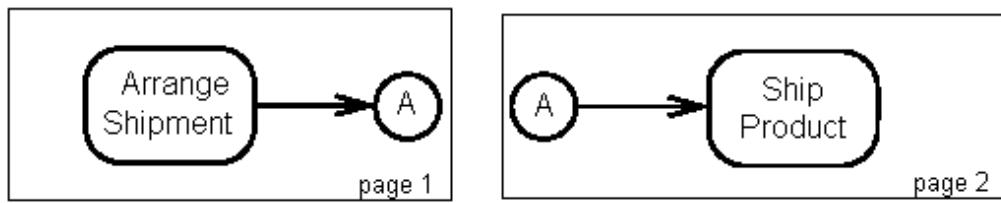


Data Store

UML 2 introduces a data store node, which supports a representation of persistent data. It is basically a special form of object node, with the stereotype “datastore.” Inputs to a data store are considered to be storing/updating data, while outputs are considered to be extracting data as in a query. An example of its use is shown below:

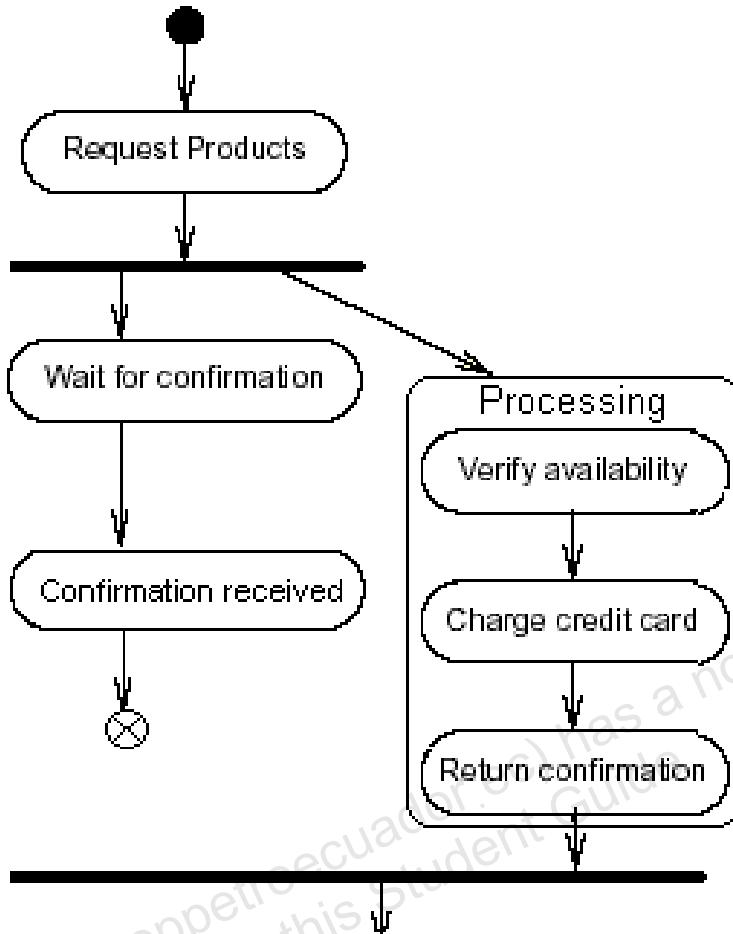


New in UML 2 is the ability to use connectors in Activity diagrams, either across page boundaries or simply for convenience in drawing lines. An example of how to draw such connectors is shown below:



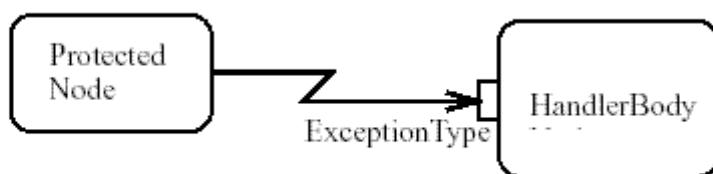
Flow Final Node

UML 2 provides a new final node, flow final, which indicates the end of a particular flow, while other actions in the activity continue. Reaching an activity final node, in contrast, ends all actions. The following figure shows an example of a flow final node.

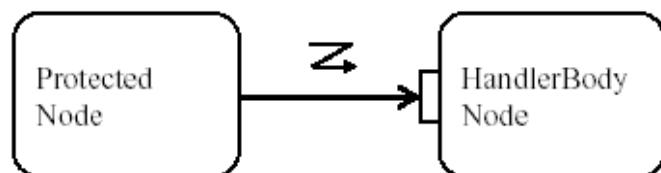


Exceptions

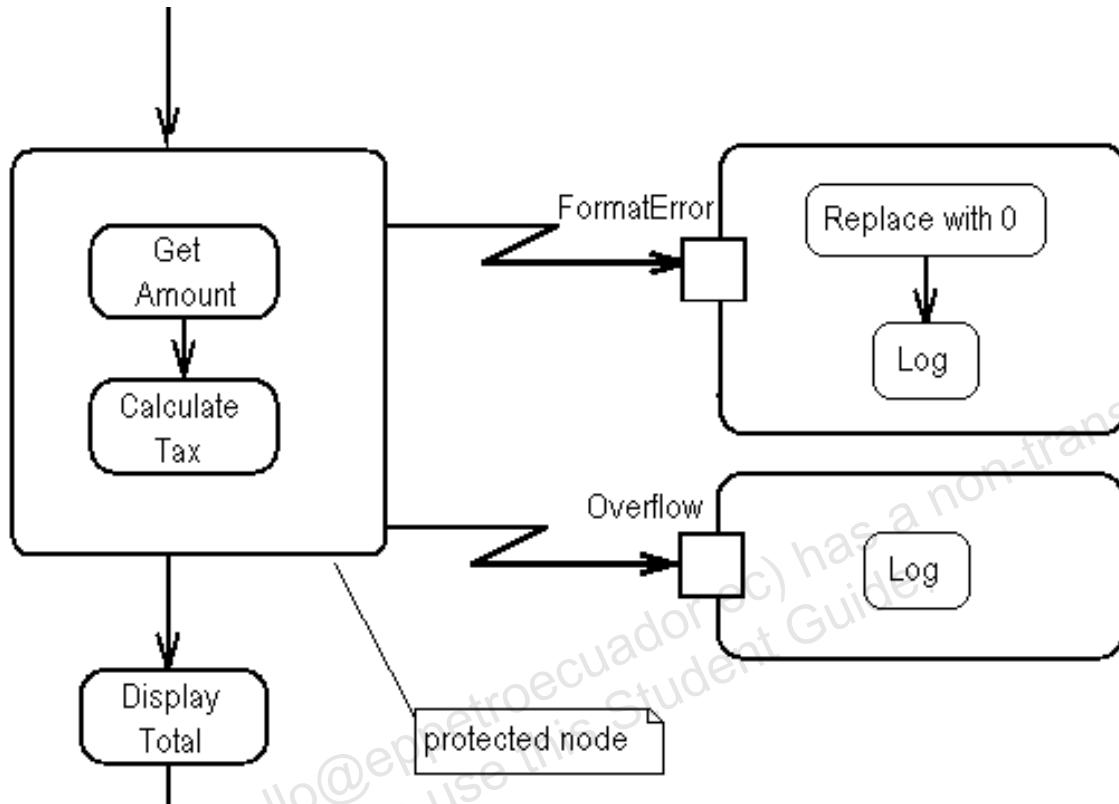
Exceptions can be modeled in an Activity diagram. A node called a protected node is the try block, and the catch block is called the handler body. If an exception is thrown, the appropriate handler body is invoked, if one exists. Otherwise, the exception is propagated to the enclosing protected node if one exists. The notation is shown below:



An alternate notation is shown below:



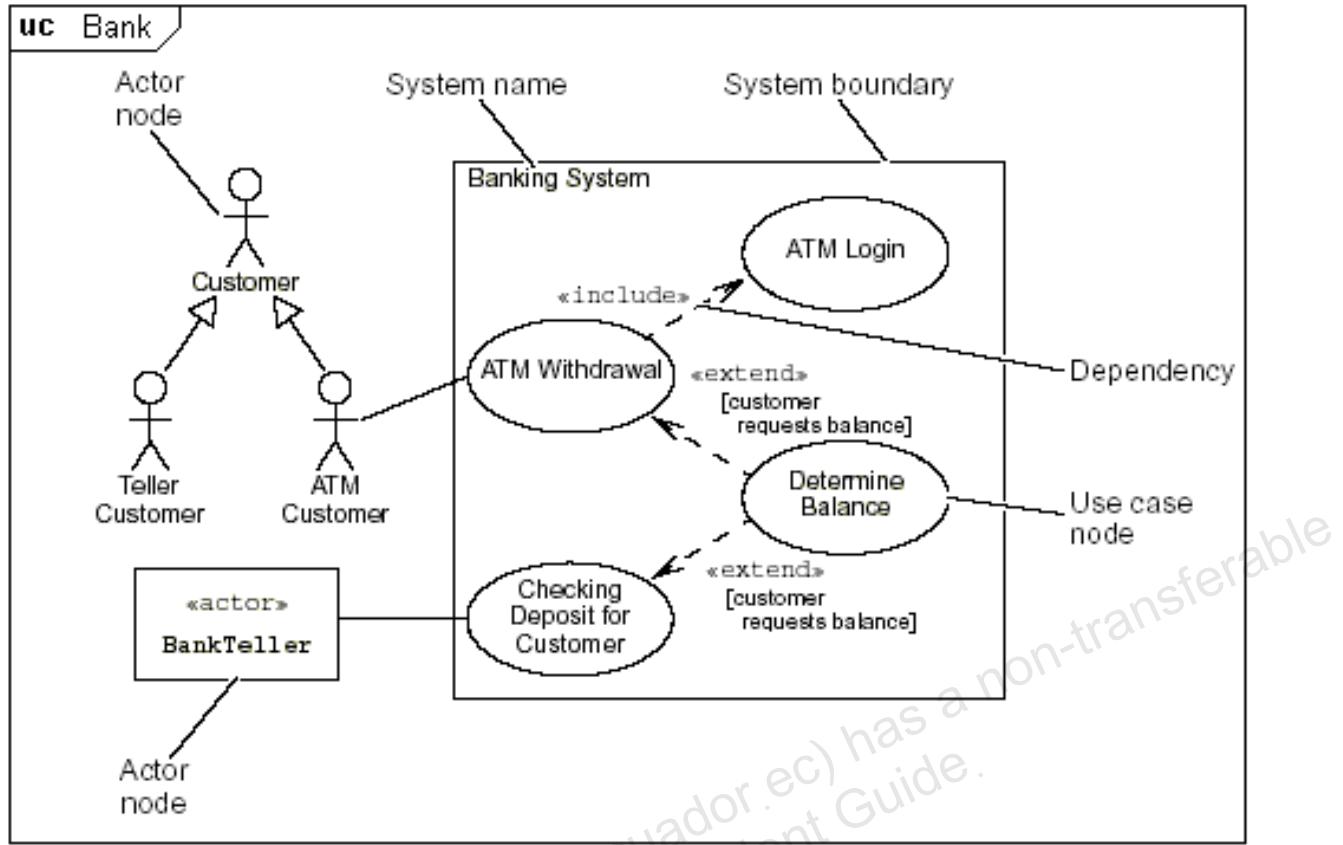
The figure below shows an example of using exception notation in an Activity diagram. The two activities on the left, getting an amount and calculating a tax, are enclosed in a protected node. If a format error occurs during either of these activities, the activities in the top handler on the right is performed. If an overflow error occurs during either of the activities, the activities in the lower right handler is performed.



Use Case Diagram

A Use Case diagram represents the functionality provided by the system to external users. The Use Case diagram is composed of actors, use case nodes, and their relationships. Actors can be humans or other systems.

The figure below shows a simple banking Use Case diagram. An actor node can be denoted as a stick figure (as in the three Customer actors) or as a class node with the stereotype of "actor." Actors can be related hierarchically, as shown.



A use case node is denoted by a labeled oval. The label indicates the activity that the system performs for the actor. Use case nodes are grouped into a system box, which is usually labeled in the top left corner. A solid line connects the appropriate actor(s) to the use case node.

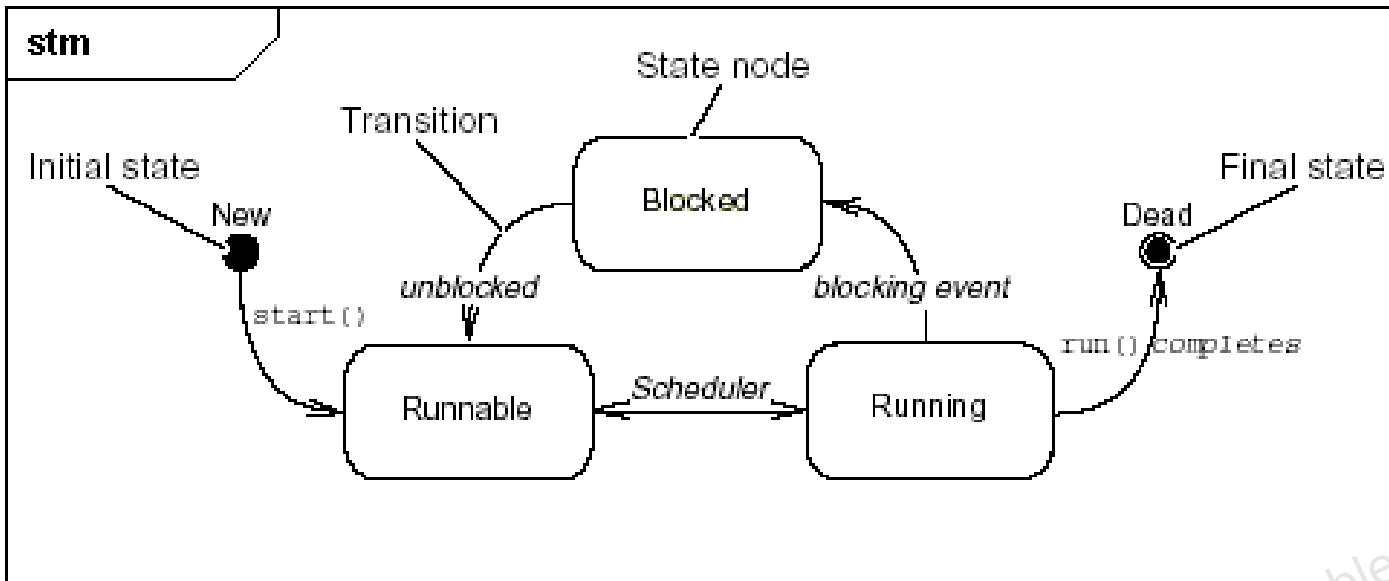
Use case nodes can depend on other use cases. For example, the ATM Withdrawal use case includes the ATM Login use case, as shown by the stereotype “include.” Use case nodes can also extend other use cases to provide optional functionality. For example, the Determine Balance use case can be used to extend (add steps to) the Checking Deposit for Customer use case. A guard condition is used to show that the extension happens only when the user requests it.

State Machine Diagram

A State Machine diagram represents the states of a class, and its responses to external and internal trigger events.

What UML 2 calls a State Machine diagram is known by several other names including State diagram and State Transition diagram. Before the UML 2, it was known as a Statechart diagram.

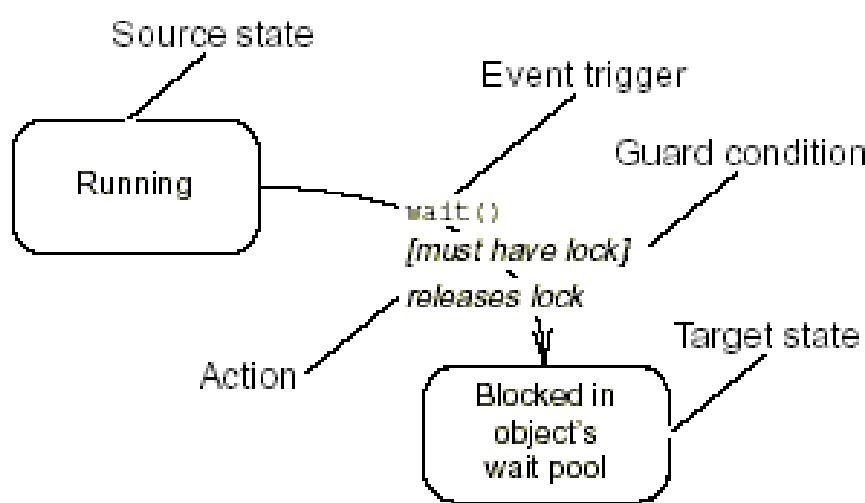
The figure below shows an example State Machine diagram. Every State Machine diagram should have an initial state (the state of the object at its creation) and a final state. By definition, no state can transition into the initial state and the final state cannot transition to any other state.



Transitions

A transition in a State Machine diagram has five elements:

- Source state: The state affected by the transition
- Event trigger: The event that makes the transition eligible to fire, providing its guard condition is satisfied, when it is received by the object in the source state
- Guard condition: A Boolean expression used to determine if the state transition should be made when the event trigger occurs
- Action: A computation or operation performed on the object that makes the state transition
- Target state: The state that is active after the completion of the transition



Interaction Diagrams

There are four types of Interaction diagrams defined in the UML.

- Communication Diagram

This diagram was known as a Collaboration diagram prior to UML 2.

- Sequence Diagram

The way this diagram handles loops has been changed in UML 2.

- Interaction Overview Diagram

This diagram is new in UML 2.

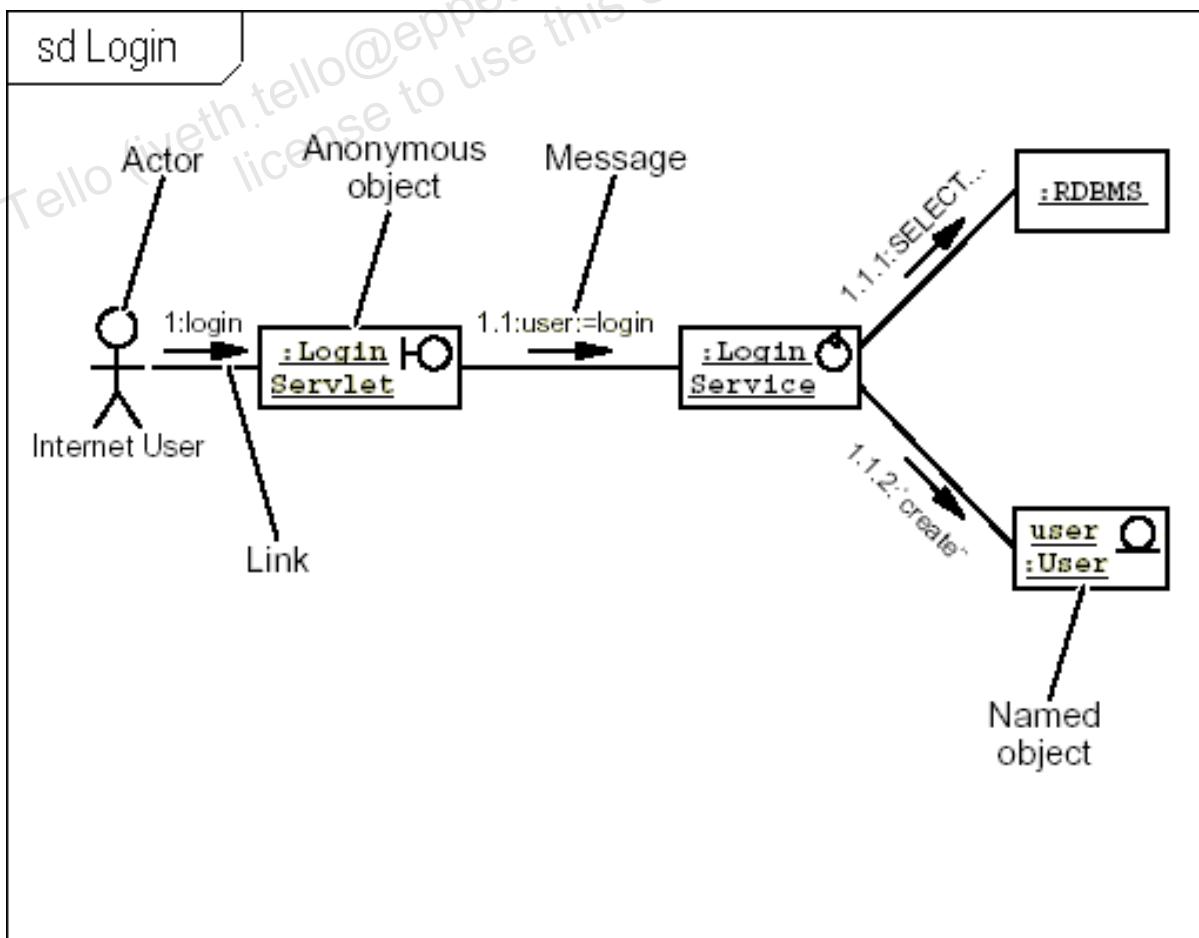
- Timing Diagram

This diagram is new in UML 2.

Communication Diagram

A Communication diagram (previously known as a Collaboration diagram) represents a task carried out by several objects. These diagrams are composed of objects, their links, and the message exchanges that accomplish the task or behavior.

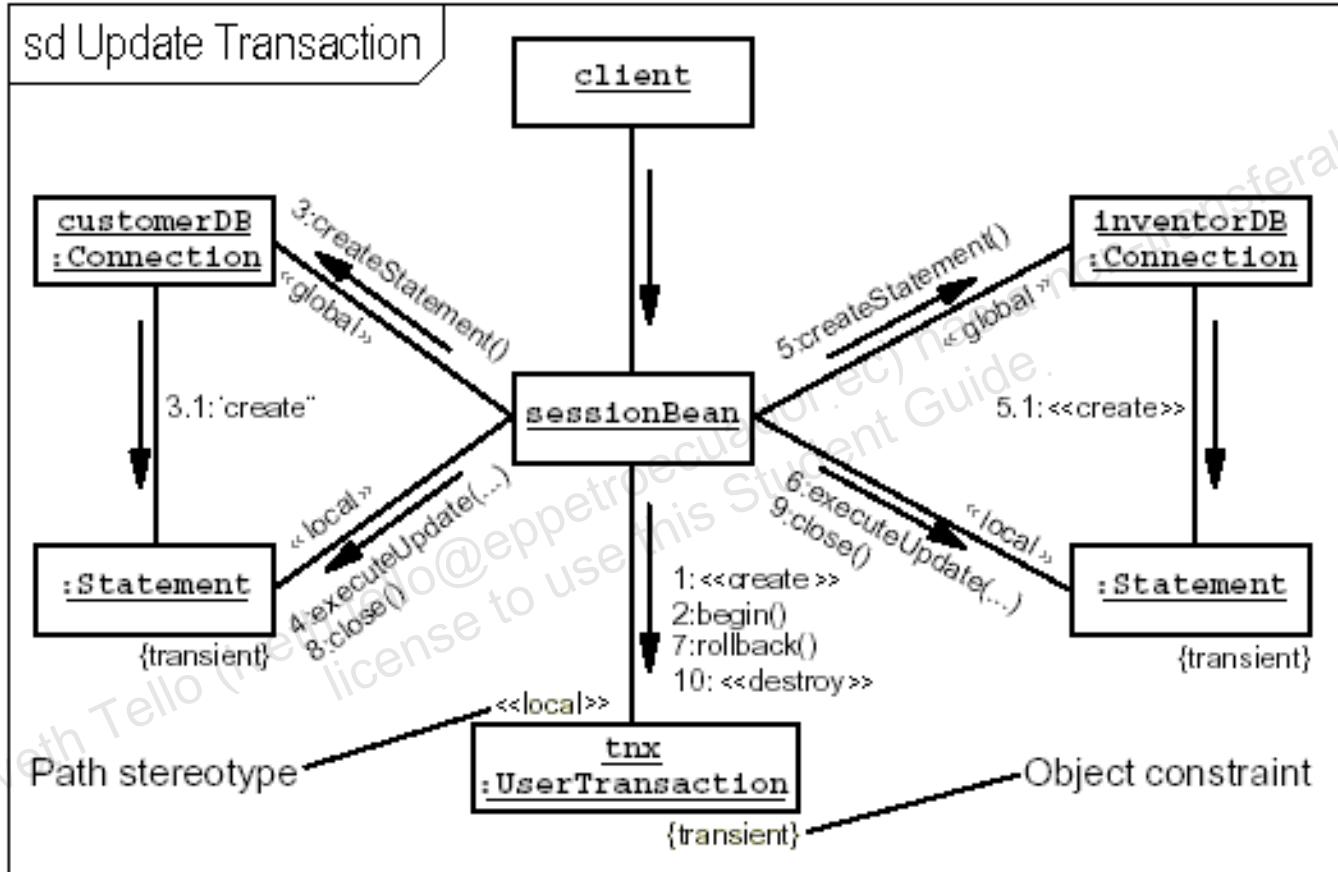
The figure below shows a Communication diagram in which an actor initiates a login sequence within a web application using a servlet.



The servlet in the figure uses an object of the `LoginService` class to perform the lookup of the username, verify the password, and create the `User` object. The links between objects show the dependencies and communications between these objects. Messages between objects are shown by the messages on the links. Messages are indicated by an arrow in the direction of the message, and a text string declares the type of message.

The text of this message string is unrestricted. Messages are also labeled with a sequence number that indicates the order of the message calls.

The figure below shows a more elaborate Communication diagram. In this diagram, a client object initiates an action on a session bean. This session bean then performs two



You can label the links with a stereotype to indicate if the object is global or local to the call sequence. In this example, the connection objects are global, and the statement and transaction objects are local. You can also label objects with a constraint to indicate if the object is transient.

Concurrency

You can show that two or more messages are sent concurrently in Communication diagrams by placing a letter after the sequence number. For example, the numbering of these two messages indicates that they are sent in parallel:

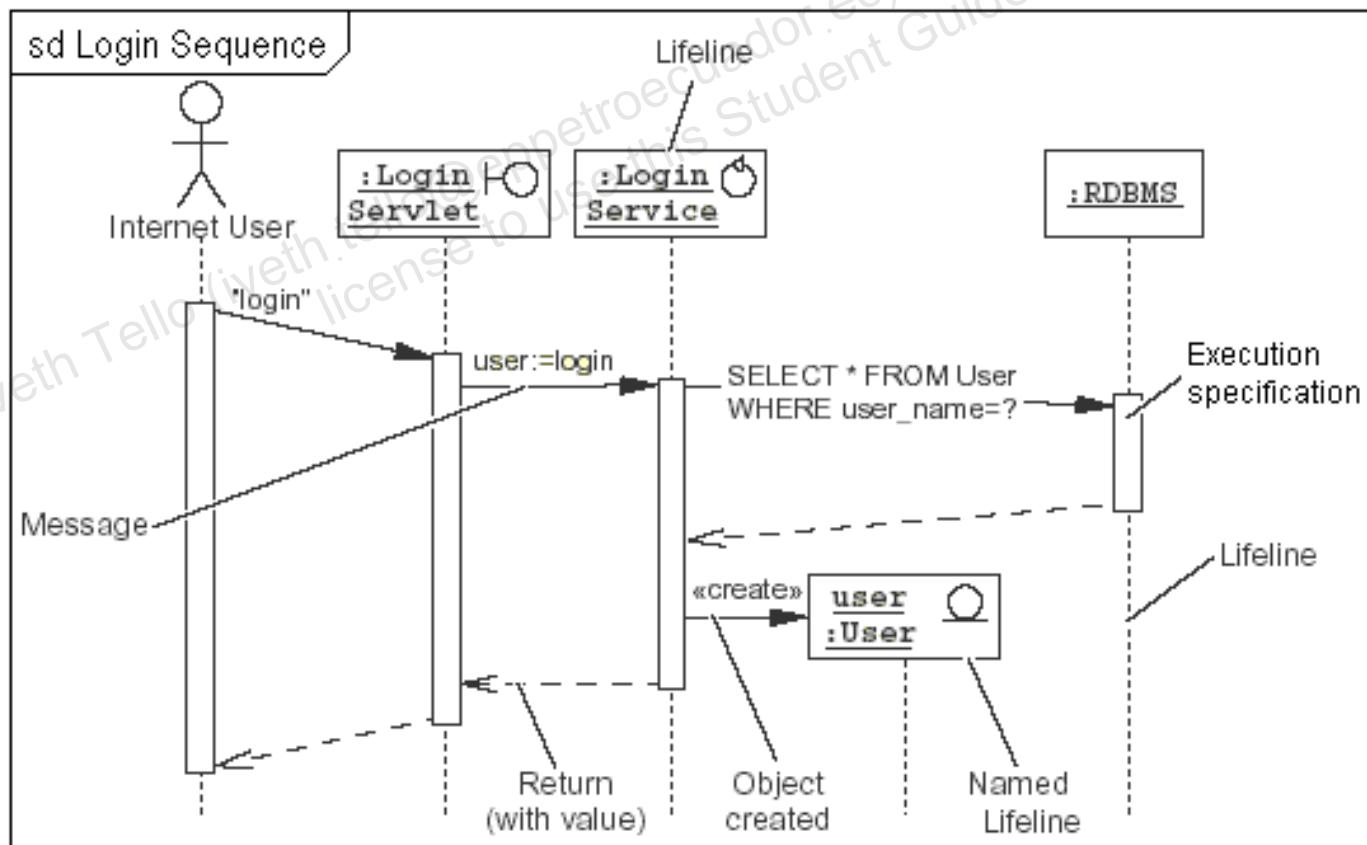
- 3a:alertUser()
- 3b:createErrorLog()

Sequence Diagram

A Sequence diagram represents a time sequence of messages exchanged between several objects to accomplish a particular task. A Sequence diagram helps you to understand the flow of messages and events that occur in a certain process, typically a use case. A Sequence diagram contains the same information as a Communication diagram of the same process, but in a time-sequenced format. Two diagrams that show the same information in different formats are known as isomorphs of each other.

The figure below shows a Sequence diagram in which an actor initiates a login sequence within a web application which uses a servlet. Time in a Sequence diagram is indicated by vertical position. Items that appear lower in the diagram happen after those higher up. The lifelines are the icons at the top of the diagram, formerly known as roles, that can show a class name, followed by a colon followed by an instance name that describes the required role. If the class name is omitted, the lifeline is considered anonymous.

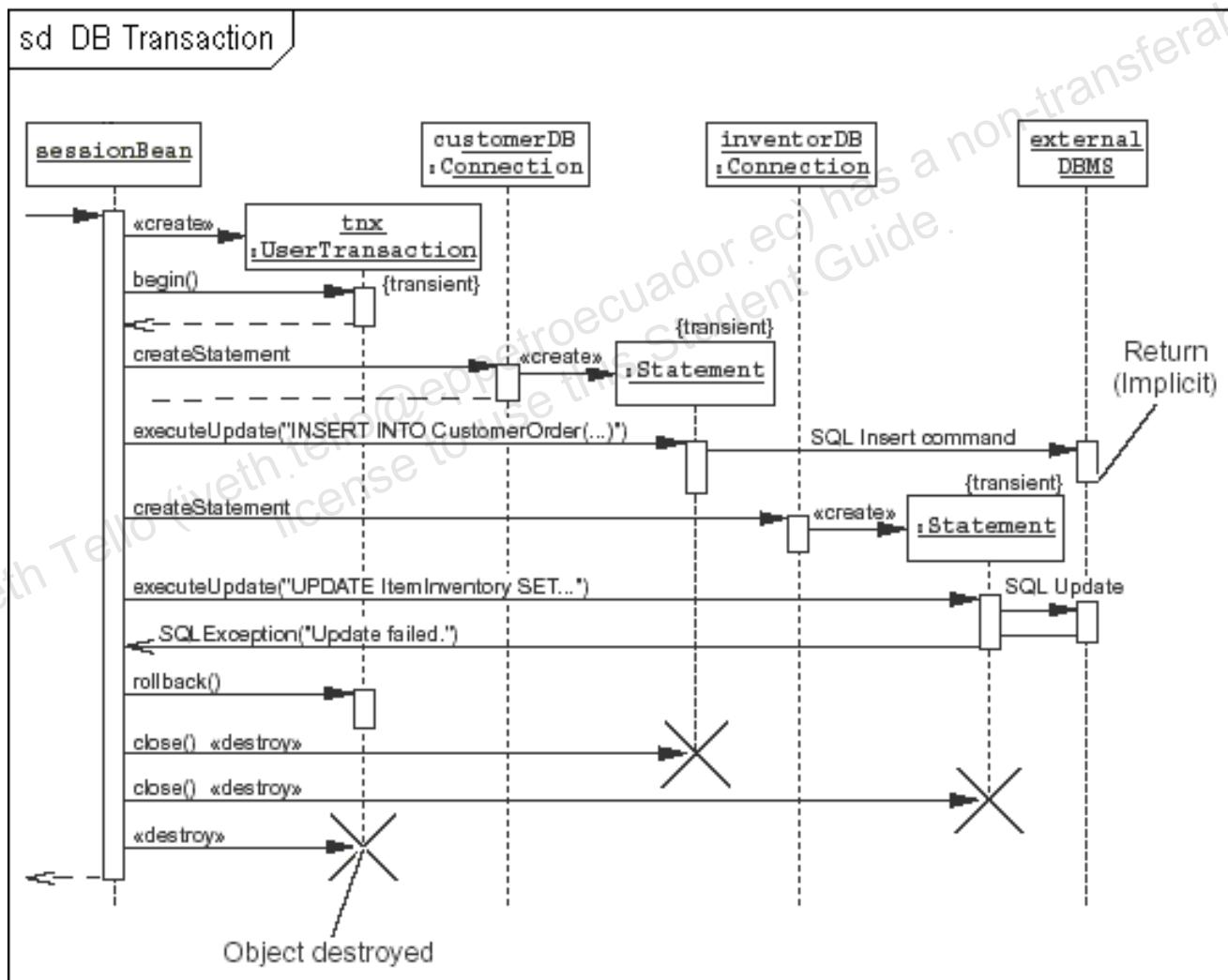
Prior to UML 2, it was only the vertical line that drops down below the rectangles at the top of the Sequence diagram that was known as the lifeline. The line was dashed in UML 1.x, but as of UML 2 it can be solid or dashed.



The figure above shows a message as an arrow between the servlet and the service object. The arrow is perfectly horizontal, which indicates that the message is probably implemented by the local method call. The message arrow between the actor and the servlet is angled, which indicates that the message is sent between components on different machines, such as an HTTP message from the user's web browser to the web container that handles the login servlet.

The wider areas on the lifelines labeled execution specification (previously known as an activation bar or an activation box) show the period during which a method is active. You can think of the method as being on the call stack until its execution specification bar has ended.

The figure below shows a more elaborate Sequence diagram.



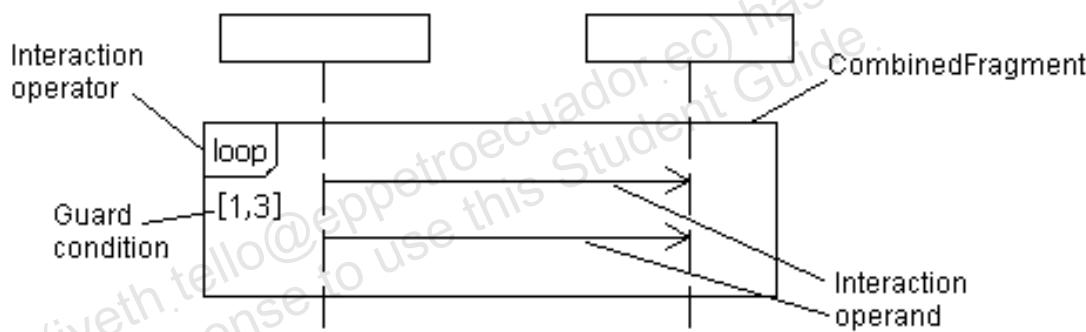
The figure above shows a few more details about Sequence diagrams. First, the return arrow is not always important. A return arrow is implicit at the end of the activation box.

Sequence diagrams can show the creation and destruction of objects explicitly. Lifelines at the top of the diagram existed before the leftmost entry message. Lifelines that have a message arrow pointing to the head of the node with the “create” message are created during the execution of the sequence. The destruction of an object is shown with a large cross that terminates the lifeline.

Sequence diagrams can also show asynchronous messages. This type of message uses a solid line with stick arrow head instead of a filled arrowhead.

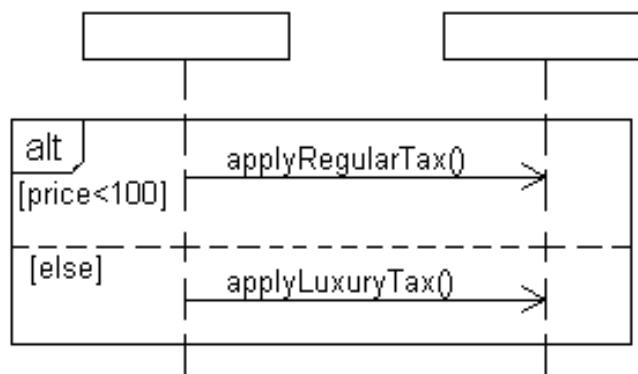
CombinedFragments

Loops, conditionals, and certain other constructs can be shown in Sequence diagrams as CombinedFragments using frames. A frame is a rectangle which encloses one or more messages in a Sequence diagram. In the upper left corner of the rectangle is a label, which shows the interaction operator for that CombinedFragment. It shows the notation used for a CombinedFragment.



The CombinedFragment in the figure above shows that the two messages that are the interaction operands are repeated exactly three times.

The figure below shows how a conditional is expressed in CombinedFragment notation:



Some Common CombinedFragment Operators

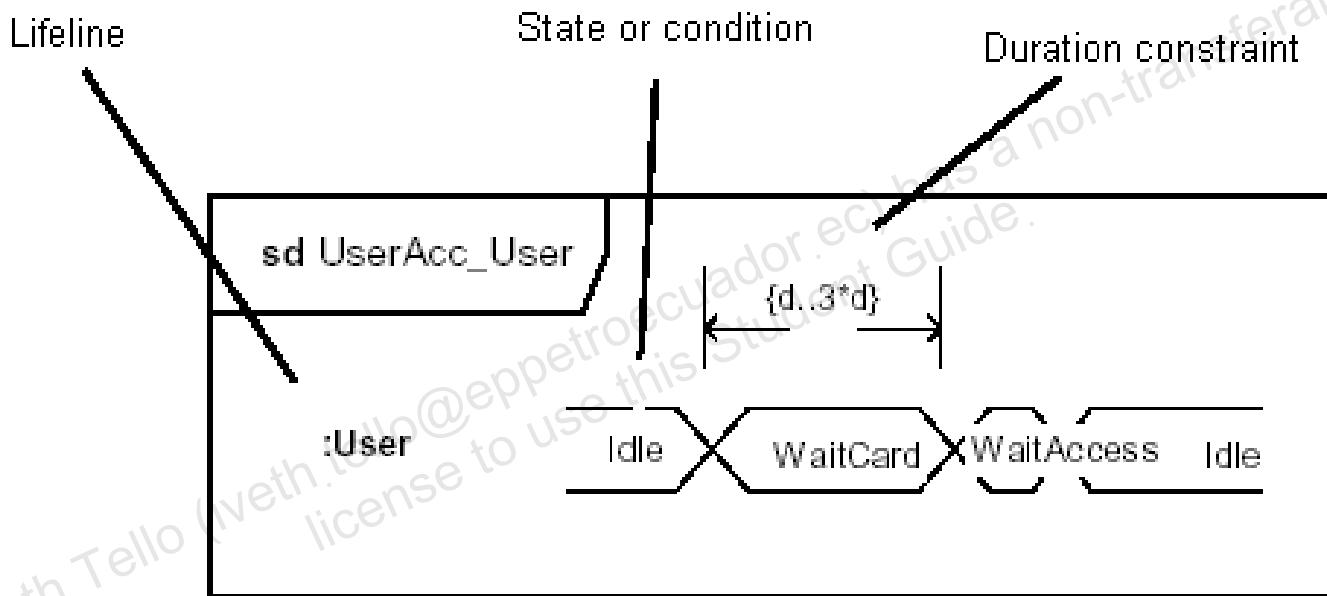
Operator	Meaning
alt	Alternatives. Two or more alternatives are provided, and only one can be chosen, based on a guard condition.
opt	Option. One alternative is provided, with a guard condition. If the guard condition does not hold, nothing is done.
break	Break. If the guard is true, the rest of the content of the CombinedFragment is ignored.
par	Parallel. The operations inside the CombinedFragment may be performed in parallel.
loop	Loop. The operation(s) in the CombinedFragment are repeated as specified by a guard condition.
critical	Critical. Only one thread can run in this CombinedFragment.
assert	Assert. Only if the guard condition of the assert is true can execution continue.
strict	Strict. Operands within the fragment maintain sequencing according to their vertical time line positions.
seq	Weak sequence. Operands within the fragment from different lifelines may be performed in any order. Operations from the same lifeline maintain sequencing.

Timing Diagram

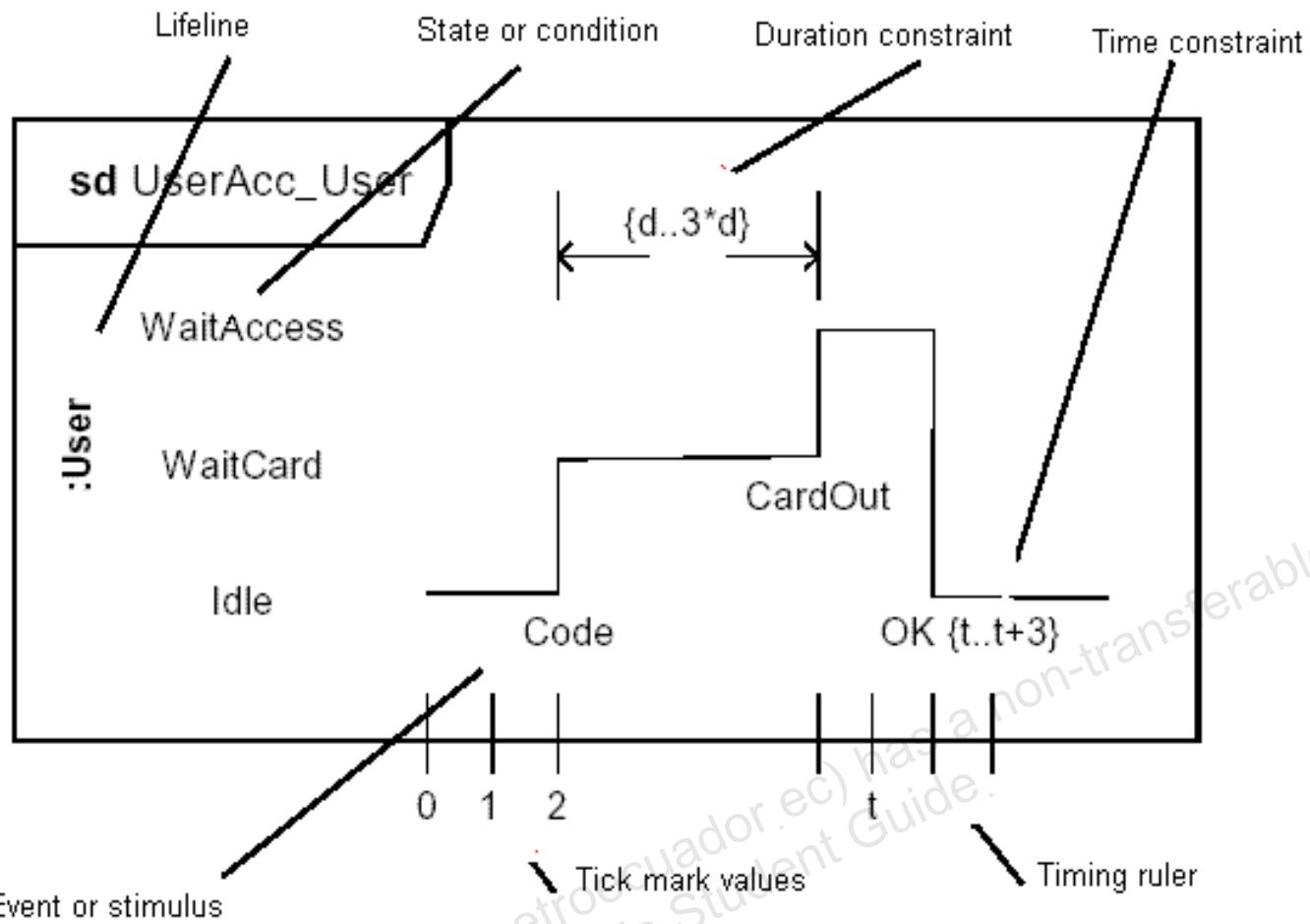
This is a new type of Interaction diagram used to indicate timing and duration constraints. Of course, any behavioral diagram can indicate this, but Timing diagrams are specifically intended for this purpose, and are used primarily in real-time applications where they can model timing, concurrency and synchronization, and fault tolerance.

Timing diagrams are similar to State Machine diagrams, but state machines deal with transitions based on events, while Timing diagrams deal with transitions based on time periods.

There are two kinds of Timing diagrams. A concise value Timing diagram taken from the UML 2.1.1 Superstructure Specification, page 518, modeling the lifeline for a user requesting access to some system such as an ATM via a card, is shown below:



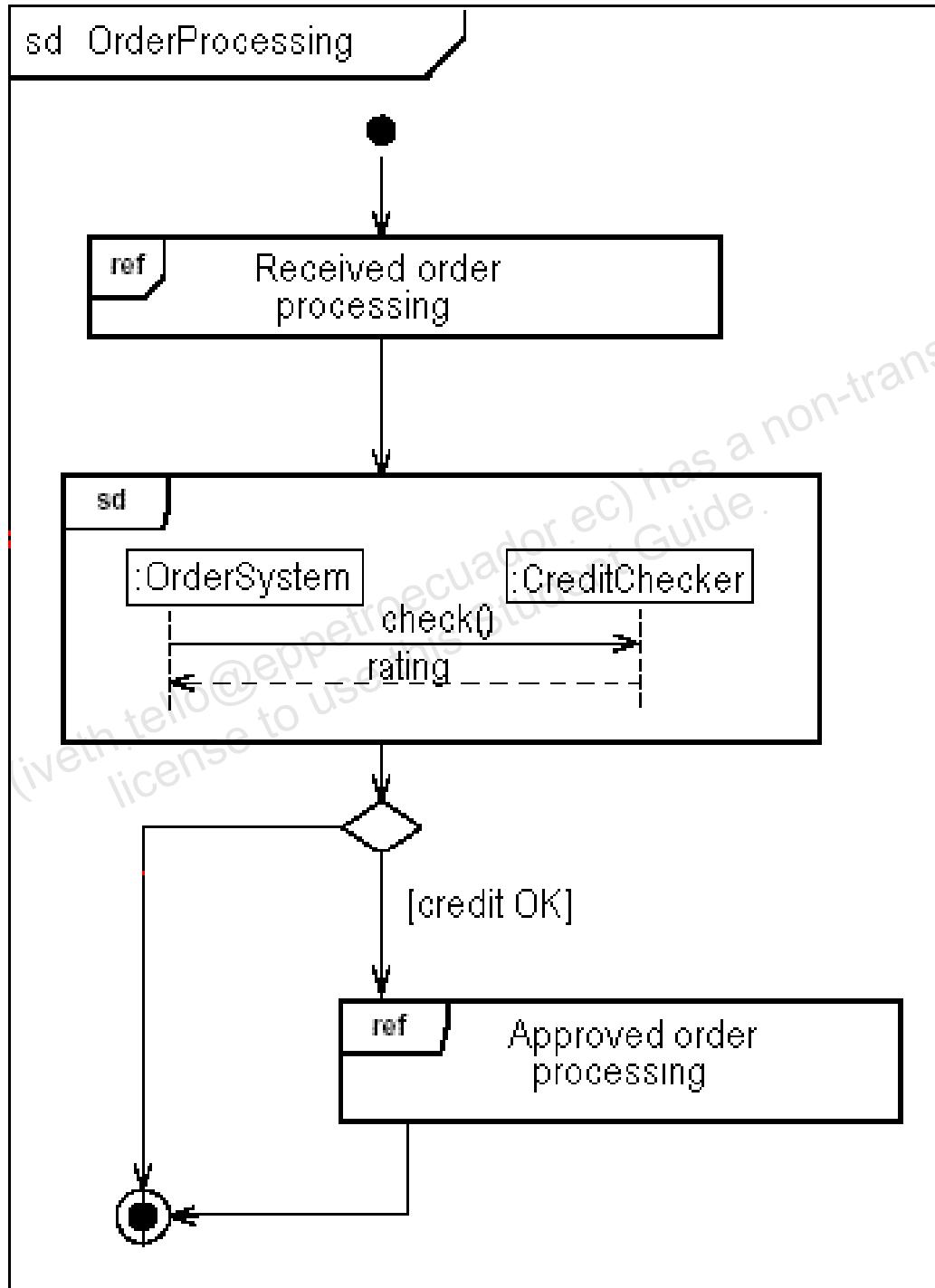
A more robust Timing diagram for the same activity, taken from the UML 2.1.1 Superstructure Specification, page 517, is shown below:



Interaction Overview Diagram

This diagram is new in UML 2, and is a kind of Activity diagram that uses interaction nodes instead of activity nodes. This diagram is appropriate for large systems where it can provide an overview of the flow of control flow in a somewhat simplified format.

Embedded in the Interaction Overview diagram are interactions, which can be fully detailed Sequence, Communication, or Timing diagrams identified by labels. In addition, the Interaction Overview diagram can contain interaction occurrence frames, identified by a label of ref, which indicate an activity or an operation that should be invoked at this point, and which is fully described in another Interaction diagram. Thus the Interaction Overview diagram provides a high-level view of interactions that is easier to comprehend because it eliminates detail. An Interaction Overview diagram example is provided below. In it, two other Interaction diagrams are indicated by reference, meaning that they are documented elsewhere, while one is drawn in-line.



Appendix B

Acronyms

Iveth Tello (iveth.tello@eppetroecuador.ec) has a non-transferable
license to use this Student Guide.

Numerics

2PC two-phase commit

A

ACID atomicity, consistency, isolation, durability (the four characteristics of transactions)

API application programming interface

B

B2B business to business

BMT bean-managed transaction

C

CGI Common Gateway Interface

CICS Customer Information Control System

CMM Capability Maturity Models

CMP container-managed persistence

CMR container-managed relationships

CMT container-managed transaction

COM Common Object Model

CORBA Common Object Request Broker Architecture

COTS commercial, off-the-shelf

CPU central processing unit

CRC Class-Responsibility-Collaboration

D

DAO	data access object
DBMS	database management system
DD	deployment descriptor
DTP	distributed transaction processing

E

e-Commerce	electronic commerce
EIS	enterprise information system
EJB	Enterprise JavaBeans
ERP	enterprise resource planning

F

FR	functional requirement
FTP	File Transfer Protocol

G

GUI	graphical user interface
------------	--------------------------

H

HP	hardware platform
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
HTTPS	HTTP over Secure Socket Layer (SSL) protocol
HW	hardware

I

iAS	internet Authentication Service
IDL	Interface Definition Language

IIOP	Internet Inter-ORB Protocol
IMAP	Internet Message Access Protocol
I/O	input/output
IP	Internet Protocol
IT	information technology

J

Java EE	Java Platform, Enterprise Edition
J2SE	Java 2 Platform, Standard Edition
JAAS	Java Authentication and Authorization Service
JAF	JavaBeans Activation Framework
JAR	Java Archive

JavaBeans

components components based on the JavaBeans component architecture

JAXP	Java API for XMP Processing
JAX-RPC	Java API for XML-based RPC
JDBC	Java DataBase Connectivity
JDMK	Java Dynamic Management Kit
JDO	Java Data Object
JMS	Java Message Service
JMX	Java Management extensions
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JRE	Java runtime environment
JRMP	Java Remote Method Protocol
JSP	JavaServer Pages
JTA	Java transaction API
JVM	Java Virtual Machine

K

Kbytes	kilobytes
---------------	-----------

L

LDAP	Lightweight Directory Access Protocol
LP	lower platform

M

Mbytes	megabytes
MDB	message-driven bean
MOM	message-oriented middleware
MVC	
pattern	model-view-controller pattern
MVS	Multiple Virtual Storage, a common operating system on IBM mainframes

N

NFR	nonfunctional requirements
NIS	Network Information System
NPV	net present value

O

OLAP	online analytical processing
OLTP	online transaction processing
OMG	Object Management Group
OO	object orientation or object-oriented
OOAD	object-oriented analysis and design
ORB	Object Request Broker
OS	operating system
OSI	Open System Interconnection

P

PAC	
pattern	Presentation-Abstraction-Control pattern

Q

QoS	quality of service
------------	--------------------

R

RAR resource (adapter) archive

RAS

Requirements reliability, availability, and serviceability requirements

RDBMS relational database management system

RM resource manager

RMI Remote Method Invocation

ROI return on investment

RPC remote procedure call

RUP Rational Unified Process

S

SEI Software Engineering Institute

SMTP Simple Mail Transfer Protocol

SNMP Simple Network Management Protocol

Solaris

OE Solaris Operating Environment

SPI service provider interface

SQL structured query language

SSL secure socket layer

SRS System Requirements Specification

Sun ONE Sun Open Net Environment

T

TCO total cost of ownership

TCP Transmission Control Protocol

TO transfer object

TPC Transaction Processing Performance Council

U

UDP User Datagram Protocol

UI	user interface
UID	user identification (number)
UML	Unified Modeling Language
UP	Unified (Software Development) Process
URL	Uniform Resource Locator
V	
VM	virtual machine
VO	value object
VP	virtual platform
W	
WAR	web archive
X	
XA	extended architecture
XML	Extensible Markup Language

Appendix C

Glossary

Iveth Tello (iveth.tello@eppetroecuador.ec) has a non-transferable
license to use this Student Guide.

A

abstraction

A generalization for the main characteristics of an entity as relevant to a particular discussion, problem, or point of view, rather than the details.

accounting

A mechanism that keeps track of actions that are performed, see: *logging*.

activity diagram

A diagram showing the activities performed within a class. These diagrams are similar to state diagrams, but describe the behavior of a class in response to internal processing rather than external events.

actor

A class of people or systems that interact with a system. Use cases describe interactions between the actors and the system. See use case.

application

Any specific use of the computer. The term is often used synonymously with software program.

application programmer's interface (API)

The interface to a library or package of language-specific functions or methods.

architecturally significant use case

A use case that contributes significant, or greater than average, risk to the project.

architecture

The integration of software and hardware components, with usage, functionality, performance, reuse, and aesthetic concerns, to form a system.

asynchronous communication

The transmission of data at intervals which are neither fixed nor predetermined. That is, data can be sent between parties intermittently rather than as part of a defined response, steady stream of information.

auditing

A mechanism for inspecting accounting logs to determine if something improper has happened.

authentication

A means for the system to verify that the identity of the principals performing an operation are who or what they claim they are. Authentication is typically achieved through the use of a password or digital signature mechanism.

authorization

A means for matching principals with the rights to perform certain operations. File systems typically have permissions, such as, “The owner of this file can read, write, or delete it, but other users can only read it.”

B

bandwidth

A term that describes the data capacity of a network connection.

Bean

A reusable software component written in the Java programming language, typically with properties and methods. Also known as a JavaBeans component.

boundary

The area where two objects interact.

brittleness

The degree to which small changes will break large portions of a system. Brittleness is the inverse of flexibility.

browser

A program, used to view World Wide Web materials, that is capable of interpreting URLs and understanding different Internet protocols.

C

class diagram

A diagram that shows a set of classes, interfaces, collaborations, and their relationships. Class diagrams address the static view of a system.

client

A software program, process, or system that requests information or services from another software application, process, or system, such as a server. For example, a browser is a client that accesses data from HTTP servers. A client is part of a client-server architecture.

cohesion

A measure of how parts of a class or component are logically related to each other and the overall whole. Higher cohesion indicates that the features within a class help to realize its intent.

commercial off the shelf software (COTS)

Software that is available through retail channels.

Common Object Request Broker Architecture (CORBA)

The architecture and specifications aimed at software developers and designers who want to produce applications that comply with OMG standards for the ORB. CORBA enables communication between objects written in the Java programming language and objects written in another language.

concurrency

A state of a system where multiple actions are performed simultaneously. Concurrent data access is when more than one distinct transaction accesses or modifies a particular piece of data at the same time.

container

The interface between a Java EE component and the Java EE server. Containers provide services to the component and manage the component's lifecycle.

coupling

A dependency between classes or components. It is usually desirable for a class to be associated with just enough other classes to allow it to realize its responsibilities. This is called low coupling.

D

decryption

The process of turning ciphertext back into plaintext. This process usually requires a code or a key. A key is a sequence of characters that are used to encode and decode a file. See encryption.

distributed application

A program that makes calls to other address spaces, possibly on another physical machine.

distributed computing

The technique of allowing applications that run on one machine to access applications that are running on another machine. That is, client programs make calls to programs in other address spaces, either local or remote.

distributed object computing

An extension of distributed computing, where objects are implemented in an address space separate from the client program.

domain

The business area in which the application is to be used. Practices, concepts, terminology, rules, and so on, are taken from the domain for use in the application, and the application must conform to standards defined and used in the domain.

domain model

A model of a domain that is independent of the implementation mechanisms used in systems supporting that domain.

E

encapsulation

The process of hiding implementation details behind well-defined interfaces. For example, ensuring that internal data structures cannot be accessed directly.

encryption

The process of scrambling files or programs, changing one character string, byte, or bit to another through a mathematical algorithm (encipher). A message that is not enciphered is referred to as plaintext or cleartext and an enciphered message is called ciphertext. See decryption.

enduring business themes

Themes that represent the slowly changing or nonchanging aspects of the business domain or problem domain. When you focus on these abstract themes, you can promote the development of object-oriented frameworks that enhance reusability.

evolutionary prototype

A prototype of the system that demonstrates typical use cases and conformance to nonfunctional requirements.

F

firewall

A machine that connects two or more networks, and runs filtering and logging software that restricts and monitors traffic passing from one network to another. All traffic between two networks must pass through a firewall if the protection is to be effective. Firewalls enable the implementation of network and application security policies.

forward caching

A data management technique that copies data from its source into the application space in an attempt to improve performance.

friction

The amount of interaction that occurs between two components. Friction is measured based on how a change in one component affects both components. Friction is closely related to coupling.

FTP

File Transfer Protocol. FTP is an Internet client-server protocol for transferring files between computers.

functional requirements

The operational aspects of the system that fulfill the use cases. The behaviors that the system must exhibit

G

GUI

Graphical user interface. The term GUI can also refer to the techniques involved in using graphics, along with a keyboard and a mouse, to provide an easy-to-use interface to a program.

H

horizontal service

A service based on the infrastructure of the system and accessed at many layers.

Horizontal services are items, such as security, that are carried throughout the application and that touch the application at many levels.

HTTP

Hypertext Transfer Protocol. HTTP is the most common protocol that is used to transfer hypertext documents on the World Wide Web.

I

identification

A means for the system to refer to a principal. Identification is typically accomplished with the use of a login name or a user id.

IOP

Internet Inter-ORB Protocol. A protocol used in distributed systems to allow communication between objects written in different programming languages. IOP is a key part of CORBA.

IMAP

Internet Message Access Protocol. Another protocol for accessing email. For example, IMAP can be used to view the header and the sender of an email before electing to download the message.

inception

In the Unified Process and others, the initial project phase focusing on defining scope.

increment

An element of the development process that is focused on completion of a subset of the use cases as a single project.

incremental development

Giving form to a system in multiple smaller steps over time rather than one large step.

Incremental development is characterized by achieving working and tested functionality along the way.

Information Management System (IMS)

A database management system for mainframes from IBM.

Interface Definition Language (IDL)

A set of language constructs that can be used to define the interface between an implementation of the interface and an application that executes operations on the interface. IDL is used in the development of CORBA applications.

Internet

The worldwide network of computers communicating using the TCP/IP protocols.

iteration

A distinct set of activities with a baseline plan and evaluation criteria that results in the completion and release of an increment. See increment.

J

Java

An object-oriented programming language developed by Sun Microsystems to solve a number of problems in modern programming practice.

Java IDL API

A set of classes and interfaces that enables developers to define a set of remote interfaces using the CORBA IDL standard. Java IDL API maps IDL constructs onto a set of stubs and skeletons that can be used to create a CORBA implementation.

JDBC API

Java Database Connectivity application programming interface. A set of interfaces designed to insulate a database application developer from a specific database vendor.

JMS

Java Message Service. A standard set of Java technology interfaces that are implemented by vendors to build Java-based messaging services.

JNDI

Java Naming and Directory Interface. JNDI allows a Java application to access naming and directory services. In a distributed application, components might use JNDI to locate each other. JNDI can be used for looking up Enterprise JavaBeans (EJB) technology components, components accessed with RMI, and CORBA components, among others.

L

latency space

A graphical representation of the communication latency compared to the number of round trip messages.

latency

A term that describes the amount of time it takes for a minimal-size packet to make a round trip.

layering

A hierarchy of separation.

legacy

A term used to describe almost any preexisting system that must be integrated with a new development.

level of abstraction

Refers to the place in a hierarchy of abstractions that range from high levels (very abstract) to low levels (very concrete).

logging

A mechanism that keeps track of the actions that are performed, who did them, when, and so on. If things happen that should not, then logging records can be used to investigate how these improper actions occurred, and perhaps to correct the damage.

M

messaging system

Allows objects to communicate remotely without having any knowledge of each other. A message is published to a destination object and any object that is interested in messages of a particular topic will register themselves and their interest with the destination. The destination then delivers messages appropriately.

MIME

Multi-purpose Internet Mail Extensions allows exchange of many different data types on the Internet, such as audio, video, text, and applications. A server will include the MIME information in the header of a file transfer and the client will then use the MIME to select appropriate software.

multicast

A special form of broadcast where copies of the packet are only delivered to a subset of all possible destinations.

N

naming service

A server application that enables clients on a network to look up information or objects providing services. The server puts the remote object into the naming service and the client later retrieves it. The information stored in a naming service takes the form of name and value pairs.

network

A group of connected computers.

NFS

A distributed application that enables remote file systems to be accessed by the end-user in the same way that a user would access a local file system.

NIS

Network Information System. Developed by Sun Microsystems as a network naming and administration system.

nonfunctional requirements

Project requirements that include service-level requirements (systemic qualities), implementation constraints, and business rules (domain model).

O

object aliasing

The effect of having multiple references or objects representing the same data.

object-based patterns

Patterns that do not require the transfer of behavior with arguments and return values. Object-based patterns can be distributed over a network.

Object Management Group (OMG)

A nonprofit international consortium that is dedicated to promoting the theory and practice of object technology for the development of distributed computing systems.

Object Request Broker (ORB)

A program that provides the communications infrastructure that enables objects to transparently make and receive requests and responses in a distributed environment.

P

pass by reference

The passing of a parameter in a method call whose value enables the method to access the original value of the variable. Changes made to the parameter value affect the value in the caller method.

pass by value

The passing of a parameter in a method call by copying the value of the named variable.

Pattern

An expression of a possible solution to a common problem in a given context.

POJO

Plain Old Java Object

port

A connection place between a client and a server application running on a network. A port can either be well known, such as the number for that application is pre-assigned, or it can be dynamically assigned when the server application starts up. Port numbers are from 0 to 65536. Ports 0 to 1024 are reserved for use by certain privileged services. For the HTTP service, port 80 is defined as a default and it does not have to be specified in the URL.

principal

A person or process that is known to the system and that might request certain functionality.

protocol

An agreed convention for intercomputer communication.

proxy

Something that stands in between two actors. For example, if you use a proxy server, then your requests will be sent to the proxy, which will then send requests to the Internet. The proxy stands between you and the Internet.

R

Rational Unified Process (RUP) methodology

A software engineering process that extends UP. A proprietary form of the Unified Process available from IBM, Inc.

refactoring

A process of modifying an existing code structure without modifying its current API. Refactoring uses the object-oriented concept of encapsulation to make changes to a software implementation.

reification

The process of capturing an abstraction as an object to fulfill a specific design need.

Remote Procedure Call (RPC)

A paradigm for implementing the client-server model of distributed computing. A request is sent to a remote system to execute a designated procedure (through the use of supplied arguments) and to return the result to the caller.

requirement

A specification of what a system must do or conform to be successful.

resource access control

A mechanism for enforcing the rules documented in the authorization system. To be effective, resource access control must be based upon functionality in the operating system.

REST

Representational State Transfer (REST) style of software architecture. An important concept in REST is the existence of resources, each of which can be referred to with a global identifier, that is, a URI. In particular, data and functionality are considered resources that can be identified and accessed through URIs.

RMI API

Remote Method Invocation application programming interface. A set of classes and interfaces designed to enable developers to make calls to remote objects that exist in the runtime of a different virtual machine invocation.

robustness analysis

The process of analyzing use case scenarios, identifying a set of objects that will participate in each use case, and classifying these objects into three types: boundary objects, entity objects, and control objects. Robustness analysis is a simple, but useful, technique for moving from analysis to design and ensuring that your use-case scenarios are complete and realistic.

S

saturation

The point at which a performance curve begins to show a leveling off or downturn of performance and throughput as additional load is added to the system.

scenario

A single outcome of a use-case execution that is based on a sequence of known decisions.

server

An application that runs on a machine that is called upon for services by other applications, which are often running on other machines.

SMTP

Simple Mail Transfer Protocol. SMTP is a TCP/IP protocol that is used to send and receive email. Because SMTP has a limited ability to queue messages, it is used in conjunction with other protocols such as IMAP and POP3.

socket

A software endpoint for network communication. Two programs on different machines each open a socket to communicate over the network. This is the low-level mechanism that supports most networking programs.

SSL

Secure Sockets Layer. SSL is a commonly used protocol that is used to ensure secure communication between client and server. Many major browsers support SSL, as well as most server software.

stakeholders

Anyone with an interest in the outcome of a project.

SOA

Service Oriented Architecture. Facilitates the development of enterprise applications as modular business services that can be easily integrated and reused, creating a truly flexible, adaptable IT infrastructure.

SOAP

SOAP provides a mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment. SOAP is grounding in XML. The SOAP message format is defined by an XML schema, which uses XML namespaces to make SOAP very extensible.

surface area

The place where modules communicate through their interfaces. When you create abstractions for your architecture, you define the modules to interact with other modules in simple, well-defined, and relatively small ways that clearly distinguish their boundaries or surface area.

synchronous communication

The term that is used whenever a client makes a request of a server and then waits for the response.

system boundary

The part of the system solution defined by who or what uses the system.

systemic quality

A quality of, or related to, a system, as opposed to a function provided by the system. Systemic qualities include scalability, performance, availability, reliability, and others.

systems requirement specification (SRS)

A document outlining the requirements for a system solution.

T

TCP

Transmission Control Protocol. A virtual circuit protocol of the Internet protocol family. TCP provides reliable, flow-controlled, in-order, two-way transmission of data in a byte stream.

thread

A task that can execute concurrently with other tasks in the same process.

transaction

A single unit of work performed as a series of individual operations, where the whole series must all succeed or all fail.

transfer object

A container or aggregate object that contains multiple elements. Unlike a normal object, transfer objects might well be unrelated except in the context of the problem at hand.

U

UDP

User Datagram Protocol. A transport protocol in the Internet suite of protocols, which delivers datagrams. Because datagrams are messages that are not acknowledged, the delivery of them is unreliable.

Unified Modeling Language (UML)

A language for visualizing, specifying, constructing, and documenting the artifacts of a system.

Unified Process (UP) methodology

A software engineering process, which consists of four phases including inception, elaboration, construction, and transition.

use case

A description of a set of actions (including variants) performed by a system, which yields an observable result of value to an actor.

V

value object

A container or aggregate object that contains multiple elements. Unlike a normal object, value objects might well be unrelated except in the context of the problem at hand. See also transfer object.

vertical service

A service based on the content of the system that reflects the business model. For example, an inventory service can present some object that represents an item in a warehouse inventory.

W

workflow

The steps used in a waterfall-style development model. Information flows in one direction through the workflows, beginning with a review and update of the requirements, then passing through analysis and design, and finally ending with the testing of an executable system.

Appendix D

UMLet Tips

Iveth Tello (iveth.tello@eppetroecuador.ec) has a non-transferable
license to use this Student Guide.

How to add elements to the diagram?

Double-click any element in the palette; it will appear in the upper left corner of the main diagram window.

How to duplicate elements on the diagram?

Double-click an element to duplicate it. Alternatively, use copy/paste or Ctrl+C/Ctrl+V.

How to select multiple elements?

Hold Ctrl to select multiple elements.

How to lasso-select multiple elements?

Press Ctrl and left-click to select a rectangle containing the desired elements.

How to change UML elements?

Select an element and modify its attributes in the lower right text panel. Each element type has a simple markup language, e.g., the text "/ClassName/" causes "ClassName" to become italic. The markup languages are best explored via the sample UML elements in the palettes.

How to enter comments in a UML element description?

UMLet supports C++-style comments---starting a line with "//", e.g., "//my comment..", will let UMLet ignore that markup line.

How to change the color of UML elements?

Right-click an element and select its background or foreground color via the context menu.

Alternatively, just type the name of the color in the element description, e.g., "bg=black", or "fg=red".

How to create UML relations?

Double-click a relation and then drag its end points to the borders of UML elements; they will stick there.

How to edit the relations?

Many UML tools make it time-consuming to change the type or the direction of a relation. In UMLet, simply modify the linetype, i.e., by changing the line "lt=" in the element description. For example, change "lt=<." to "lt=->>" to change the direction, the arrow type, and the line's dots at the same time.

How to label relations?

Edit the name of a relation in the relation's description.

Role names can be specified using "r1=" or "r2=".

For multiplicities use "m1=" or "m2=".

Qualifiers are done with "q1=" or "q2=".

How to create sequence diagrams?

Change the current palette to "Sequence - all in one". Add the sequence diagram element to the diagram via double-click.

This element's markup language is slightly more complex. The main idea is that each lane has a name and an ID (defined by the string "_name~ID_"). The IDs can then be used to define messages between lanes, e.g., "id1->id3".

How to create activity diagrams?

Change the current palette to "Activity - all in one". Add the activity diagram element to the diagram via double-click.

Here, TABs in the element description are used to define the activity forks.