

Architect Enterprise Applications with Java EE

Volume 1 – Student Guide

D68136GC10
Edition 1.0
July 2012
D71725

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.

1

Introducing Enterprise Architecture

ORACLE®

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

Lesson Objectives

After completing this lesson, you should be able to do the following:

- Describe the goals of the course
- Define enterprise and enterprise application architecture
- List the challenges of enterprise applications
- Define software architecture
- Describe the need for software architecture
- Define an architect's roles and responsibilities



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

Lesson Agenda

- Course objectives, agenda, and appendices used in the course
- What is enterprise architecture
- An architect's roles and responsibilities



ORACLE

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

Course Objectives

After completing this course, you should be able to:

- Analyze, evaluate, and prescribe architectural strategies to solve enterprise and application architectural challenges
- Use Java EE component technologies to solve typical problems in system architecture
- Derive software systems using solutions defined in Java EE and enterprise architecture integration (EAI) patterns
- Address quality-of-service requirements in a cost-effective manner using engineering trade-off techniques
- Describe the role of the architect and the products an architect delivers



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

Course Objectives

- Use UML to model architectural components
- List and describe typical problems associated with large-scale enterprise systems
- Identify risks associated with architectural choices and consider strategies for mitigating those risk
- Consider and discuss the pros and cons, and trade-offs of various architectural choices and strategies
- Consider and discuss business and technical factors and constraints on architectural choices and strategies



ORACLE

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

Target Audience

This course is designed for:

- Java Enterprise Edition developers
- Those looking to become architects



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

If you are concerned whether your background and experience matches the target audience, ask the instructor.

Introductions

Please introduce yourself. Tell us about:

- Your name, company, and role
- Your experience with Java Enterprise Edition
- Your experience with design
- Your experience with architecture (if any)
- Any specific expectations you have for this class



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

Course Agenda

- Day 1:
 - Introducing Enterprise Architecture
 - Fundamental architectural concepts
 - Developing a security architecture
- Day 2:
 - Understanding nonfunctional requirements
 - Defining common problems and solutions: risk factors and system flexibility



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

The class schedule might vary depending upon the pace of the class. The schedule shown is an estimate. The instructor may provide updates.

Course Agenda

- Day 3:
 - Defining common problems and solutions: networks, transactions, and capacity planning
 - Java EE 6 New Features
- Day 4:
 - Developing an architecture for the Client tier
 - Developing an architecture for the Web tier
 - Developing an architecture for the Business tier
- Day 5:
 - Developing an architecture for the Integration and Resource tiers
 - Evaluating the software architecture

 ORACLE

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

Classroom Guidelines

- Be on time for each session. The instructor will start promptly.
- Please ask questions, but be respectful of the topic at hand and the interest of other students.
- Ensure cell phones and pagers are silent.



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

We hope that these guidelines will help the class proceed smoothly and enable you to get the maximum benefit from the course.

Appendices Used in the Course

- **Appendix A: UML 2.0 Quick Reference**
 - Abbreviated reference to UML 2.0 notation
- **Appendix B: Acronyms**
 - List of frequently used acronyms
- **Appendix C: Glossary**
 - Glossary of terms used in the course
- **Appendix D: UMLet Tips**
 - How to use UMLet for capturing UML diagrams for class practices

 ORACLE

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

For More Information

| Topic | Website |
|-----------------------|---|
| Education & Training | http://education.oracle.com |
| Product Documentation | http://www.oracle.com/technology/documentation |
| Product Downloads | http://www.oracle.com/technology/software |
| Product Articles | http://www.oracle.com/technology/pub/articles |
| Product Support | http://www.oracle.com/support |
| Product Forums | http://forums.oracle.com |
| Product Tutorials | http://www.oracle.com/technology/obe |
| Sample Code | http://www.oracle.com/technology/sample_code |



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

This course and the instructor will attempt to address any questions that you might have, but after you complete the course Oracle provides a variety of resources you may use to obtain additional information.

Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- What Is enterprise architecture
- An architect's roles and responsibilities

ORACLE

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

Discussion Questions

- Why does architecture get such close scrutiny today, given that it did not figure into software projects in the past?
- How might you minimize or control project risks that are associated with distributed systems?



ORACLE

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

Challenges of Enterprise Applications

- Increasing development productivity
- Responding to increasing demand
- Sustaining the value of existing information systems
- Maintaining system security



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

As more and more organizations rely on information technology to reach their customers and partners, the development of enterprise applications is facing more and more challenges, for example:

- **Increasing development productivity:** With a variety of technologies and products available, it is difficult to build a consistent knowledge base as a blueprint to guide the development practice.
- **Responding to increasing demand:** The information system needs to maintain high performance, high availability, and it also needs to scale easily to handle the increasing load.
- **Sustaining the value of existing information systems:** Most organizations rely on multiple existing systems to deliver their business value. It is difficult to integrate these systems together to maximize the value of the assets. In addition, it is increasingly difficult to maintain existing systems as changes and fixes are introduced into the system.
- **Maintaining system security:** Most enterprise applications must run in a secure distributed environment. It is difficult to develop an information system that operates at a high security level while being efficient and unobtrusive.

What Is Software Architecture?

Software architecture describes the structures of the system, including:

- Software elements
 - Elements are captured as abstractions
 - Correspond to high level system modules or components
- External visible properties of elements
 - Describe element features exposed to others
 - Typically represent services provided to other elements
- Relationships of elements
 - Describe how elements interact with others



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

The elements are captured in the software architecture as abstractions, with a focus on their visible behavior and their interaction with other elements. Elements typically appear in the architecture in the form of high level software components or system modules. The external visible properties of an element describe the features an element exposes to others, such as provided services, performance information and security implications, and so on. These external visible properties are typically described in the form of component interfaces.

The architecture of a software system typically requires multiple structures from different perspectives (also referred to as views) to capture the elements and properties they possess. Traditional object-oriented design focuses on a static view of the system to identify software components and their interface-level specifications. This view is essential to describe the functionality of the software system, and how the functionality is implemented. However, for large-scale enterprise systems, there are typically a set of associated quality-related requirements. To address these quality-related properties of the system, additional views are required to describe the deployment of software components, their security context, and so on.

From a practical perspective, the architecture can be considered as an abstract blueprint that describes essential principles and decisions about the software system, including its essential components and their interactions. The goal of creating the software architecture in a software development project is to facilitate the implementation of the software product, and ensure that the software product meets its requirements.

Justifying the Need for Software Architecture

- For years, software has been developed without a special role called *architect*.
- Why is it that software engineering groups are now employing people in this role?
 - Crucial changes in software engineering have increased the importance of architecture:
 - Scale
 - Security
 - Distribution: minimal versus highly distributed
- As systems move from a single host to large-scale, distributed enterprise systems, the risks and difficulty of meeting project requirements increases.



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

For years, software has been developed without creating an architecture and without a special role called the architect who designs the structure for the final product. However, several changes have occurred in software engineering that have caused software engineering groups to seek people who can fill the architect role to mitigate the associated risks.

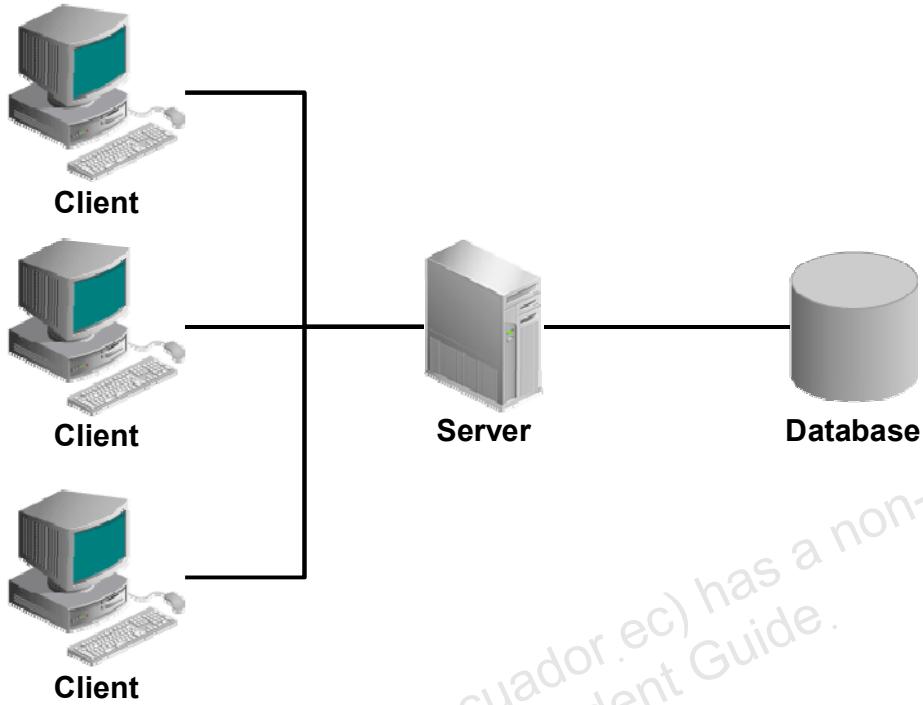
The most obvious change is one of scale. The scale indicates the amount of resources necessary to implement and support the system. Scale has been increasing since the first computer program was written. Most enterprise applications now utilize multiple hardware devices and software products to deliver business values to an increasing number of users. Furthermore, the deployment of the enterprise application needs to accommodate more users and deliver more services as the business grows while maintaining the qualities.

Another important change is security. Security used to be treated as an after-thought, and little action was taken until breaches had occurred. Due to the high cost of recovering from a security attack, security is now considered a key component that must be incorporated into the complete enterprise application development cycle. Besides normal security practices that protect the enterprise applications from unauthorized access and security attacks, such as a Denial of Service (DOS), an important business driver for security comes from federal and government regulatory compliance requirements, such as the Sarbanes-Oxley Act of 2002 (SOX) and Health Insurance Privacy and Portability Act of 1996 (HIPPA).

These regulations essentially mandate many organizations to take a more proactive approach by designing and building security into enterprise applications by default. To resolve the issues of scalability, distribution, and security, and reduce the risks, a new job role called the architect was created. The responsibility of an architect is to create a blueprint (architecture) that facilitates the successful design, implementation, and deployment of enterprise applications. It is at the architecture level that many quality-related decisions are made, and these decisions are used in the design, development, and deployment of the applications.

A more sudden change has occurred in the area of distribution. In recent years, more applications are expected to run over a network. At the minimum, these applications require a client and a server. The rapid increases in load that are typical of Internet-deployed systems can require additional distributed middleware components such as additional load balancers and server instances on the server side. This further increases distribution.

Client-Server System

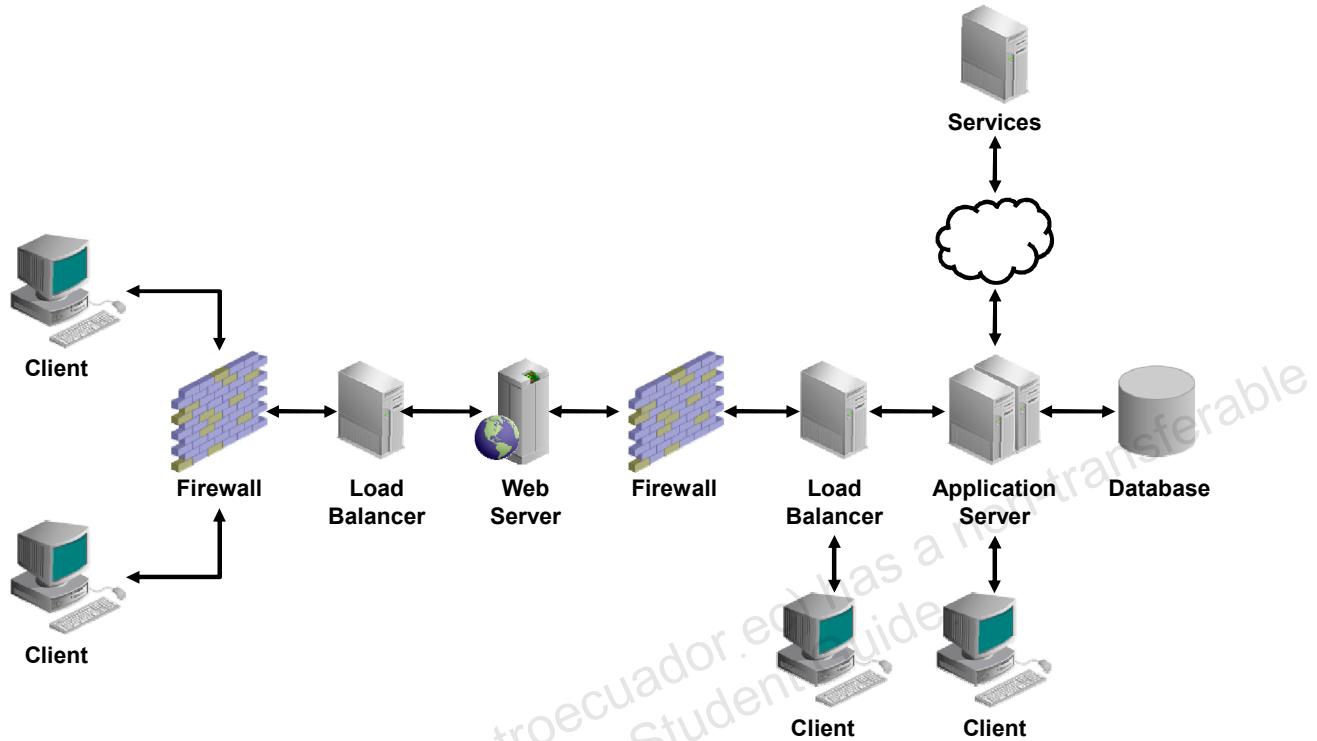


ORACLE®

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

In a system that runs entirely on a single host, all pieces of the system communicate with each other reasonably quickly and at approximately the same speed. Even when the system becomes a client-server system, as shown in the slide, communication is easy to control. This situation gave rise to the old adage “Compute near the data; validate near the user.”

Highly Distributed System



ORACLE

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

In a more distributed system, the ability to make the right choices about where components are located and how they communicate becomes more critical and difficult. The importance of these types of decisions created the need for an architecture for the system.

The critical role of the architect is to perform high-level planning for the location of, and the communication among, software components. This role distinguishes the architect from the designer, programmer, integrator, and others.

The goal of this high-level planning is to ensure that the system:

- Is robust if partial failures occur
- Handles the required load
- Is scalable when the demand for concurrent use exceeds the original design parameters

Quality of Service

The architect is primarily concerned with quality of service (QoS):

- Project failure
- Nonfunctional requirements:
 - Constraints
 - Systemic qualities



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

The architect is primarily concerned with quality of service: ensuring that the performance and reliability of the system are high enough that the users are willing to use the system. The architect works with use cases and provides crucial input to the architectural development. But use cases are not the architect's main focus.

Project Failure

Project failure is not usually attributable to functionality problems. Many projects reach functional completeness, but they are so delayed or otherwise unacceptable to the end users that they are abandoned. Attention to systemic qualities can help you to avoid these pitfalls and can lead you to project success.

Nonfunctional Requirements

The focus on nonfunctional requirements is generally considered to be the primary purpose of architecture. This focus has become necessary because a highly distributed system mandates a good architectural design to ensure that the system performs sufficiently well when all the functional requirements are met.

Systemic qualities: The requirements that guarantee a certain quality of service can be achieved after the system is deployed. Systemic qualities include scalability, performance, availability, and reliability, among others.

Generally, the focus on quality of service is a surprise to many experienced designers. They often find focusing on quality of service to be frustrating, and they want to work with more tangible design aspects. However, experience is proving that as long as a system has a good architecture, other problems are fixable. By contrast, the system lacking a good architecture might be functionally perfect, but still require a total rewrite to be usable or to revise the functionality.

Risk Evaluation and Control

- A good architect controls risk to guarantee that the nonfunctional requirements are met by the system design.
- This means risk control is a project cost, and it must be treated as such.
- Risk Evaluation: To make qualified decisions about how much risk control is enough and how much is too much.
- Cost Analysis: The best way to approach the problem is to perform a cost analysis of several options for controlling risk.



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

Risk Evaluation

To make qualified decisions about how much risk control is enough and how much is too much, you evaluate and compare the risks and mitigation strategies based on cost and probability of occurrence. You do not want to create high-cost solutions to mitigate low-probability risks. If designing such a solution was your goal, you could simply design systems that provide T1 connections and mainframes for every client that connects. This is not the best solution, and an architect should not take a position of throw more hardware at it... that will fix all the problems.

Cost Analysis

The best way to approach the problem is to perform a cost analysis of several options for controlling risk. Some options might be less costly, but leave some risk in place; others might be more costly, but control the risk completely. You can always look at the do nothing option: assume that the risk is realized and examine the cost impact. This pay me now, or pay me later approach to deciding what architectural solutions fit best is formalized in the return on investment (ROI).

For example, consider the use of a single server to provide availability compared to the implementation of the system on a server cluster. The cost of implementing a single-system environment is significantly less than that of the cluster: usually three to five times cheaper, possibly even more. This savings is due to the number of systems required to create a cluster (two to five or more, depending on configuration and types of systems used) and the level of expertise required to install and maintain the configuration.

On the other hand, you must consider the cost of lost time and business in the single-system configuration. If the system goes down, how long would it take to replace it? How much money could be lost while the system is being replaced? And how often is the single system likely to fail? This analysis might indicate that, even though the cost of implementing and maintaining a cluster might appear to be more costly at first, the long-term cost of not using a cluster might justify its use.

The Goal of Architecture

Among the goals of architecture are:

- Resolve the issues of scalability, distribution, and security
- Reduce project risks
- Capture quality-related decisions
- Facilitate the design, implementation, and deployment of the application
- Provide clear vision for IT staff
- Provide relationship between business need and IT implementation
- Absorb changes in technology more easily and quickly
- Measure ROI for IT



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

Due to the increase of software complexity, more and more organizations are starting to consider the creation and validation of the architecture as essential activities to ensure the quality of the product.

What Is Enterprise Architecture?

“Enterprise Architecture (EA) is a method and an organizing principle that aligns functional business objectives and strategies with an IT strategy and execution plan.”

-Oracle Enterprise Architecture

“1. A formal description of a system, or a detailed plan of the system at component level, to guide its implementation”
(source: ISO/IEC 42010:2007).

2. The structure of components, their inter-relationships, and the principles and guidelines governing their design and evolution over time.”

- The Open Group

 ORACLE

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

The idea of Enterprise Architecture began with the work of Dewey Walker, IBM's director of architecture in the late 1960s. He produced architecture planning documents that later became known as Business Systems Planning. One of his students, John Zachman, who became widely recognized as a leader in the field of enterprise architecture, identified the need to use a logical construction blueprint (i.e., an architecture) for defining and controlling the integration of systems and their components.

Zachman saw that architecture was the bridge between IT strategy and implementation. His framework defined and captured an architecture from six different perspectives in the enterprise: Data, Function, Network, People, Time and Motivation. Since Zachman introduced his work, there have been other frameworks proposed. We will examine these in the next lesson.

Enterprise Architects and Enterprise Application Architects

- Enterprise Architect's work:
 - Standards
 - Guidelines
 - Practices
 - Processes
 - Tool selection
 - Library & Framework selection
 - Methodology selection
 - Policies and procedures
- Enterprise Application Architect's work:
 - Technical software architecture
 - Design for nonfunctional requirements
 - System solutions



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

In this course, we draw a distinction between role of Enterprise Architect and Enterprise Application Architect.

Enterprise architects are responsible for nontechnical aspects of the enterprise architecture. This includes:

- Defining and ensuring compliance with enterprise IT standards
- Defining guidelines for system development
- Defining, codifying, and publishing best practices
- Selecting tools: IDE, Servers, Databases, Libraries, Frameworks
- Selecting methodologies and ensuring compliance with the methodology across the enterprise
- Defining, managing, and implementing policies and procedures around EA.
- Defining reference architectures
- Ensuring interteam standards and consistency
- Developing change management process for architecture
- Developing and communicating standards
- Evaluating and selecting contractors, outsource providers, and hiring standards

Enterprise Application Architecture is responsible for the implementation of the EA at the application level. This involves implementing the technical architecture, designing to account for nonfunctional requirements (performance, response time, capacity planning, sizing, and so on) and crafting a system solution.

Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- What Is enterprise architecture
- An architect's roles and responsibilities



ORACLE

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

The Architect Role

- “The architect’s role is to make sure that there is a vision... So there is a lot of negotiation and envisioning that becomes part of an architect’s role that is above and beyond just the technology... there is an art to be able to take the long-term vision for an organization”

- Jeanne Ross, *the MIT CISR*

- “An enterprise architect is more like a city planner. It's looking at not just the building, but what facilities are needed, what the likely through-put is going to be, the demands on the system, all the things a city planner would look at if he were building a city from scratch or an infrastructure project in a city.”

- Steve Nunn, *vice president and COO of the Open Group*

ORACLE

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

To resolve the issues of scalability, distribution, and security, and reduce the risks, a new job role called the architect was created. The responsibility of an architect is to create a blueprint (architecture) that facilitates the successful design, implementation, and deployment of enterprise applications. It is at the architecture level that many quality-related decisions are made, and these decisions are used in the design, development, and deployment of the applications.

Java EE technology can help address all these issues. Systems that are built with Java EE technology are typically divided into multiple tiers, and these tiers communicate over the network. The first tier is conventionally a Web browser that communicates through web protocols with the web container. The Java EE container software, for both web and Enterprise JavaBeans (EJB) technology, can be built to handle replication, redundancy, and failover without requiring special considerations in the application software. Furthermore, the Java EE platform provides a flexible security model that makes the development and deployment of the application easier.

More than likely, making architecture-level decisions requires the architect to consider different trade-off options. Besides quality-related requirements, additional factors, such as budget, time-to-market, and the development team's skill set, can also affect the decision making. When creating the architecture, the architect must take these factors into account.

The Architect's Involvement

This includes:

- Help analyze requirements and create the use case model
- Help define SLR for nonfunctional requirements
- Identify appropriate hardware, software, personnel resources
- Validate the project plan
- Identify architecturally significant entities and their relationships
- Define the overall technical structure of the system
- Identify high-level application components and major reusable services, and technologies to implement them
- Create the architectural prototype to support end-to-end testing



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

The architect is involved throughout the development process to ensure the project success. The architect must ensure that the final product conforms to the original vision of its design.

The architect can help analyze the requirements and create the use-case model to capture the functional requirements, and Service-Level Requirements (SLR) for non-functional requirements. The architect works with the project manager to identify appropriate hardware, software, and personnel resources for the project and validate the project plan.

Also, the architect works with the artifacts produced in the requirements analysis to identify architecturally significant entities and their relationships for the system under development. This work results in an artifact called the Domain Model. Based on the domain model, the architect defines the overall technical structure of the system. The outcome of this activity is a preliminary Architecture Description Document (ADD) that describes a skeleton system.

The architect refines the ADD by identifying high-level application components and major reusable services, and selecting appropriate technologies used to implement these components and services. To validate these decisions, the architect typically develops an architectural prototype to allow end-to-end tests. The goal of these tests is to ensure that the system built based on the architectural prototype can meet the project requirements defined in the SLR. The architectural prototype serves as the backbone of the system under development.

Responsibilities of an Architect

Technical responsibilities:

- Create the architecture to address quality-related requirements.
- Identify architecturally significant use cases involving
 - Concurrency
 - Performance
 - Legacy integration
- Recommend necessary technologies and frameworks for the project.
- Develop or guide the development of the architectural prototype.



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

The architectural function includes technology and management responsibilities, which are aimed at turning a business problem into a technical solution.

The technology responsibilities of an architect include:

- Creating the architecture to address quality-related requirements and mitigate the risks of the product. Because almost all development projects are resource constrained, the architect must make trade-off decisions to satisfy the most important quality-of-service (QoS) requirements.
- Identifying use cases, which are aspects of the system that have the greatest degree of risk associated with them. An architecturally significant use case typically involves the following QoS requirements:
 - Concurrency requirements
 - Timing or performance requirements
 - Legacy integration requirements

Recommending the necessary technologies, libraries, and frameworks used for the project. In fact, it is not uncommon that the architect recommends a proven set of technologies, libraries, frameworks, and development guidelines as part of an organization's development policy.

Developing or guiding the development of the architectural prototype. This prototype might be purely conceptual in the form of documents and diagrams. More commonly, it might involve an element of implemented code. Sometimes, the architectural prototype also serves as a mentoring tool that allows the development team to gain more insight into the project and technologies involved.

Depending on the nature of the project, architecturally significant use cases can be identified based on other QoS requirements involved, such as system flexibility and maintainability. For example, the goal of the project is to develop a framework that is independent of any specific products, then system flexibility can be the most important QoS requirement.

Java EE technology simplifies many of the design and coding issues related to large enterprise systems. This simplicity results from the nature of the framework that Java EE technology offers. However, the framework also imposes constraints on the solution that is created, and a solution that does not take into account the peculiarities of Java EE technology is likely to perform poorly or fail to fulfill the overall expectations of the users. For this reason, an understanding of the Java EE technology is important even at the architectural level.

Responsibilities of an Architect

- Management responsibilities:
 - Understand and manage project cost
 - Manage communication
 - Refine and clarify requirements
 - Convince stakeholders of the validity of decisions made
 - Mentor the project team members
- Governance responsibilities:
 - Ensure compliance with specified architecture.
 - Ensure consistent business rules implementation.



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

The architect must also accept some management responsibilities. The exact nature of the management skills required varies according to the particular project and company, but the responsibilities include:

Understanding cost: Individuals with budget responsibility and some of the stakeholders often lack the experience and knowledge to make decisions about the technologies and approaches to risk mitigation. However, these decisions are their responsibility. The architect must provide information and guidance to enable them to make correct decisions. In some companies, the architect is required to make these decisions directly.

Managing communication: Architects require interpersonal skills to work effectively with stakeholders and team members on the following tasks:

- Refining and clarifying business and user requirements
- Convincing stakeholders of the validity of the decisions that have been made
- Mentoring team members in the proper use of the chosen technologies

Architect and Other Team Members

- The architect must take into account the working practices and skills of the team as a whole.
- The architect must understand how the architecture impacts the team.
- The architect works with the team to ensure that the product stays true to the architecture.



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

The development team can be composed of individuals who serve in a variety of roles. Depending on the working practices, the architecture might be created by an individual, or by many people within the team. Either way, the architecture must take into account the working practices and skills of the team as a whole. Therefore, it is important to understand what the team must achieve, to appreciate how the architecture impacts the team, and to consider how the architect might interact with the team.

The architect owns the structure of the final product in the form of a system blueprint and works with the team to ensure that the product stays true to the system blueprint.

Team Member Functions and Tasks

Team leadership

- Acts as a liaison between the system designer and the development team
- Answers questions that are raised by the development team with regard to the implementation of the design
- Works with the architect and system designer if problems occur with the implementation

Enterprise modeling

- Models the enduring business themes of the application and helps to bridge the business to the architecture
- Is sometimes known as enterprise modeler, business analyst, domain modeler, or system analyst

System architecture

- Visualizes design and implementation for hardware and server software
- Is experienced or familiar with database design, capacity planning, server clustering, load-balancing, and fail-over strategies
- Provides a deployment environment that delivers scalability, high availability, and reliability
- Is sometimes known as a system architect or infrastructure architect

Application architecture

- Visualizes design and implementation for application software and component integration
- Is experienced or familiar with typical business applications, integrating applications, and object-oriented methodology
- Provides an application structure that both delivers end-to-end functionality and supports nonfunctional requirements

System design

- Uses the architectural blueprint to create a more detailed design plan of the application infrastructure
- Serves in roles, such as system designer, cluster engineer, storage engineer, and network engineer

Application design

- Uses the architectural blueprint to create a more detailed design plan for implementation by software developers

Software development

- Implements the architectural design

Testing

Tests subpieces of the design and reports potential pitfalls of the architecture

Quality assurance (QA)

- Tests the completed units to verify functionality
- Verifies that the inputs and outputs of the pieces of the system are exactly as expected

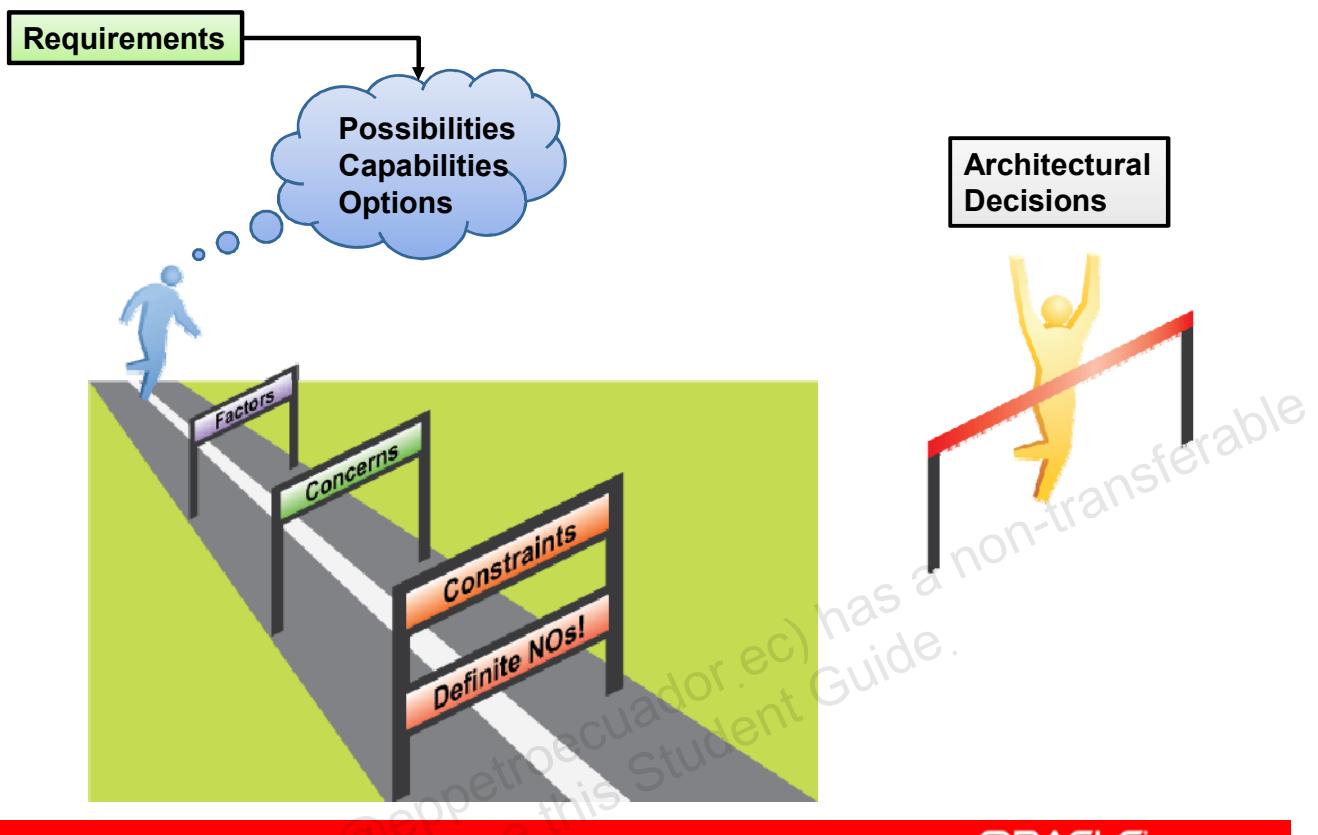
Configuration

Configures the application for a specific operating domain

Stakeholders

Stakeholders have an interest in the product and can include users, customers, and the development team.

Analyze – Evaluate – Prescribe



ORACLE®

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

In this course we will focus on the role of the architect in analyzing, evaluating, and prescribing architectural choices, strategies, and solutions.

It is the architect's responsibility to identify possible options based on the stated requirements. The architect must then weigh the capabilities in light of various factors, which we will examine in more detail throughout this course.

Influencing Factors



ORACLE

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

Many factors influence the choices and recommendations of the architect. The slide lists only a few. Some factors turn out to be constraints. And some choices are simply “forbidden.”

For example, as an architect you may suggest using biometric data to meet the stakeholder’s requirement of the most reliable way to identify a user. However, company culture forbids that form of technology. So, you must choose the next best idea, realizing you must sacrifice meeting the requirement as stated.

Of course, there is typically no “one right answer.” It is a matter of trade-offs and negotiation.

Note: CMM Level refers to the SEI Capability Maturity Model Integration: “CMMI can be used to guide process improvement across a project, a division, or an entire organization. It helps integrate traditionally separate organizational functions, set process improvement goals and priorities, provide guidance for quality processes, and provide a point of reference for appraising current processes.” <http://www.sei.cmu.edu/cmmi/>

What Do You Think?

“Enterprise architecture has almost nothing to do with technology. But it has almost everything to do with people.”

- Oracle ACE Director Mike van Alst

On the organizational/human side of the enterprise architecture equation, the convergence of Baby Boomers and members of Generations X and Y—as both employees of the enterprise and as its customers—has exposed differences in each generation’s experience and expectations. Does bridging the generation gap hold an important key to enterprise architecture success? Why or why not?



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

“A lot of the old architects—well, I’m one of them—have IT backgrounds,” observes van Alst, an independent architect. “A lot of the younger architects have more of a business background. That helps the older architects maintain a focus on business properties.” Younger architects, in turn, can benefit from the fact that older architects have been around the block more than once. “We can help them,” van Alst says, “by adding our experience with all those solutions that did not work well for the last 25 years.” That experience can also help to imbue younger architects with a long view that can counteract the hype that often surrounds new technologies.

Finding ways to bridge the gaps between those generations within the enterprise can help to facilitate change and keep pace with the rapidly evolving marketplace. Social computing is likely to play a significant role. Bridging the enterprise generation gap is more than just a nice idea. “Cooperation in itself will be a very pervasive principle,” says van Alst, adding that it is “a guiding principle for enterprise architecture.”

What Do You Think?

To be a great architect, you need talent. But greatness is accorded to those responsible for great architectures, those that stand out from others in the value they afford. The great enterprise architect is one who is credited with contributing in a significant way to a great enterprise.

- Dana Bredemeyer and Ruth Malan

In addition to technical skills, what other skills, talent, qualities, and knowledge are required to be a great architect?



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

EA “nontechnical” skills:

- **Manage Communications between teams:** Organizational communication is more important than ever. Work is more complex and faster, workers are more distributed and “tribal knowledge” is of greater competitive advantage. Ideas for enabling communication include: Wikis, chat programs, mailing lists, voice and video chats, Monday morning status reports and meeting minutes.
- **Project Management:** An architect needs to manage project information and interdependencies. They also need at least basic project management and planning skills.
- **Political influence:** In addition to crafting a good architecture, an architect must be able to drive support for the architecture with the stakeholders. An excellent architecture is worthless unless it is adopted and utilized. It is considered harder to implement (put in place and utilize) than build an architecture. Building influence in the organization is long and hard work with no quick clear outcome, unlike building an architecture. It requires a different skill set. The focus here is on relationships and people not technologies and concepts. It is not unusual to hear architects say this is out of their comfort zone. Sponsorship and authority and control are the most requested political support, yet many do not receive this and are still successful. How? By building personal and organizational influence.

- **Essential Skills:** Organization, communication, and Influence
- **Organization:** The ability to bring order to chaos; creating effective ways for getting things done and producing consistent results (encompasses systemic and systematic thinking).
- **Communication:** The ability to inspire, motivate, and deliver critical information (vision, values, purpose) in a consistent and clear manner.
- **Influence:** The ability to positively affect people and create organizational alignment. Help others overcome resistance, reluctance, and inertia.

Additional Resources

- Lankhorst, Marc. *Enterprise Architecture at Work: Modelling, Communication and Analysis*. Springer, 2nd ed., 2009.
- Ross, Jeanne. *Enterprise Architecture As a Strategy*. Harvard Business Press, 2006.
- Bredemeyer, Dana and Ruth Malan. *What it Takes to be a Great EA*. Executive Report, Cutter Consortium, Vol 7, No. 8, 2004.



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

<http://www.infoq.com/news/2010/07/EnterpriseArchitect>

http://advice.cio.com/michael_kavis/role_of_the_enterprise_architect

<http://www.bredemeyer.com/Architect/RoleOfTheArchitect.htm>

<http://www.itbusinessedge.com/cm/blogs/lawson/defining-the-enterprise-architect/?cs=15963>

Rhubart, Bob. *Something Old, Something New*. <http://www.oracle.com/technetwork/issue-archive/2010/10-sep/o50architect-165466.html>

Quiz

Crucial changes in the design and development of software have led to the need for the role of an architect. Select all the requirements that apply:

- a. For systems to scale
- b. For systems to have better User Interface designs
- c. For systems to have security designed in up-front
- d. For systems to move from client-server to highly distributed



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

Answer: a, c, d

The need for good user interface designs are a constant in the development of software.

Quiz

Highly distributed systems separate the required components of a system into discrete elements. One of the most important considerations for a highly distributed system is:

- a. That the right vendors are chosen for their compatibility
- b. That the proper version of Java Enterprise Edition is chosen to support the components required
- c. To handle increased traffic if the application becomes popular among users
- d. To separate the database and application server layers



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

Answer: c

Distributed systems make it possible to anticipate changes in increased traffic.

Quiz

Quality of Service (QoS) is an Architect's primary focus. Which of the following are critical Non-Functional Requirements (NFR's) that will affect the user's satisfaction and acceptance of a system? Choose all that apply.

- a. Page loads and refreshes occur in under 1 second at nominal load
- b. Pages must include in-line validation for form fields
- c. Database tables cannot exceed 2 Gigabytes in size
- d. The Oracle 11g customer database must be used for this project



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

Answer: a, c, d

B is really a functional requirement.

Summary

In this lesson, you should have learned how to:

- Describe the goals of the course
- Define enterprise and enterprise application architecture
- List the challenges of enterprise applications
- Define software architecture
- Describe the need for software architecture
- Define an architect's roles and responsibilities



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

Practice 1 Overview: Review Questions: Introducing Enterprise Architecture

This practice covers the following topics:

- What is enterprise architecture
- An architect's roles and responsibilities

Introducing Fundamental Architectural Concepts



ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Distinguish between architecture and design
- Describe the use of architectural patterns
- Describe the architecture workflow
- Describe the diagrams of the key architecture views
- Select the architecture type
- Describe common enterprise architecture frameworks



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

Discussion Questions

- What, if anything, makes architecture different from design?
- Does object-oriented analysis and design have anything to do with architecture?
- What goes on in the environment in which architecture is created and turned into a product? Why does the architect care?
- What documents does the architect produce? What purpose do these documents serve?
- Is it important for the architect to be experienced in evaluating Java Platform, Enterprise Edition (Java EE) technologies? If so, why?
- What are patterns? Why are they relevant?

ORACLE

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

Lesson Agenda

- Distinguish between architecture and design
- Introduction to architectural patterns
- Architecture deliverable artifacts
- Architecture modeling using UML
- Architecture workflow
- What Is an Enterprise Architecture framework?



ORACLE

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

Which Is More Important, Architecture or Design?

Didn't you say this system's Architecture was nice and clean?



Yes... You just can't see it because it's hidden behind the code.



ORACLE

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

Can a good architecture make up for a bad design?

Can a good design overcome a bad architecture?

Distinguishing Between Architecture and Design

| | Architect | Designer |
|-------------------|---|--|
| Abstraction level | High/broad Focus on few details | Low/specific Focus on many details |
| Deliverables | System and subsystem plans, architectural prototype | Component designs, code specifications |
| Area of focus | Nonfunctional requirements, risk management | Functional requirements |



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

It can be difficult to distinguish between architecture and design because the distinction varies between companies and even between projects within a company. However, you can draw some useful comparisons about the architecture and design functions, based on the common elements of a number of projects.

Adding to the often blurred distinction between the role of the architect and designer is that the architect is likely to perform functions of enterprise modelers, system (infrastructure) architects, application architects, system (infrastructure) designers, and application designers. The architect role tends to vary in different companies and projects, based on whether the architect is expected to perform any or all of these additional functions.

Note: Java EE technology helps to clarify the distinction between architecture and design because it allows an architect to think in terms of high-level abstractions that are supported by the Java EE technology. However, to avoid causing technology-specific architectural problems, the architect must understand the ways in which the choice of technology might impact the architecture.

Common Principles Between Architecture and Design

- Common characteristics of architecture and design:
 - Abstraction
 - Encapsulation
 - Cohesion
 - Coupling
- Like design and implementation, the architecture also requires refactoring to incorporate changes.

ORACLE

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

When you develop the architecture of a system, you focus on planning subsystems. Therefore, the basis for the architecture is components, rather than individual objects. However, these components are essentially high-level objects, and the principles of good object-oriented design also apply at the architectural level.

Four characteristics are central to the quality of a component-based design:

- **Abstraction:** Focuses on the architecturally significant details of a component. Other details are addressed in a more detailed design phase. Abstraction helps the development team modularize the components of a system. It also breaks up tasks so that the team can work in parallel, yet independently, and decrease the time to market. An example of an abstraction is a box labeled “business object” that represents the business logic. The details are hidden, but everyone on the team understands what this box represents. As mentioned previously, a key difference between architecture and design is in the level of details. Architecture operates at a high level of abstraction, while design operates at a low level of abstraction.

- **Encapsulation:** Supports abstraction by hiding the details that were ignored at the time the abstraction was defined. Encapsulation creates a boundary between the details that created the abstraction and the components that use the abstraction.
- **Cohesion:** Supports abstraction and encapsulation by ensuring that all of the details hidden by the encapsulation boundary contribute to the definition of the abstraction. Any detail that does not contribute to the abstraction reduces cohesion and is considered extraneous.
- **Coupling:** Measures the connectedness of components. Inside an encapsulation boundary, coupling can be helpful because it adds to the cohesion of the abstraction. Outside the boundary, coupling can be a hindrance because it reduces the flexibility and reusability of components.

Architectural Principles

Architecture principles are axioms or assumptions that suggest good practices when constructing a system architecture:

- Separation of Concerns
- Dependency Inversion Principle
- Separate volatile from stable components
- Use component and container frameworks
- Keep component interfaces simple and clear
- Keep remote component interfaces coarse-grained



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

Architecture principles are axioms or assumptions that suggest good practices when constructing a system architecture. These principles form the basis of many architectural patterns and form the decisions an architect makes to satisfy the quality of service requirements.

Separation of Concerns

This principle tells the architect to separate components into different functional or infrastructure purposes. For example, it is important to separate the user interface from business logic (called the Model). Even user interfaces can be separated into visual elements (called a View) and user interaction logic (called a Controller). This separation of concerns leads to the Model-View-Controller (MVC) architecture pattern.

Dependency Inversion Principle

Depend upon abstractions. Do not depend upon concretions. A client component that depends on an interface makes the software more flexible because the supplier component can be changed or replaced without affecting the client component.

Separate volatile from stable components

As a system changes, you want to reduce the changes to a small number of packages.

This principle guides the grouping of components into volatile elements (such as the user interface) and more stable elements (such as the domain entities).

Use component and container frameworks

A component is a software element that is managed by a container. For example, a servlet is managed by a web container. Typically, a software developer creates components. These components are built to fulfill interface or language specification.

A container is a software framework that manages components built to a certain specification. For example, a web container is a framework for managing HTTP requests and dispatching the requests to a servlet. A container is a system (or library) that you can build or acquire. For example, Tomcat is an implementation of a web container.

A container framework is a powerful tool for the architect. These frameworks enable the architect to purchase software that provides infrastructure for the rest of the software system. This enables the designers and programmers to concentrate on the business logic components rather than the infrastructure.

Keep component interfaces simple and clear

The more complex a component's interface is, the harder it is for software developers to understand how to use the component. Component interfaces should also be highly cohesive.

Keep remote component interfaces coarse-grained

Remote components require network traffic to communicate. You should keep the number of remote requests (which requires sending network messages) to a minimum. Therefore, keep remote interfaces simple and coarse-grained. For example, a LoginService component could have three methods: `setUserName`, `setPassword`, and `performLogin`. However, this would require three network messages to perform a single login operation. Instead, the LoginService component could have a single method: `login(username, password)`. Even though both techniques send approximately the same amount of data, the latter technique incurs the latency of only one network message rather than three.

Lesson Agenda

- Distinguish between architecture and design
- Introduction to architectural patterns
- Architecture deliverable artifacts
- Architecture modeling using UML
- Architecture workflow
- What Is an Enterprise Architecture framework?



ORACLE

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

Architectural Patterns and Design Patterns

- Patterns are standard solutions to commonly recurring problems in a particular context.
- By using a pattern-based reasoning process to plan systems, the architect can analyze systems to identify problems and trade-offs with respect to service-level requirements.
- Each of these problems might be solved by applying a particular pattern, or it might require further analysis and decomposition.
- The architect can then synthesize the system by combining the pattern solutions into aggregate constructs.



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

To be effective at selecting and applying patterns, an architect must be familiar with a variety of pattern catalogs. Each of these catalogs focuses on a particular aspect of system development: analysis, design, architecture, and technology.

The architect primarily uses design and architectural patterns:

- Design patterns define the structure and behavior to construct effective and reusable OO software components to support the functional requirements.

For example, one commonly used design pattern is the Composite pattern. This pattern supports the ability to treat collections of objects in the same manner (using the same methods) as the individual objects. For example, a computer manufacturer might have an inventory system that calculates cost of individual components, such as memory sticks, CPUs, hard drives, and so on, as well as complete system, which are a composite of individual components. This is a functional requirement.

- Architectural patterns define the structure and behavior for systems and subsystems to support the nonfunctional requirements. For example, one commonly used architectural pattern is the Layers pattern. This pattern describes a technique for partitioning systems into abstractions in a way that allows coupling only between adjacent abstractions.

Common examples of layering include APIs, virtual machines, and client-server structures.

Another common pattern is the Pipes and Filters pattern. This pattern is based on the idea that data moves from host to host and undergoes some value-added processing within each host. A data transfer path is a pipe, and a data processor is a filter. Examples of this pattern include UNIX tools and many common compilers.

The relationship between architectural patterns and design patterns is basically one of scale and focus. Architectural patterns specify systems in terms of subsystems and high-level components that fulfill service-level requirements. Design patterns are used when creating the individual subsystems and lower-level components that make up the subsystems specified by an architectural pattern. Design patterns directly support the functional requirements.

Note: You use both architectural and design patterns in a similar way. To build large, complex constructs, you usually apply several patterns to achieve the structure and behavior that you are looking for.

Architectural Patterns

- The Layers pattern
- The Model View Controller (MVC) pattern
- The Tiers pattern



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

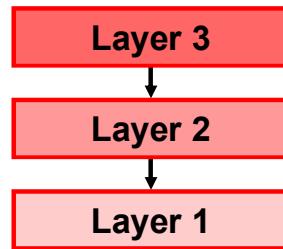
A pattern describes a recurring design problem that appears in a context, and presents a proven solution. Design patterns have proven to be an essential tool to help solve object-oriented design problems. Similarly, architectural patterns are helpful in creating the software architecture by providing a well-defined structural template for concrete software systems.

Commonly used architectural patterns in distributed enterprise application include the Layers pattern, the Model-View-Controller (MVC) pattern, and the Tiers pattern.

Architectural patterns, and patterns in general, are a way to document expertise. We will discuss other patterns throughout the course.

The Layers Pattern

The Layers pattern structures applications so they can be decomposed into groups of subtasks, and each group represents a particular level of abstraction.



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

A layer relies on services implemented in its adjacent lower layer to implement its own services. Typically, the Layers pattern is used to decompose the system to either isolate the dependency on system level services or limit the visibility of application-level components.

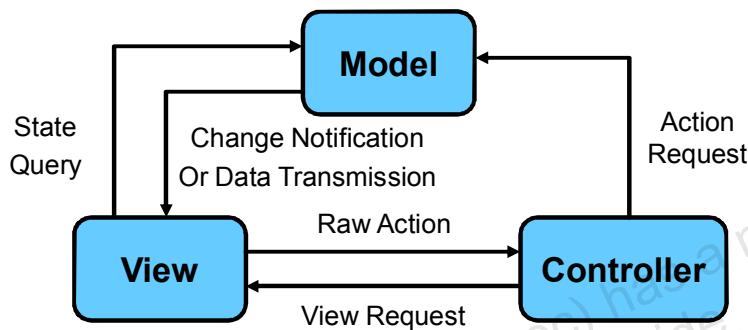
Examples of the Layers patterns include the Open Systems Interconnection (OSI) network model, the Transmission Control Protocol/Internet Protocol (TCP/IP) network model, and the multi-tiered architecture of enterprise applications.

The Layers pattern helps establish logical tiers and technology layers in a distributed system and ensures that abstraction boundaries are defined and enforced. However, you can use the Layers pattern at other levels in the design of architectures to promote flexibility.

As an example, consider a system that must integrate with a moderately volatile database. This scenario could arise when a company is in a period of acquisition and merger, and an application must maintain a level of integrity in the face of changing data representations. If the architect uses a database abstraction layer in the form of object wrappers, the application is insulated from database changes.

The MVC Pattern

The MVC pattern separates concerns in data presentation, processing, and storage. This pattern is useful for emphasizing separation of concerns.



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

The MVC pattern, shown in the slide, enhances the flexibility and maintainability of an application by separating the application into three distinct modules:

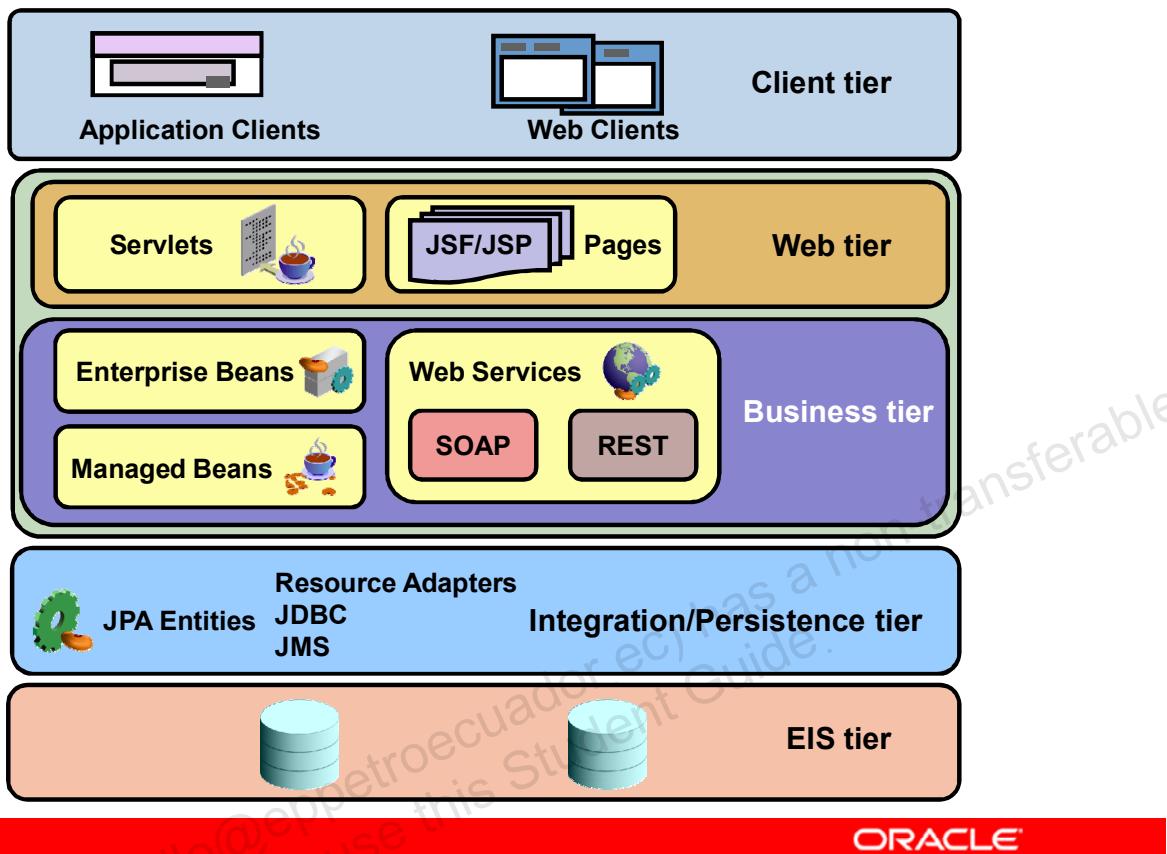
- **Model:** The data model
- **View:** The components for viewing the presentation of the data model
- **Controller:** The components for allowing interaction

You can use the MVC pattern to design a data model with clearly specified interfaces that allow the model to be manipulated and queried.

You can use the MVC pattern to ensure proper decoupling among components within a particular tier, which helps with the separation of concerns in developing application components.

For example, if you focus on the Web tier and apply the MVC pattern to this tier, the result is the Model 2 architecture for Java technology Web components.

The Tiers Pattern



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

Tiers describe how a distributed application is decomposed into modules to reduce the coupling and improve the system flexibility. This permits each tier to be modified or enhanced without affecting other tiers. Typically, an enterprise application can be decomposed to possess some or all of the following tiers: client tier, web presentation tier, business logic tier, integration tier, and enterprise resource tier.

The client tier contains application functions that are clients of the services implemented in the application. Examples of components that can appear in the client tier are web browsers, standalone Java clients, and web service clients.

The web tier contains the application modules that provide services to clients using web technology. Web tier modules receive requests through HTTP methods, and reply to those requests by sending some form of markup language, such as HTML or XML, back to the client.

The business logic tier contains application modules that implement the principle business logic of the application. Notice that both SOAP and RESTful Web services are supported. Also, managed beans, a generalization of the managed beans specified by JavaServer Faces, are lightweight, container managed objects.

The integration tier contains modules and technologies that assist other modules in connecting and communicating with the enterprise resources tier. Resource adapters allow Java EE application components to interact with and access the underlying resource manager of the EIS.

The enterprise resource tier contains systems and modules that provide valuable business functionality for the application. In an enterprise application, these are frequently non-Java modules, such as Database Management Systems (DBMSs), mainframe applications or Custom-Off-The-Shelf (COTS) packages.

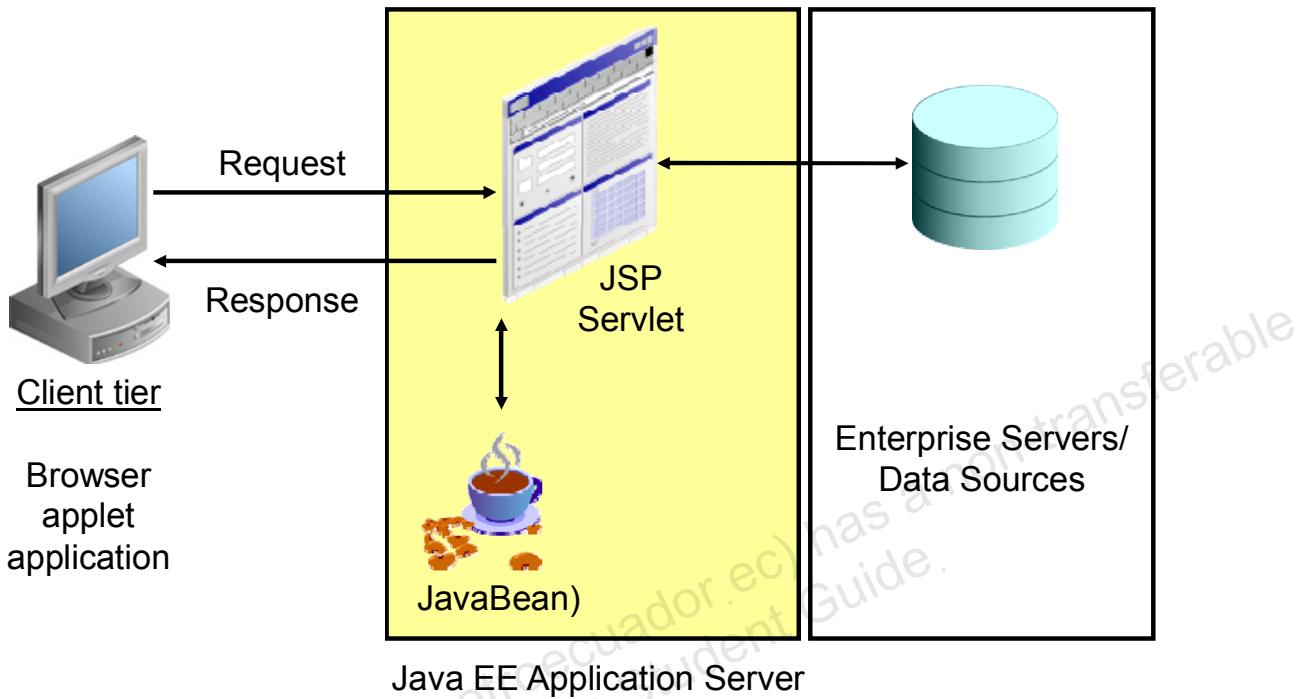
The Layers and Tiers patterns both follow certain principles:

- Each element should depend only upon the element immediately adjacent to it.
- All dependencies should propagate in the same direction.
- The most volatile parts should depend upon the more stable parts, not the converse.

The slide shows the canonical Java EE 5 Tier architecture.

Beginning with Java EE 6, Tiers are a more logical concept, and not literally physically separate tiers. For example, an EJB can physically execute in different containers (Client, Web, or Business) but may be logically assigned to a particular tier based on its use. So, an EJB may physically execute in the web container, but may be logically thought of as being "in" the Business tier.

Model 1 Architecture



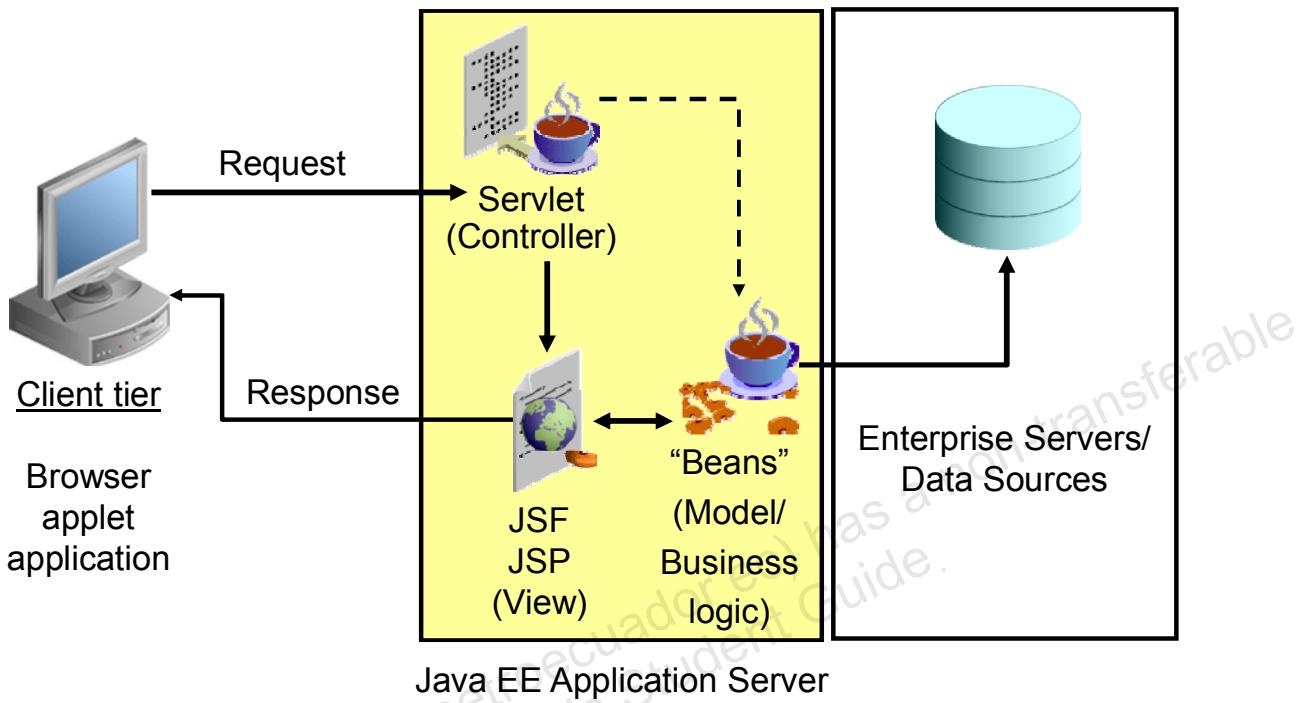
ORACLE

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

In the Model 1 architecture, the JSP page alone is responsible for processing the incoming request and replying back to the client. There is still separation of presentation from content, because all data access is performed using beans. Although the Model 1 architecture should be perfectly suitable for simple applications, it may not be desirable for complex implementations. Indiscriminate usage of this architecture usually leads to a significant amount of scriptlets or Java code embedded within the JSP page, especially if there is a significant amount of request processing to be performed.

Model 1 suffers from hard-coded navigation and blurring of the responsibility of the controller, model, and view. This is not normally used any longer in favor of Model 2 (next slide).

Model 2 Architecture

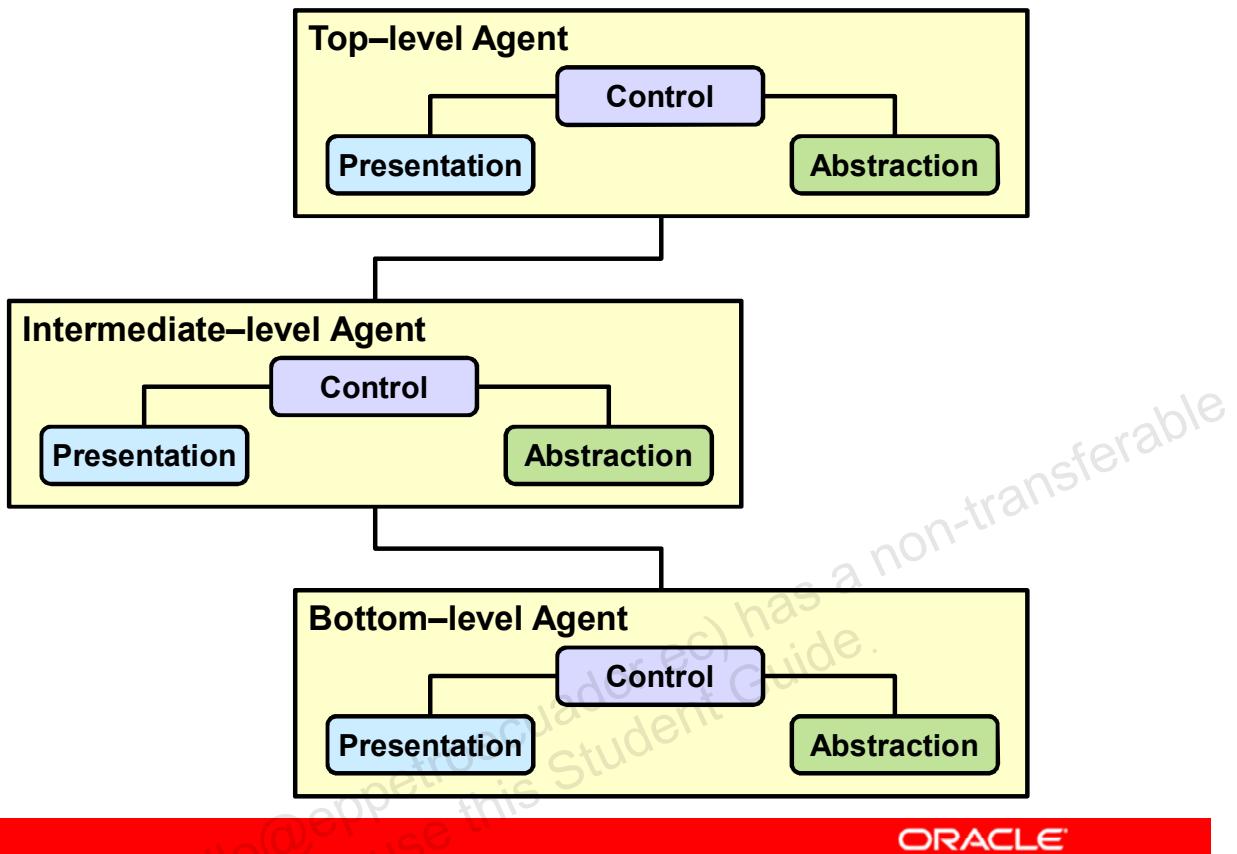


ORACLE

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

The Model 2 architecture, is a hybrid approach for serving dynamic content, since it combines the use of both servlets and JSP. It takes advantage of the predominant strengths of both technologies, using JSP to generate the presentation layer and servlets to perform process-intensive tasks. Here, the servlet acts as the controller and is in charge of the request processing and the creation of any beans or objects used by the JSP, as well as deciding, depending on the user's actions, which JSP page to forward the request to. Note particularly that there is no processing logic within the JSP page itself; it is simply responsible for retrieving any objects or beans that may have been previously created by the servlet, and extracting the dynamic content from that servlet for insertion within static templates.

PAC Architectural Pattern



ORACLE

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

The PAC pattern is a further refinement of the Model View Controller pattern. For complex UIs, it creates a hierarchy of interacting, cooperating agents. Each agent is its own “mini” MVC component. It differs from MVC in that within each agent, the presentation and abstraction are completely separated.

Use the Presentation Abstraction Control (PAC) pattern to introduce a level of decoupling between tiers and to further define the separation of concerns that was initiated with the MVC pattern. The presentation components reside on the presentation tier, and the control and abstraction components reside on the business tier. This makes the model flexible and allows for rapid changes in presentation without impacting the business model that is managed by the control and abstraction components.

Lesson Agenda

- Distinguish between architecture and design
- Introduction to architectural patterns
- **Architecture deliverable artifacts**
- Architecture modeling using UML
- Architecture workflow
- What Is an Enterprise Architecture framework?



ORACLE

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

Typical Software Deliverable Artifacts

Architect provides input into:

- Vision statement
- Functional requirements
- Nonfunctional requirements (NFRs)
- Domain Model
- Risk list
- Project plan
- Object-oriented design document

Architect is responsible for:

- Architecture Description Document (ADD)
- Architectural Prototype



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

In the course of a software development process, document-based artifacts are required for guidance and communication. The architect might be involved in creating most of the artifacts. However, the primary deliverables an architect is responsible for are the Domain Model, the Architecture Description Document (ADD), and the Architectural Prototype Service-Level Requirements (SLR) that define the systemic qualities the software product must possess, such as performance, availability, and security. System constraints that describe the software and hardware products and technologies that must be included in the implementation of the software system.

Vision statement: Establishes the business vision of the project, and identifies key features and business values of the project.

Functional requirements: Identify the behavior aspects of the system and describe the purpose of the system. These requirements are typically derived from a customer's wish list of the features, and represented using use case diagrams and documents.

Nonfunctional requirements (NFRs): Describe the following two categories of requirements:

- Service-Level Requirements (SLR)
- System constraints

Domain Model: Captures the structural and the behavioral relationships between entities represented as system-level components.

Project plan: Defines the major milestones that terminate the project phases and assigns use cases to increments within the phases. The bulk of the project plan maintenance work is performed by project managers.

Object-oriented design document: Provides a detailed model that facilitates the implementation of a software system.

Architecture Description Document (ADD): Represents the deployment environment and shows how and where software and service components should be located and how those components should communicate. The ADD is created based on UML diagrams.

Architectural Prototype: Provides an end-to-end demonstration or implementation of the primary technologies and components described in the Domain Model and ADD. Risk list: Lists known risks, an assessment of the risks, and mitigation plans that are intended to address the risks.

Architectural Blueprint

- Includes the Domain model and ADD
- Focuses on quality of service (QoS) related requirements, instead of functionality, because:
 - Multiple solutions can be used to implement the system functionality.
 - The architect must select a solution that meets the quality of service (QoS) related requirements.
 - Example: System must scale to handle 5% more concurrent user per month.
- Captures multiple views of the system



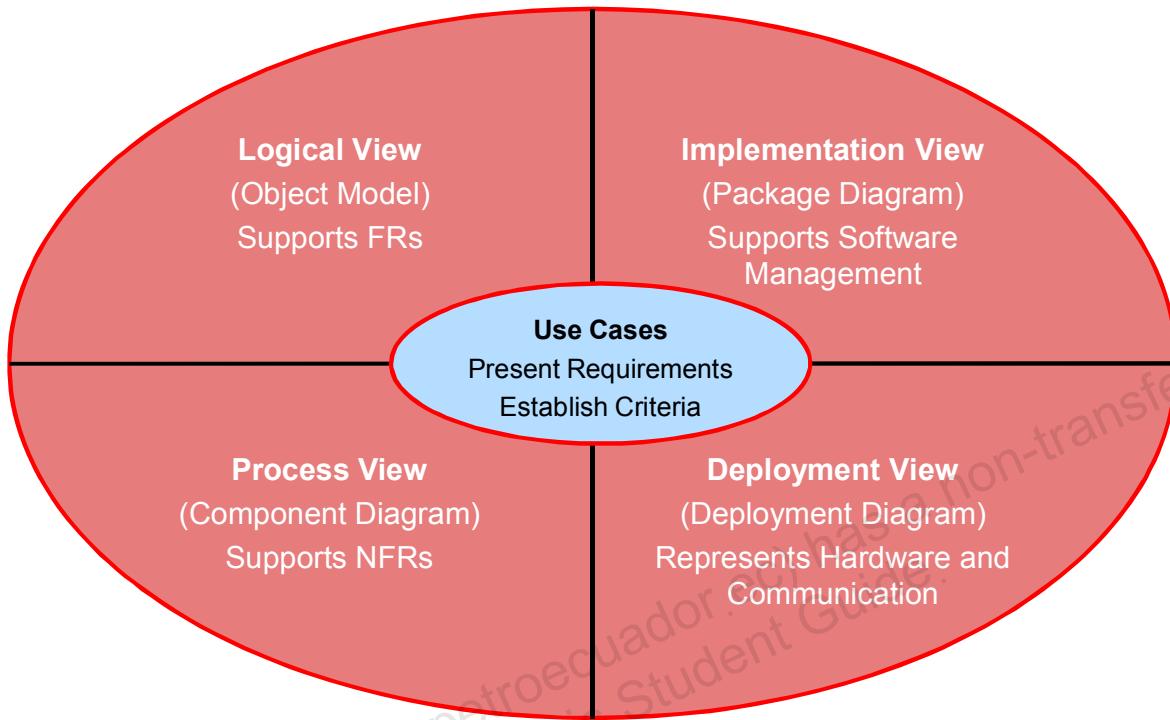
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The domain model and the ADD form the architectural blueprint of the software system. The blueprint must focus on addressing the quality-of-service requirements of the system. System and software designers use the blueprint as a guide for selecting products and specifying implementations.

At first, it might be surprising to know that the architectural blueprint focuses on quality-of-service requirements, instead of the functionality. The reason for this is that the functionality rarely dictates the outlook of the software. For example, in a typical business-to-customer (B2C) scenario, the application collects some input from the user by using a user interface, validates the input, triggers a transactional operation against a database, and sends the output back to the user.

There are multiple solutions to building this functionality. Each solution can present a different way to decompose the system into components and arrange their deployment. Among these solutions, the architect must choose the one that is most appropriate to meet the quality-of-service requirements of the system. For example, to make the system easily scale to accommodate more users in the near future, the solution might need to use a cluster that allows additional runtime instances to be added transparently. The architectural blueprint must clearly describe this information.

The 4+1 View Model



ORACLE

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

To capture the quality-of-service requirements in the architectural blueprint, multiple views of the system are required.

The 4+1 view model, shown in the slide, is one example of a multiple view model:

- **Logical view**: Supports the functional requirements and is represented by the object model. It describes the decomposition of the system into abstract software modules.
- **Implementation (development) view**: Supports software management. It describes how software modules are organized in the development environment.
- **Deployment (physical) view**: Supports the deployment and installation of the software system. It provides a representation of the hardware layout and communication protocols, and it also maps the software modules to hardware nodes.
- **Process view**: Supports nonfunctional requirements. It describes the concurrency and distribution of the functionality into processes and how these processes communicate.
- **Use case view**: Presents an abstraction of a requirement and establishes test criteria for architectural prototypes

The 4+1 view model includes both static and dynamic views.

The static views are also known as structural views. Structure diagrams emphasize the things that must be present in the system being modeled. Since structure diagrams represent the structure they are used extensively in documenting the architecture of software systems. These include: Class, Statechart, Component, and Deployment diagrams.

Dynamic views show the behavior of the system. Behavior diagrams emphasize what must happen in the system being modeled. Since behavior diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems. These include: Use Case and activity diagrams.

By considering multiple views, the development of the project is consistently guided by an architecture that takes into account the primary concerns of the system, including tiers, layers, and systemic qualities.

The Architectural Prototype

- Demonstrates the viability of the technologies and products prescribed in the architectural blueprint
- Provides a testing platform for the quality-of-service requirements
- Presents a concrete view of the component model, standards and other recommendations
- Forms the backbone of the project development moving forward



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

The architectural prototype is a model of the architectural blueprint. To create the architectural prototype, the architect should identify a small number of architecturally significant use cases, and create an end-to-end implementation of these use cases. You can create the architectural prototype at a conceptual level using a collection of documents and diagrams. It is also common to create a physical implementation of a skeleton system (some times referred to as an evolutionary prototype) that can be deployed and executed. In these cases, additional features are added to the skeleton system throughout the development process until the final product is delivered.

Lesson Agenda

- Distinguish between architecture and design
- Introduction to architectural patterns
- Architecture deliverable artifacts
- **Architecture modeling using UML**
- Architecture workflow
- What Is an Enterprise Architecture framework?



ORACLE

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

Architecture Modeling Using UML

The following section lists several types of UML diagrams that are typically used in documenting the software architecture:

- Class diagram
- Interaction diagram
- Component diagram
- Deployment diagram
- Package diagram



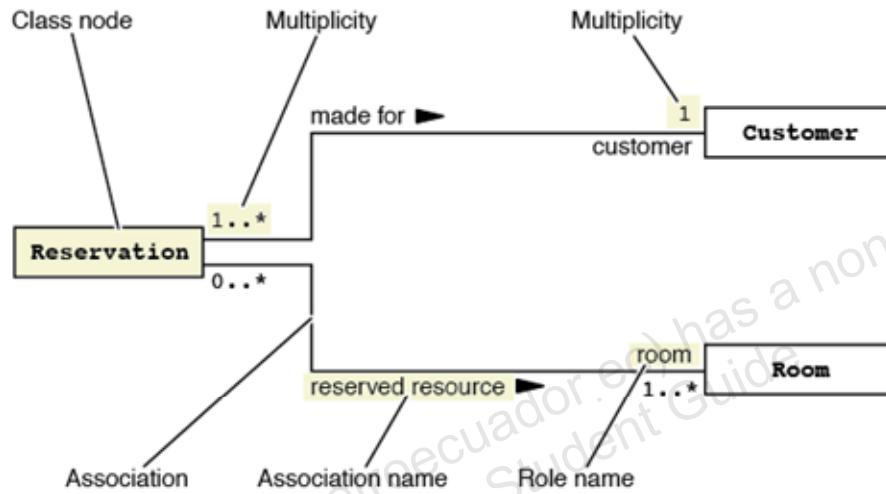
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The Unified Modeling Language (UML) provides an industry standard to visualize, describe, and document software systems. It provides a comprehensive set of diagrams to model system components and their interactions. UML diagrams are aligned with typical views of a software system so that they provide a powerful toolkit to model the architecture.

For a more detailed description of UML diagrams, refer to Appendix A.

Class Diagrams

Class diagrams visually represent classes, the members of the class, and the relationships between classes such as associations and inheritance.



ORACLE

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

The Class diagram in the UML is probably the most recognized. Class diagrams are typically used to describe the domain model.

Interaction Diagrams

UML Interaction diagrams are the collective name for the following diagrams:

- Sequence diagrams
- Communication diagrams
 - Formerly known as Collaboration diagrams
- Interaction Overview diagrams
 - A combination of an Activity diagram and a Sequence diagram fragments
- Use interaction diagrams to model the interaction and data communication between components



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

Each UML Interaction diagram is used to show the sequence of interactions that occur between objects during:

- One or two use case scenarios
- A fragment of one use case scenario
- Interaction diagrams highlight the following:
 - Which of the objects interact
 - What messages are communicated between objects
 - The sequence of the interactions

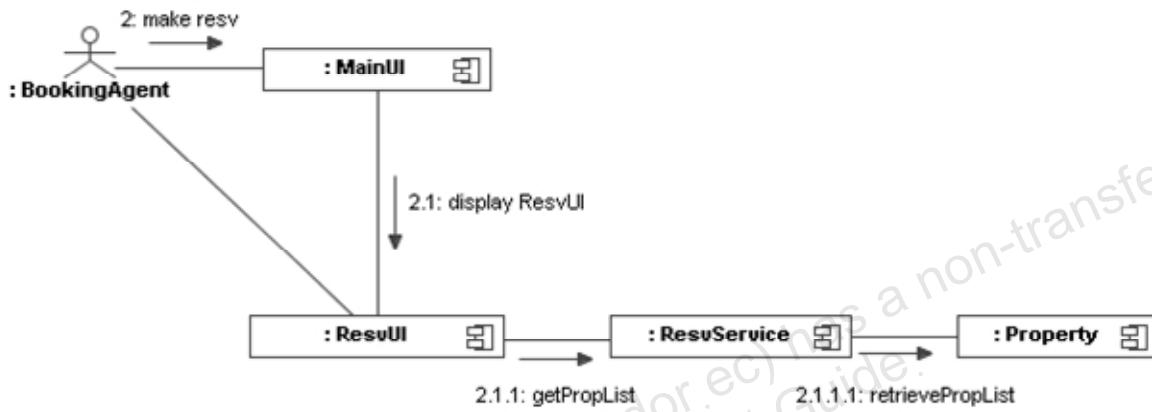
UML Interaction diagrams might also be used to show the sequence of interactions that occur between:

- Systems
- Subsystems

Interaction diagrams are primarily used to show interaction between specific objects; however, they are useful for showing interactions between composite objects, that include subsystems and systems. For example, you can use Interaction diagrams to show interactions between an ATM, the Bank that owns the ATM, and the card users Bank.

Communication Diagrams

A Communication diagram represents the objects of a system, their relationships, and the messages sent between objects.



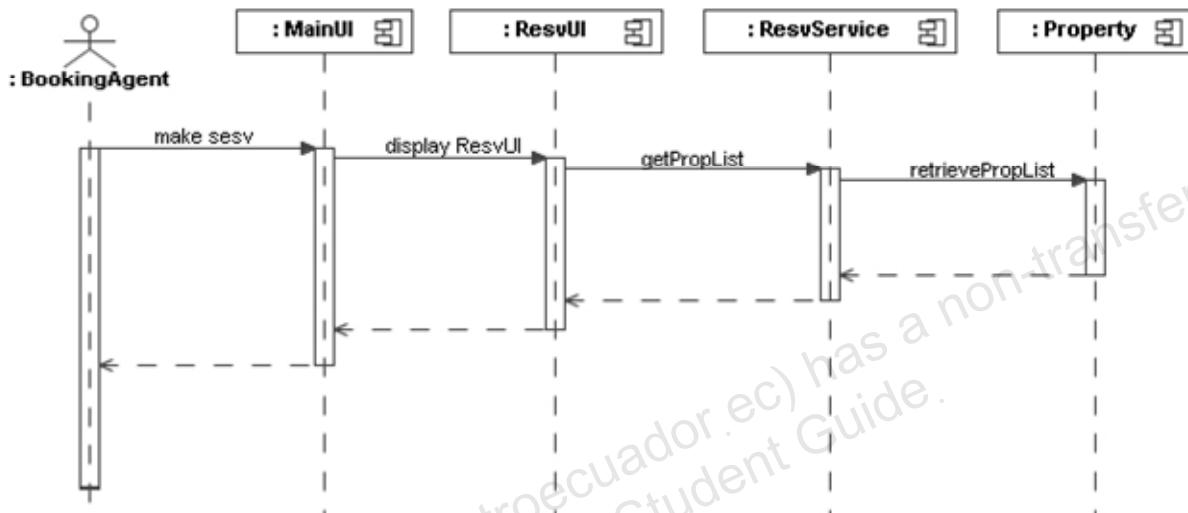
ORACLE

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

Communication diagrams, also called collaboration diagrams in UML specification 1.x, emphasize the data links among components in the interaction.

Sequence Diagrams

A Sequence diagram represents the objects of a system and the messages sent between objects organized in a time-ordered fashion.



ORACLE

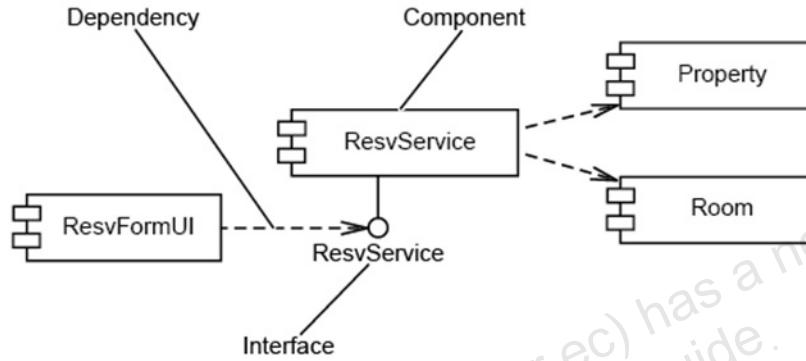
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

In an architecture, you can use one of these two types of diagrams to model the interaction and data communication among software components. Which type to choose is typically a matter of preference.

Sequence diagrams emphasize the exchange of data over time.

Component Diagrams

A Component diagram represents physical, software pieces of a system and their relationships.



ORACLE

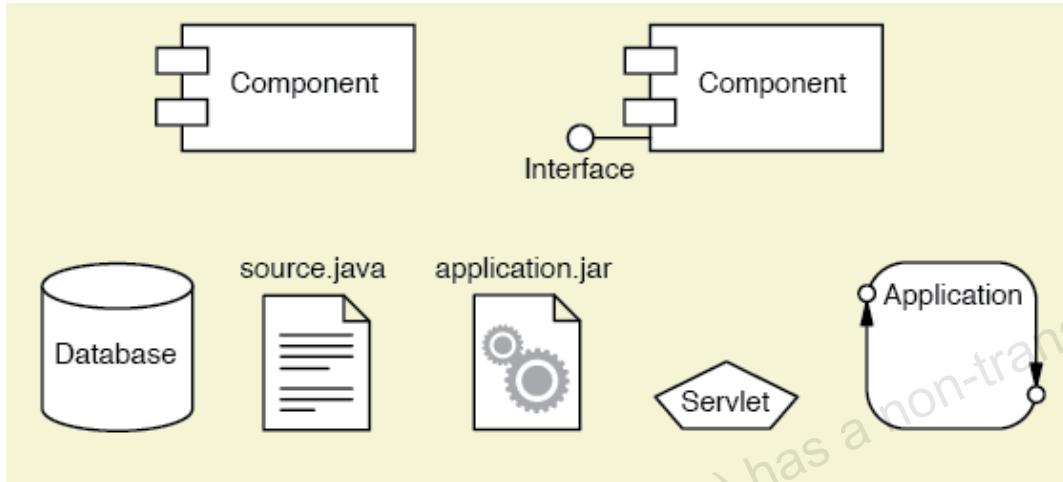
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

A component diagram shows the major software components of a system and how they can be integrated. Component diagrams can contain non-object-oriented software components, such as legacy procedural code and Web documents. When one component (or object) collaborates with another component, this collaboration is illustrated with a dependency between the client component and the service component.

Note: The component notation in UML 2.0 was changed to what is shown in the slide. In your UML diagrams, you can use the UML 1.x style component notation, or customized graphical representations for components.

Component diagrams can be a good way to show how all of the object-oriented and non-object-oriented components fit together in your system. It is also a good way to describe the high-level software module structure of the system.

Types of Components



ORACLE

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

You can use a UML Component diagram in a wide variety of ways. A few characteristics of components and Component diagrams are:

- A component represents any software unit.
- A component can be any type of software, including nonexecutable software. Examples include JavaBean components, source code modules, HTML files, and so on.
- A component can be large and abstract.
- A component can be a complete system or subsystem. For example, a DBMS server can be considered a single component.
- A component can be small.
- A component can also be a single object; therefore, a class can represent a set of component objects.
- A component might have an interface that it exports as a service to other components.

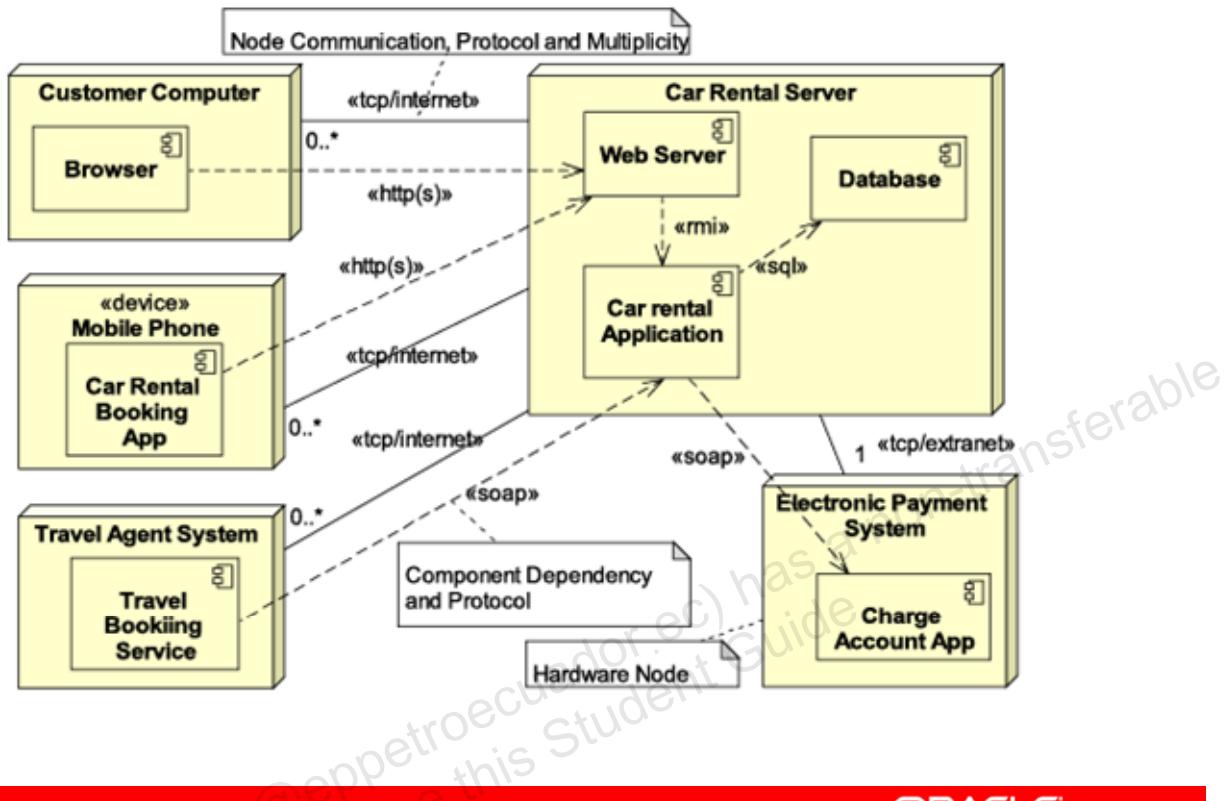
An interface can be conceptual or concrete. An abstract interface might be the communication protocol used by a large component; for example, structured query language can be considered the interface to communicate to a DBMS.

A concrete interface describes the set of methods implemented by some component. In Java technology, an interface declaration serves this purpose.

- A component can be a file, such as a source code file, an object file, a complete executable, a data file, HTML or media files, and so on.

Many things can be components. Not all of them will be executable. It is perfectly valid to think of an HTML file as a static component. Components can also be collections of other components. For example, a `.jar` file is a collection of class components.

Deployment Diagrams



A deployment diagram shows the hardware nodes and their layout in the system. The deployment diagram is useful for seeing how a distributed system is configured. In the architecture, software components might be displayed inside the hardware nodes to show how they will be deployed. An example of a deployment diagram is shown in the slide.

A Deployment diagrams contain the following elements:

Hardware nodes can represent any type of physical hardware.

A hardware node is usually a computer, but it can also be any physical device that communicates with a computer network. Such devices include printers, scanners, personal digital assistant (PDA), cell phone, network routers, firewalls, and so on.

Links between hardware nodes indicate connectivity and can include the communication protocol used between the nodes.

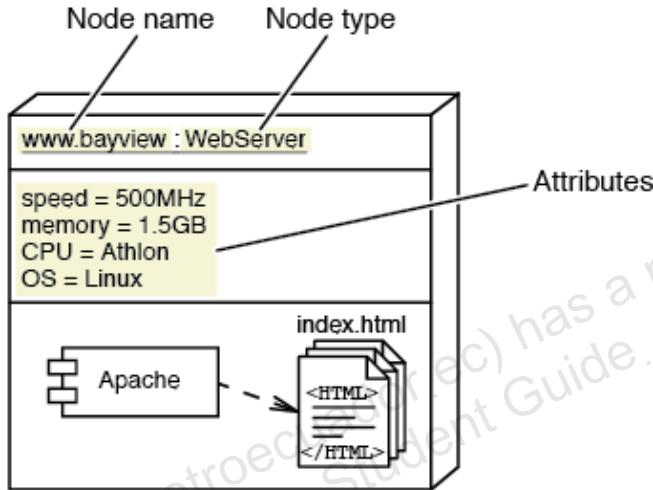
A Deployment diagram represents a graph of links that show the topology of a real computer network. These links are shown as solid lines with no arrows. There is usually a stereotype label on the link, that specifies the communication protocol used between the nodes.

Software components are placed within hardware nodes to show the distribution of the software across the network.

A Deployment diagram can show how to deploy software components onto the hardware nodes of the system solution. These software components are usually large-scale components, such as complete executable applications and class libraries.

Types of Deployment Diagrams

- A *descriptor* Deployment diagram shows the fundamental hardware configuration.
- An *instance* Deployment diagram shows the hardware configuration and includes specific hardware decisions.



ORACLE

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

Deployment diagrams come in two forms:

A descriptor Deployment diagram shows the fundamental hardware configuration.

This is a conceptual diagram. The hardware nodes in this form are meant to be thought of as typical pieces of hardware. On page A-28 is an example of a descriptor Deployment diagram.

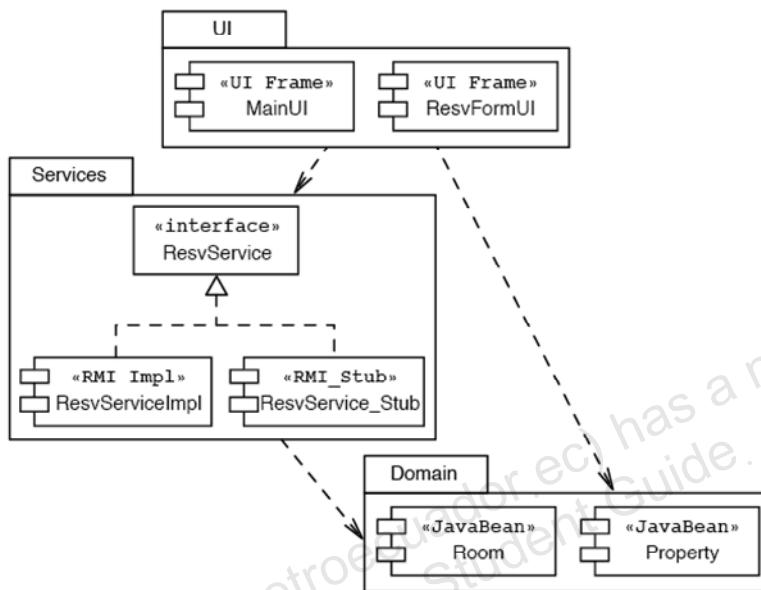
An instance Deployment diagram shows the hardware configuration and includes specific hardware decisions.

This form represents an actual or planned hardware topology. The hardware nodes in this type of Deployment diagram uses notation similar to Object nodes in which the name text is underlined and the full name includes the name of the node and the type of hardware for that node.

Notice that you can specify attributes of a hardware node. Example attributes include the speed and memory capacity of the computer.

Package Diagrams

A UML Package diagram shows dependencies between packages.



ORACLE

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

Use package diagrams to describe the organization of systems, technologies and components. Package diagrams allow you to group other constructs created in the model. Package diagrams are used to represent the organization of software components based on packages or namespaces. At the architecture level, package diagrams can be used to describe how software and hardware products, technologies, and application components are organized in a layered approach. The slide shows an example of such a package diagram.

Lesson Agenda

- Distinguish between architecture and design
- Introduction to architectural patterns
- Architecture deliverable artifacts
- Architecture modeling using UML
- **Architecture workflow**
- What Is an Enterprise Architecture framework?

ORACLE

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

Architecture Workflow Steps

1. Select an architecture type for the system.
2. Create a detailed Deployment diagram for the architecturally significant use cases.
3. Refine the Architecture model to satisfy the NFRs.
4. Create and test the Architecture baseline.
5. Document the technology choices in a tiers Package diagram.
6. Create an Architecture template from the final, detailed Deployment diagram.



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

The Architecture workflow is difficult to describe because the tasks involved vary greatly depending on which NFRs the architect focuses. This course presents a simplified workflow:

Select an architecture type for the system.

An architect selects the architecture type that best satisfies the high-level constraints and NFRs. An architecture type refers to a small set of abstract architectures, such as standalone, client/server, App-centric n-tier, web-centric n-tier, and enterprise n-tier. This step is discussed later in the lesson. Based on the selected architecture type, a high-level Deployment diagram is created to show the distribution of the top-level system components on each hardware node.

Create a detailed Deployment diagram for the architecturally significant use cases.

The architect creates a detailed Deployment diagram that shows the main components necessary to support the architecturally significant use cases. To create this diagram, the architect must create a Design model of the architecturally significant use cases and then place these abstract components into an infrastructure that supports the NFRs. Therefore, the architect must do some Design work to create the detailed Deployment diagram.

Refine the Architecture model to satisfy the NFRs.

The architect uses Architectural patterns to transform the high-level architecture type into a robust hardware topology that supports the NFRs. This workflow activity is covered in later lessons.

Create and test the Architecture baseline.

The architect implements the architecturally significant use cases in an evolutionary prototype. When all architecturally significant use cases have been developed, the evolutionary prototype is called the Architecture baseline. The Architecture baseline represents the version of the system solution that manages all risks. The Architecture baseline is tested to verify that the selected systemic qualities have been satisfied. The Architecture baseline is refined by applying additional patterns to satisfy the systemic qualities.

Document the technology choices in a tiers Package diagram.

For each tier, the architect identifies the appropriate technologies to be used in that tier.

Create an Architecture template from the final, detailed Deployment diagram.

The Architecture template is an abstract version of the detailed Deployment diagram in which the Design components are identified within the structure of the architecture (and infrastructure) components. You can use this template to guide the development team during the Construction phase.

This is not a formal artifact of the Architecture workflow. However, you will use this template in the course.

Select the Architecture Type

- Selecting the architecture type for a system provides the development team a vision of the top-level software and hardware structure of the system.
- This is recorded in high-level Deployment and Component diagrams.
- The architecture to use depends on many factors.



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

The architecture to use depends on many factors, including:

The platform constraints in the system requirements

For example, operating system, application platform, and hardware to use when deploying the system might constrain the architecture type.

The modes of user interaction

How a given actor will use the system might constrain the architecture. There are many modes of interaction; some examples are GUI, web, voice, direct manipulation, handheld devices, and so on.

The persistence mechanism

The type of persistent storage will guide architecture decisions. These issues are described in the lesson titled “Developing an Architecture for the Integration and Resource Tiers.”

Data and transactional integrity

Similarly, data and transactional integrity will guide architecture decisions. These issues are covered in later lessons.

Type of Software Architectures

There are hundreds of successful software architectures. A few common types are:

- Standalone applications
- Client/Server (2-tier) applications
- N-tier applications
- Web-centric n-tier applications
- Enterprise n-tier applications

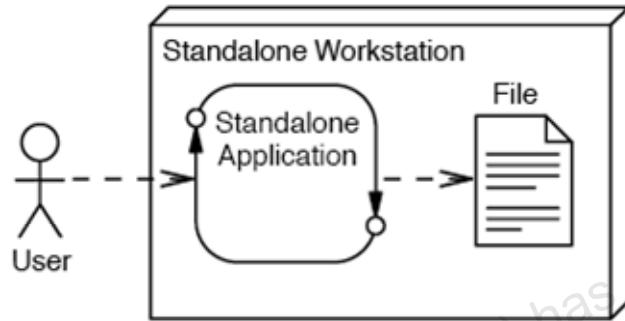


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using the selected architecture type, the architect provides an essential artifact of the Architecture workflow: the high-level Deployment diagram. This artifact shows the fundamental hardware topology as well as the top-level components that are hosted by each hardware node. These top-level components are usually independent applications.

Standalone Applications

Standalone applications exist as a single application component hosted by the user's workstation.



ORACLE

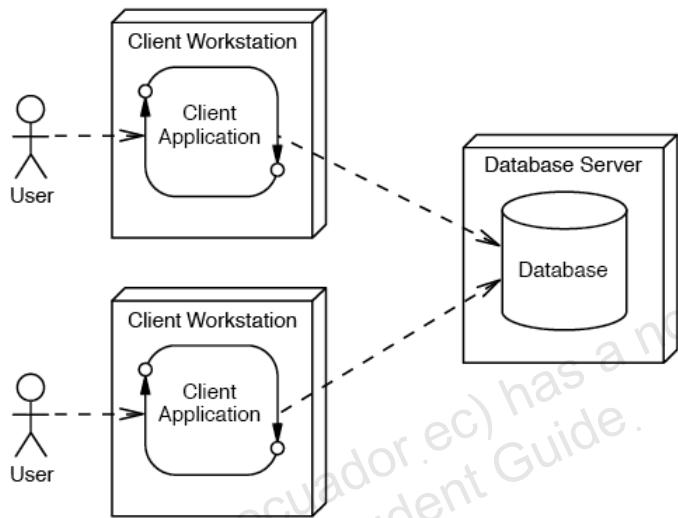
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Standalone applications are very common. Examples include word processors, graphics applications, text editors, and so on. These types of applications do not access a central data store (such as a DBMS) and do not participate in network communication.

This architecture type requires updating every user's workstation when the application changes. This is called the deployment problem. For most standalone applications this is not a serious problem. The user decides when to upgrade. It is only a problem when the user must share application-generated files with other users. Different versions of the software might have incompatible file formats.

Client/Server (2-Tier) Applications

Client/server applications are client applications hosted by the user's workstation that communicate to a common data store.



ORACLE

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

There are two subtypes of client/server applications: thin client and thick client. A thin client refers to a UI that does not contain any business logic. In a client/server application a thin client is usually a forms-based front-end to a database. A thick client refers to a UI that provides business logic on the client tier. Thick clients can provide a more rich user experience. In the first case, the data store manages the data and transactional integrity rules. In the second case, the client application provides this management.

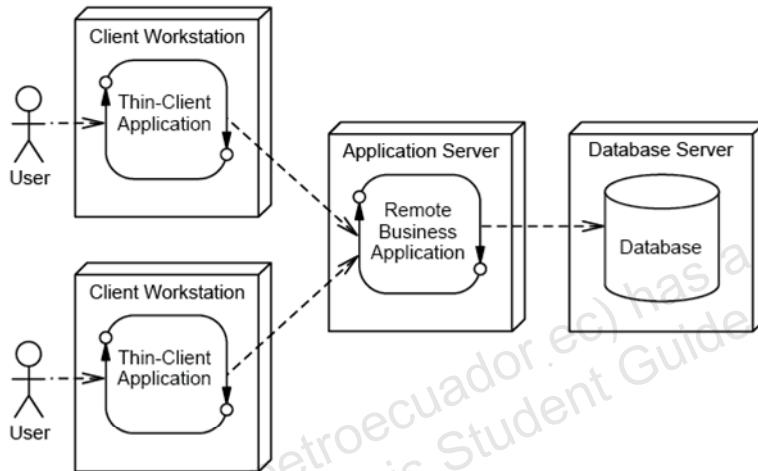
The following website provides a definition of thin client:

<http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?thin+client>.

This architecture type also suffers from the deployment problem. This problem is significant in this situation, because every client must have the latest version of the client application when the data store is upgraded. Every user must be using the same version of the software or data corruption problems might occur.

Application-centric N-Tier Applications

- Application-centric applications partitions the business components onto a separate application server.
- The client workstations host a thin-client application that uses the services of the application server.



ORACLE®

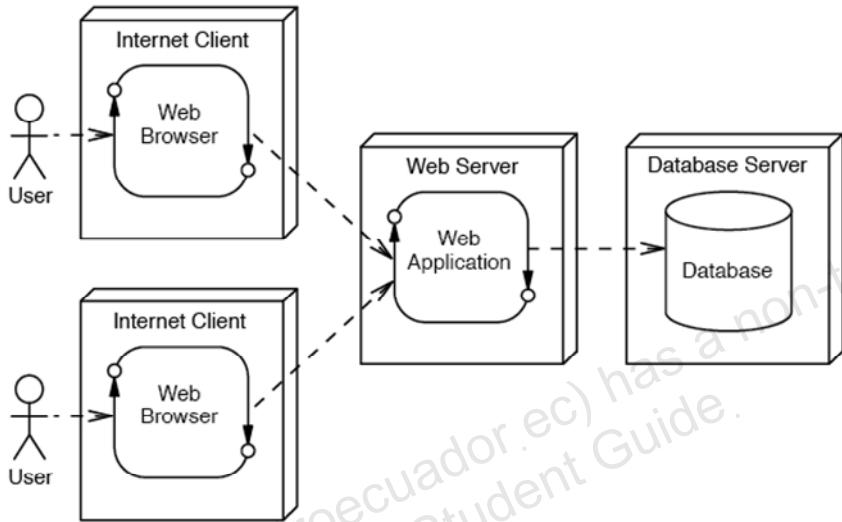
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

In this configuration, the application server manages the data integrity and performs all data store operations.

By separating the client applications from the business service application, these two high-level components can be upgraded independently. For example, if a bug in the business application is fixed, then only the application server needs to be upgraded.

Web-centric N-Tier Applications

Web-centric applications provide access to the application through a web browser on the client workstation.



ORACLE

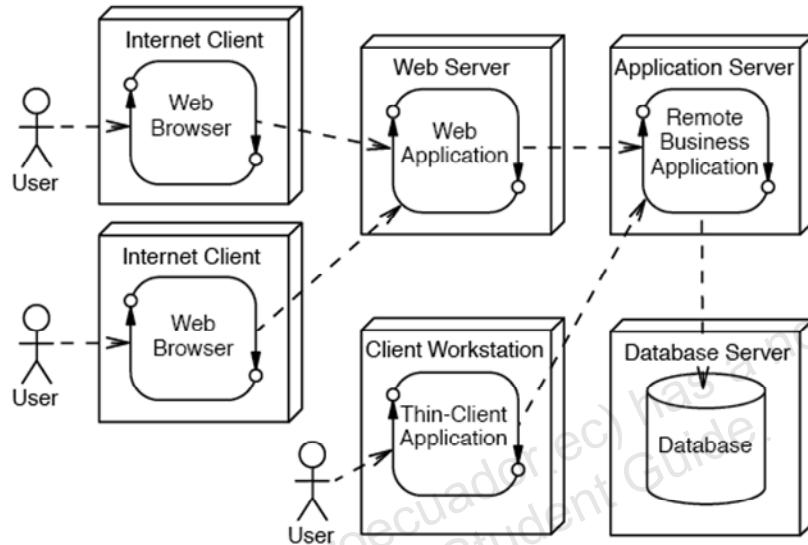
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

There is one major drawback to this architecture. If users within your company need to use the application, they must use the Web application. This is a security risk because you are making internal business functions accessible to the Internet. Alternatively, there could be thick-client applications that access the database directly in a client/server architecture. This could cause data integrity problems if the business logic in the web application is different than the business logic in the intranet client applications.

The deployment problem is virtually invisible to the end users. After the web application is upgraded, all users will immediately see the new application.

Enterprise-centric N-Tier Applications

Enterprise applications provide a hybrid of web-centric and application-centric architectures.



ORACLE

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

This architecture type provides the greatest flexibility by separating each logical tier onto a separate machine. The data integrity and security problems with the web-centric architecture are resolved by having access to remote business services.

The main drawback of this architecture type is the complexity of managing all of the disparate software components on multiple workstations and servers.

Creating the Detailed Deployment Diagram

To create a Detailed Deployment diagram:

1. Design the components for the architecturally significant use cases.
2. Place Design components into the Architecture model.
3. Draw the detailed Deployment diagram from the merger of the Design and Infrastructure components.
4. The merger of the Design components with the infrastructure is then drawn using a detailed Deployment diagram.



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

This is an essential part of the Architecture model because it shows the actual components that need to be developed.

The Detailed Deployment diagram also acts as a basis for the Architecture template which the development team will use to complete the construction of the system after the architecture team has built the Architecture baseline.

Lesson Agenda

- Distinguish between architecture and design
- Introduction to architectural patterns
- Architecture deliverable artifacts
- Architecture modeling using UML
- Architecture workflow
- What Is an Enterprise Architecture framework?



ORACLE

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

What Is an Enterprise Architecture Framework?

An Enterprise Architecture framework:

- Simplifies the process and guides architects through all areas of architecture development.
- Provides a toolkit that can be used to develop a wide range of differing architectures.
- Provides a collection of:
 - Best practices
 - Vocabulary
 - Standards
 - Tools
 - Processes
 - Templates
- Should describe a method for using the framework.



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

Because the discipline of Enterprise engineering and Enterprise Architecture is so broad, and because enterprises can be large and complex, the models associated with the discipline also tend to be large and complex. To manage this scale and complexity, an Architecture Framework provides tools and methods that can bring the task into focus and allow valuable artifacts to be produced when they are most needed.

Architecture Frameworks are commonly used in Information technology and Information system governance. An organization may wish to mandate that certain models be produced before a system design can be approved. Similarly, they may wish to specify certain views be used in the documentation of procured systems - the U.S. Department of Defense stipulates that specific DoDAF views be provided by equipment suppliers for capital project above a certain value.

Popular Enterprise Architecture Frameworks

- Zachman Enterprise Framework
- The Open Group Architecture Framework (TOGAF)
- OMB Federal Enterprise Architecture (FEA)
- The Gartner Methodology
- Oracle Enterprise Architecture Framework (OEAF)



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

An Enterprise Architecture Framework (EAF) maps all of the software development processes within the enterprise and how they relate and interact to fulfill the enterprise's mission. It provides organizations with the ability to understand and analyze weaknesses or inconsistencies to be identified and addressed. There are a number of already established EAF in use today; some of these frameworks were developed for very specific areas, whereas others have broader functionality.

There are a number of architectures and architectural frameworks in use today. Though they may overlap or address similar views, frameworks also have been designed to address specific needs or concerns. These frameworks differ by the stakeholders they address and the issues that concern their "world". These issues or "building blocks" represent methods, common vocabulary, standards, and tools that provide a means to implement and integrate the building blocks. In addition, government, commercial, and subcategories of each of these may require certain protocols to follow.

Comparison of EA Frameworks

| Criteria | Zachman | TOGAF | FEA | Gartner |
|--------------------------|---------|-------|-----|---------|
| Taxonomy Completeness | 4 | 2 | 2 | 1 |
| Process Completeness | 1 | 4 | 2 | 3 |
| Reference Model Guidance | 1 | 3 | 4 | 1 |
| Practice Guidance | 1 | 2 | 2 | 4 |
| Maturity Model | 1 | 1 | 3 | 2 |
| Business Focus | 1 | 2 | 1 | 4 |
| Governance Guidance | 1 | 2 | 3 | 3 |
| Partitioning Guidance | 1 | 2 | 4 | 3 |
| Prescriptive Catalog | 1 | 2 | 4 | 2 |
| Vendor Neutrality | 2 | 4 | 3 | 1 |
| Information Availability | 2 | 4 | 2 | 1 |
| Time to Value | 1 | 3 | 1 | 4 |

Rating Scale:

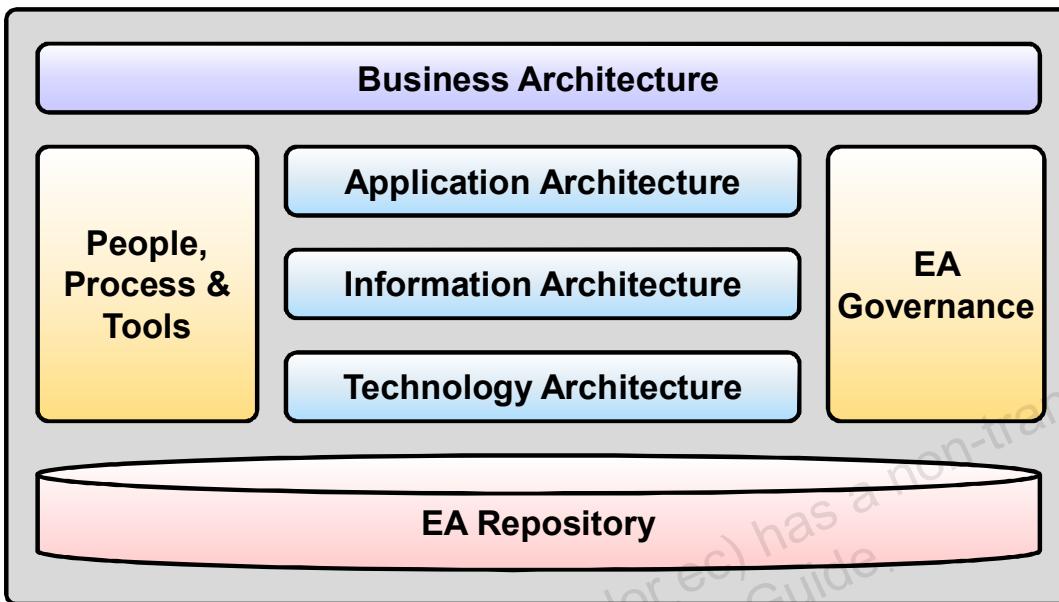
1. Very Poor
2. Inadequate
3. Acceptable
4. Very Good

ORACLE®

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

Source: Comparison of the Top Four Enterprise Architecture Methodologies
by Roger Sessions, CTO of ObjectWatch

Oracle Enterprise Architecture Framework



ORACLE®

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

To align the strategies of business with IT, a new approach for managing IT has been developed called Enterprise Architecture. Just as architecture provides a blueprint for constructing a building, Enterprise Architecture provides a blueprint and roadmap for aligning business strategy with IT.

Enterprise Architecture (EA) is a method and an organizing principle that aligns functional business objectives and strategies with an IT strategy and execution plan. The Enterprise Architecture provides a guide to direct the evolution and transformation of enterprises with technology. This in turn makes IT a more strategic asset for successfully implementing a modern business strategy.

The Oracle Enterprise Architecture Framework helps Oracle to collaboratively work with customers in developing strategic roadmaps and architecture solutions that enable business and IT alignment. Oracle emphasizes a “just enough” and “just in time” practical approach to Enterprise Architecture, which may be used standalone or as a complement to a customer’s selected EA methodology. By focusing on business results and leveraging Oracle’s unique EA assets and reference architectures, the Oracle Enterprise Architecture Framework can be employed to efficiently create architecture roadmaps for implementing business-driven enterprise solutions.

The Oracle Architectural Development Process



ORACLE

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

- Describes a streamlined process for developing mission-critical enterprise applications
- Incorporates industrial best practices
- Uses a practical approach to align IT and the business
- Provides a generic, base process for developing architectures as part of OEAFF
- Contains:
 - Six high-level phases
 - Tasks
 - Deliverables
 - Artifacts created in each phase

TOGAF to OEAf Mapping

| TOGAF | OEAF | OEAF Area |
|--|---|-----------|
| Part II: Architecture Development Method (ADM) | Oracle Architecture Development Process(OADP) | |
| Part III: ADM Guidelines & Techniques | <ul style="list-style-type: none"> • Oracle Architecture Development Process(OADP) • Applied EA Processes | |
| Part IV: Arch Content Framework – Content MetaModel ◦ Bus Arch ◦ IS Data Arch ◦ IS App Arch ◦ Tech Arch | OEA Content MetaModel 90% Fully Aligned <ul style="list-style-type: none"> • Bus Arch • Info Arch • App Arch • Tech Arch | |
| Part V: Enterprise Continuum: Arch Repository | EA Repository | |
| Part VI: Reference Models & TRM | Technology Architecture (Note: Oracle Artifacts are stored in the EA Repository) | |
| Part V: Architecture Capability Framework | EA Governance | |



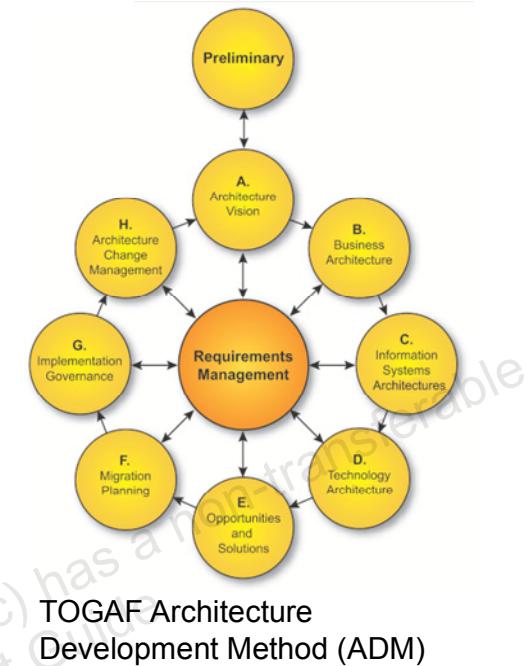
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

TOGAF is an industry standard architecture framework that may be used freely by any organization wishing to develop an information systems architecture for use within that organization. TOGAF has been developed and continuously evolved since the mid-90s by representatives of some of the world's leading IT customer and vendor organizations, working in The Open Group's Architecture Forum.

Oracle's OEAf is based on TOGAF and FEA.

OADP to TOGAF ADM Mapping

| Oracle Arch Dev Process | TOGAF ADM |
|-------------------------|--|
| Architecture Vision | Preliminary A. Architecture Vision B. Business Arch |
| Current State Arch | C. Info Sys Arch D. Tech Arch |
| Future State Arch | C. Info Sys Arch D. Tech Arch E. Opportunities & Solutions |
| Strategic Roadmap | E. Opportunities & Solutions F. Migration Planning |
| EA Governance | G. Implementation Governance H. Architecture Change Mgmt |
| Business Case | |
| EA Repository | Requirements Mgmt |



TOGAF Architecture Development Method (ADM)

ORACLE

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

The TOGAF ADM is the result of continuous contributions from a large number of architecture practitioners. It describes a method for developing an enterprise architecture, and forms the core of TOGAF. It integrates elements of TOGAF described in this document as well as other available architectural assets, to meet the business and IT needs of an organization.

Additional Resources

- The Open Group Adoption Strategies Working Group. *World-Class Enterprise Architecture*, 2010.
- The Object Management Group. Unified Modeling Language (UML), Version 2.2
<http://www.omg.org/technology/documents/formal/uml.htm>.
- *Architectural Strategies for Managing the Information Explosion*, October 2009.
<http://www.oracle.com/technetwork/topics/entarch/whatsnew/managing-the-info-explosion-131797.pdf>
- *J2EE Patterns Catalog*,
<http://www.oracle.com/technetwork/java/catalog-137601.html>



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

Kruchten, Philippe, *The 4+1 View Model of Architecture*, *IEEE Software* 12(6), 1995.

Hohmann, Luc. *Beyond Software Architecture: Creating and Sustaining Winning Solutions*. Addison-Wesley Professional, 2003.

Oracle Architecture Development Process: The Art of Realizing Your Future State Architecture [October 2009] <http://www.oracle.com/technetwork/topics/entarch/articles/oracle-ea-process-167294.pdf>

Zachman Framework: <http://www.zifa.com/framework.html>

TOGAF Version 9 <http://www.opengroup.org/togaf/>

Enterprise Architecture, <http://medianetwork.oracle.com/media/show/15646>

The Oracle Enterprise Architecture Framework [October 2009]
<http://www.oracle.com/technetwork/topics/entarch/articles/oracle-ea-framework-167292.pdf>

Quiz

The characteristics of an Architect and a Designer are often blurred. However, given the structure of an organization where there is a clearly defined role for an Architect, which of the following are the purview of the Architect? Choose all that apply.

- a. Specification of JPA components that provide ORM for database tables
- b. Specification of a load-balancer
- c. Focus on the number of users that can access an EJB component simultaneously
- d. Focus on the design of the form a user accesses to provide personal data



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

Answer: b, c

Architects generally deal with the technical requirements.

Quiz

Which of the following is considered a good practice when constructing a system architecture? Choose all that apply.

- a. Separation of Concerns
- b. Use container frameworks
- c. Use simple and clear component interfaces
- d. Ensure you use a strongly-typed programming language

 ORACLE

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

Answer: a, b, c

Whether or not you use even an OO language should not have a bearing on the architecture.

Quiz

The primary purpose of the Architectural Blueprint is to address Quality of Service requirements.

- a. True
- b. False



ORACLE

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

Answer: a

Quiz

The architectural prototype is a model of the architectural blueprint. To create the prototype, the architect should: (choose all that apply)

- a. Develop the graphical user interfaces
- b. Create a physical implementation of a skeleton system
- c. Deploy the prototype in a simulated test environment
- d. Create an end-to-end implementation of significant use-cases



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

Answer: b, d

Quiz

Which of the following statements best describes the benefit of using an Architectural Framework?

- a. They provide a vendor-neutral solution
- b. They provide tools and methods to manage complexity
- c. They address the nonfunctional requirements
- d. They prevent scope-creep by constraining system requirements



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

Answer: b

Frameworks are by design vendor neutral.

Summary

In this lesson, you should have learned how to:

- Distinguish between architecture and design
- Describe the use of architectural patterns
- Describe the architecture workflow
- Describe the diagrams of the key architecture views
- Select the architecture type
- Describe common enterprise architecture frameworks



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

Practice 2 Overview: Model Initial Architecture

Create UML Component and Deployment diagrams modeling the case study exercise architecture.



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

In this practice you will use UML Component and Deployment diagrams to model the architecture for the case study. You will start with high level models and then add detail to create the basic architecture that will be iterated and added to throughout the course.

Developing a Security Architecture

3

ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Analyze the impact of security in distributed computing
- Describe the security services in Java technology
- Define security requirements for web services

Discussion Questions

- How does the Java EE technology address security issues?
- What, if any, additional considerations should you take into account for web service security?



ORACLE

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

Lesson Agenda

- Security technologies and regulatory constraints
- Impact of security in distributed computing
- Security in the Java EE technology
- Web services security

ORACLE

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

Types of Access Control

- Access control addresses the idea of who has access to what. The “what” can be systems or sets of information or types of operations to be executed.
- Refers to control over access to resources after a user’s credentials and identity have been authenticated and access granted.
- Example: a particular user, or group of users, might only be permitted access to certain files after logging into a system, while simultaneously being denied access to all other resources.
- Three types or models:
 - Mandatory Access Control (MAC)
 - Discretionary Access Control (DAC)
 - Role-based Access Control (RBAC)

ORACLE

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

Mandatory Access Control

Mandatory access control (MAC) is a system-enforced access control mechanism that is based on label relationships. The system associates a sensitivity label with all processes that are created to execute programs. MAC policy uses this label in access control decisions. In general, processes cannot store information or communicate with other processes, unless the label of the destination is equal to the label of the process. MAC policy permits processes to read data from objects at the same label or from objects at a lower label. However, the administrator can create a labeled environment in which few lower-level objects or no lower-level objects are available.

For example: If the user's credentials match the MAC security label properties of the object access is allowed. It is important to note that both the classification and categories must match. A user with top secret classification, for example, cannot access a resource if they are not also a member of one of the required categories for that object.

By default, MAC policy is invisible to you. Regular users cannot see objects unless they have MAC access to those objects. In all cases, users cannot take any action that is contrary to MAC policy.

Discretionary Access Control

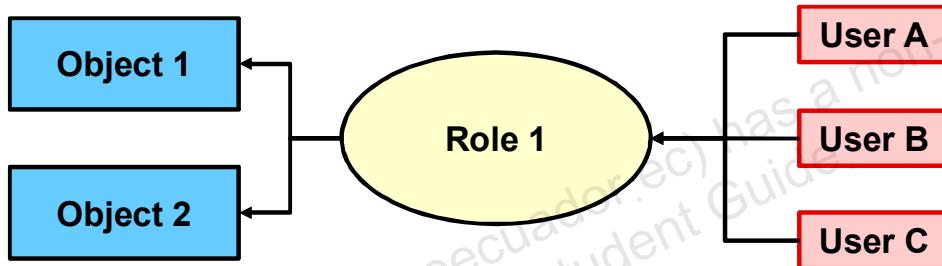
Discretionary access control (DAC) is a software mechanism for controlling user access to files and directories. DAC leaves setting protections for files and directories to the owner's discretion. The two forms of DAC are UNIX permission bits and access control lists (ACLs).

Permission bits let the owner set read, write, and execute protection by owner, group, and other users. In traditional UNIX systems, the superuser or root user can override DAC protection. With Trusted Extensions software, the ability to override DAC is permitted for administrators and authorized users only. ACLs provide a finer granularity of access control. ACLs enable owners to specify separate permissions for specific users and specific groups. For more information, see Chapter 7, Controlling Access to Files (Tasks), in System Administration Guide: Security Services.

Note: Discretionary Access Control provides a much more flexible environment than Mandatory Access Control but also increases the risk that data will be made accessible to users that should not necessarily be given access.

Role-based Access Control (RBAC)

- Control decisions are based on the functions a user is allowed to perform within the organization.
- Users cannot pass permissions to others at their discretion.
- Corresponds to the organization structure and/or business process.
- Can be difficult to implement consistently.



ORACLE

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

RBAC is considered the newer approach to mandatory access control and discretionary access control. It is widely considered best practice. Enterprise RBAC must deal with the increased complexity of managing a sufficient number of roles and assigning adequate role membership inside of a heterogeneous IT infrastructure.

For example, within a hospital system, the role of doctor can include operations to perform diagnosis, prescribe medication, and order laboratory tests; and the role of researcher can be limited to gathering anonymous clinical information for studies.

Typically, lack of coordination, momentum or perseverance on the part of IT to define and adopt a central vision results in the implementation of several strategies over time.

Management must be involved to define and enforce a “top down” strategy and approach.

Security access is often perceived to be an IT/technical problem, when in reality it is most often an organizational/cultural one. The challenge of a successful RBAC implementation is to remain focused on the people and not the technology.

Here the soft skills of the enterprise architect come into play once again. Communication to and teaching middle management is required to gain their support. RBAC can seem complex and difficult for users to grasp the need. Therefore a concise and effective communication strategy must be developed.

Enterprise role management helps organizations achieve these goals by providing a single authoritative source for roles that determine user access to drive provisioning events based on RBAC. Increasingly, industry best practices recommend the use of an enterprise role management solution to simplify and organize user access control more effectively. When implemented correctly, enterprise role management and user provisioning solutions collectively deliver positive benefits to all three principles (confidentiality, integrity, and availability) of an information security program.

The best way to leverage the benefits of an enterprise role management solution is to consider it as a component of an enterprise security architecture that includes complementary services such as access control, identity administration, directory, audit and compliance, and system management. By adding role-based access control to the enterprise security architecture, organizations can quickly reap several benefits including consistent security across applications, implementation of the least privilege principle, and overall improved interoperability and manageability to control user access.

Invasions

- SQL injection
- Cross site scripting
- Social engineering
- Pretexting

Why can't I name my son:
"Robert"); DROP TABLE
Students;--
?



ORACLE

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

SQL Injection is a way to attack the data in a database through a firewall protecting it. It is a method by which the parameters of a web-based application are modified in order to change the SQL statements that are passed to a database to return data. For example, by adding a single quote (') to the parameters, it is possible to cause a second query to be executed with the first. An attack against a database using SQL Injection could be motivated by two primary objectives: To steal data from a database from which the data should not normally be available, or to obtain system configuration data that would allow an attack profile to be built. One example of the latter would be obtaining all of the database password hashes so that passwords can be brute-forced. To gain access to an organization's host computers via the machine hosting the database. This can be done using package procedures and 3GL language extensions that allow O/S access.

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications that enables malicious attackers to inject client-side script into web pages viewed by other users. An exploited cross-site scripting vulnerability can be used by attackers to bypass access controls such as the same origin policy. Cross-site scripting carried out on websites were roughly 80% of all security vulnerabilities documented by Symantec as of 2007. Their impact may range from a petty nuisance to a significant security risk, depending on the sensitivity of the data handled by the vulnerable site, and the nature of any security mitigations implemented by the site's owner.

Social engineering is the act of manipulating people into performing actions or divulging confidential information, rather than by breaking in or using technical cracking techniques; essentially a fancier, more technical way of lying. While similar to a confidence trick or simple fraud, the term typically applies to trickery or deception for the purpose of information gathering, fraud, or computer system access; in most cases the attacker never comes face-to-face with the victim.

Pretexting: An invented scenario is created and used to target the victim in a manner that increases the likelihood they will divulge information or perform activities that would be unlikely under ordinary circumstances. An extreme example is the scenario perpetrated on Michael Douglas's character in the movie *The Game*. Here he is fooled into thinking his accounts have been broken into and uses a cell phone provided by a confederate to call his bank and he inadvertently divulges his password information. Pretexting is more directed at executives than simple social engineering.

Regulatory Constraints

- HIPAA
- Sarbanes-Oxley
 - “The thing to remember about SOX is that it is primarily focused on the accuracy of financial reporting data. IT security is important under SOX only to the extent that it enhances the reliability and integrity of that reporting.”
- *Mark Rasch*
- Removing business borders requires an environment of increased information transparency, risk visibility, and flexible business processes.



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

The enterprise must address the trade-offs between increasing regulatory issues and mandates, increasing interest in enterprise risk and the financial challenge to support these efforts

IT must determine how applications and an integrated network architectural approach can be responsive across the wide range of organizations and geographically distributed business services.

Lesson Agenda

- Security technologies and regulatory constraints
- Impact of security in distributed computing
- Security in the Java EE technology
- Web services security



ORACLE

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

Impact of Security

- Designing a secure distributed system requires consideration in three inter-related areas:
 - Network services
 - Hosting environment
 - Applications
- The security level of the system is determined by the weakest link of these three components.



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

From a technical perspective, designing a secure distributed system requires consideration in three inter-related areas: network services, hosting environment, and applications. The security level of the system is determined by the weakest link of these three components. Therefore, creating a security architecture involves applying best practices and guidelines to all three areas.

Securing the Network Services

- The network service level security is typically achieved using network devices.
- On the other hand, the network level security service does not protect against malicious attacks targeting publicly accessible protocols and services.



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

The network-service level security is typically achieved with devices, such as firewalls, routers, and switches. Routers buffer and transfer data packets between two networks. Firewalls block protocols and ports that the application does not use. Firewalls can also enforce secure network traffic by providing application-specific filtering to block malicious communications. Switches are used to separate network segments.

Additional network devices for security purposes include virtual private networks, SSL/cryptographic accelerators, intrusion detection systems, and access control lists. The network-level security typically provides the following protection:

- Protect network resources from IP connection attacks
- Protect access to unauthorized ports and protocols by packet filtering

On the other hand, the network level service does not protect malicious attacks targeting publicly accessible protocols and services. For example, if the firewall allows Internet users to access a web server, the web server is vulnerable to attacks, such as malicious code injection, buffer overflow, and content-based denial-of-service.

Securing the Hosting Environment

- The hosting environment includes the operating system and the application component runtime environment.
- Securing the operating system requires hardening and minimization techniques to remove nonessential utilities and tools, disable unused services and ports, and apply security-specific patches and upgrades.
- Securing the application's runtime environment requires you to select the appropriate security providers.



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

Securing the application's runtime environment also involves patching and upgrading the relevant software, such as JDK and application servers. Furthermore, you also need to select the appropriate security provider products that can be easily plugged into the application server to meet the service-level requirement for security.

User provisioning helps organizations by centralizing and automating the management of user accounts and entitlements in organizations' information resources such as databases, directories, business applications, and email systems.

Securing Applications

- Many security breaches are related to application-specific deficiencies ranging from architecture, design to coding.
- In order to secure the application, the architect and the development team must rely on a proactive security approach throughout the project life cycle to create a sound security solution.



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

Examples of application-specific vulnerabilities include insufficient input validation, no account lockout, injection flaws, cross-site scripting (XSS), and weak password exploits.

An excellent resource for familiarizing yourself with common web application security flaws is the open source WebGoat. It is a J2EE web application that is implemented with many intentional bad practices. For more information on the WebGoat application, visit <http://www.owasp.org>.

To secure the application, the architect and the development team must rely on a proactive security approach throughout the project life cycle to create a sound security solution.

Common Security Principles

To create a security architecture, consider the following principles described in the following sections:

- Maintaining corporate security policies
- Self-preservation
- Defense in depth
- Least privilege
- Compartmentalization
- Proportionality



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

Maintaining Corporate Security Policies

- Every company should have a corporate security policy.
- This policy outlines the security issues of importance to the company.
- Be sure to include these goals in the architecture considerations for your system.
- If you do not incorporate these goals, the technologies or decisions can become self-serving, and the overall security of the company reduced.

ORACLE

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

Having outlined the essential goals, the security policy should identify known and likely threat vectors. That is, the policy should identify ways in which an attack might be made against the company. The policy attempts to identify the following information for each potential vector:

- The difficulty of the attack
- The cost to the company of a successful attack versus cost of implementation. Consider cost per breach or frequency.

The policy should first address easy attacks and those that result in extensive damage. Difficult attacks that are unlikely to result in much damage are not as critical to address as others.

Weigh each security measure that the company adopts for effectiveness against the costs it brings. These costs include:

- Direct cost of purchasing or writing the software
- Cost of hardware
- Cost of salaries for installation, configuration, and maintenance
- Indirect costs to the organization

Indirect costs might arise because the implementation of a security measure makes daily operations more difficult. For example, consider a company that blocks outgoing FTP services, with the expectation that the block prevents employees who might otherwise send confidential information to unauthorized parties. Unfortunately, this type of block is not effective because employees can still use email to send confidential information. Moreover, blocking outgoing FTP services also reduces the effectiveness of staff who have a legitimate use for FTP services. A better approach in this case might be to install login procedures. Then, if you suspect problems, there is some chance of identifying the culprit.

One of the most important features of a well-written security policy is that it allows you to determine whether something is worthwhile. All energy that is directed at security should have a purpose, and that purpose should not be circumvented by some other hole. For example, insisting that everyone has 20-character random passwords is not useful. In most cases because it encourages people to write down their passwords and attach them to their monitors. Similarly, password security policies are pointless if everyone knows the administrator's password.

A final point on security is that insurance might be the best way to address some issues. This is a business decision. The level of insurance available for security problems is changing rapidly, as are the costs and requirements for obtaining insurance. But in the end, the point of security is to protect the assets of the company and its shareholders. If insurance is the most cost-effective way to achieve this, then use insurance, not technology, as a solution.

Self-Preservation

- Self-preservation requires that system components and services be configured, used, and managed so that they are resistant to unauthorized external influence.
- Each component used in a network architecture design is configured for self-preservation. Fundamentally, this means that each component is configured to reduce its attack surface and potential for exposure by:
 - limiting access to management functions.
 - using secure protocols.
 - ensuring that access to services and functions is restricted to authorized parties.



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

When you create a security architecture, this principle requires you to:

- Reduce the attack surface of objects, as well as the potential for undue exposure.
- Ensure that objects are in a known and trustworthy state, are free of vulnerabilities, and are configured appropriately.
- Protect management and administrative interfaces from unauthorized access.
- Prefer the use of open and vetted protocols that implement strong authentication, confidentiality, and integrity protections.

Self-preservation can be thought of as component level security (for example, hosts, switches, routers, firewalls, load balancers, and so on).

Defense in Depth

- Defense in depth mandates the use of multiple, independent, and mutually reinforcing security controls to eliminate single points of security failure.
- An example of applying the defense-in-depth principle is to configure the two firewalls that defines the DMZ differently.
- By doing this, even if the first firewall is comprised, the second firewall is not immediately vulnerable to attacks.

ORACLE

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

Components and assemblies used in network architectures must support defense in depth through the use of multiple, complementary, and reinforcing security controls. The number and type of security controls used will vary based on the threat profile of the architecture, as well as organizational policies and preferences. The use of defense in depth helps to protect the environment against single points of security failure. Note that these security controls can be integrated into the components used to provide a service (such as hosts, switches, routers, load balancers, and so on) or they can be independent entities (such as firewalls, intrusion detection systems, and VPN concentrators).

Least Privilege

- Least privilege requires that you should provide just enough privileges to users and services for them to perform a particular task.
- The applications of this principle include:
 - Users can be given rights to access on certain systems, networks, and applications based on their organizational role.
 - Applications can be started and run as unprivileged accounts with limited access to the underlying operating system.
 - Services running on an operating system cannot establish outbound network connections to other systems.
 - Hosts residing on a given network can be restricted to communicate only with other hosts on the same network, and even then perhaps using only approved protocols.

ORACLE®

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

In today's systems, it is important to have different levels of access for different types of administration, and to record who performed what action. For this reason, the old UNIX model of an all-powerful root user no longer suffices. This is in accordance with the security principle of least privilege that demands that every program and user of the system operate using the smallest set of privileges necessary to complete the job. The solution in the Oracle Solaris operating system is the use of roles for specific administrative tasks. Users only can assume the roles for which they are authorized. Rights profiles are created and assigned to roles to specify which tasks a role can perform.

Starting with Oracle Solaris 10, the operating system implements a set of privileges that provide fine-grained control over the actions of processes. For each privileged operation, such as accessing raw network devices or mounting a file system, the kernel validates that the process performing the operation has been assigned the required privilege.

Compartmentalization

- The principle of compartmentalization requires you to group and isolate objects to reduce the impact of damage to the complete system if that object is compromised.
- The applications of this principle include:
 - Isolate or group communities of services, networks, systems, and users.
 - Provide a sandbox within which applications and services can be run.
 - Enforce isolation between users, data, and objects operating in different security roles, zones, or risk profiles.



ORACLE

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

Compartmentalization is defined as the act of separating something into distinct parts, categories, or compartments (source: <http://www.dictionary.com>). Compartmentalization is a very useful approach for keeping separate (or isolated) unrelated interfaces, services, data sets, systems, networks, and user communities. It is reminiscent of the old adage “a place for everything and everything in its place.” By viewing architecture in this way, it is possible to group and isolate objects in order to manage risk, including the potential for and impact of damage in the event that an object is compromised.

Examples include Solaris zones, Linux V-Server, BSD Jails.

Proportionality

- The principle of proportionality states that “information security controls must be proportionate to the risks of modification, denial of use, or the disclosure of information.”
- This is especially true when all of the various costs are considered: initial acquisition or purchase, customization and integration, ongoing support and maintenance, administration and troubleshooting, training and education, and so on.
- It is essential, therefore, that organizations work to achieve a balance between security and cost in how they architect, implement, and manage their environment.

ORACLE

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

Put another way, proportionality means that the cost of protecting a given asset should not exceed its value.

Security can be dramatically improved by understanding and leveraging a few basic security principles, such as those discussed in this section. Integrating security systemically throughout an environment offers organizations greater opportunities to not only manage their risk but also reduce their costs. For example, by bounding the selection and use of security controls (according to the principle of proportionality), organizations can better understand what assets are most critical and deserve the most protection. Similarly, it also helps organizations better select the baseline level of security that is required throughout their environment.

Costs and Benefits of Security Features

Consider the following when you decide whether to invest in a particular security feature:

- What is the value (to you and to a potential attacker) of what you are trying to protect?
- What is the cost to implement a proposed security feature? Consider developer time, software purchase costs, and usability impact.
- Does the proposed feature make sense in the big picture? There is little point buying a really expensive lock for your front door if you leave your windows open all of the time.



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

Implementing security is not a black and white problem. Instead of thinking about a system as secure or insecure, think about a system as more or less resilient to certain types of attack.

Resolving Trade-Off Items in Security

- Most security decisions involve discussions about whether a particular course of action, piece of software, and so on, is worthwhile.
- So, it is particularly important that you make a reasonable effort at quantifying the costs of security failure in meaningful business terms.
- When you can measure these risks and the cost of mitigating them, you can address security issues with the same cost-based comparison techniques that are used for addressing all risks.



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

One issue that arises in the selection of a security product is the choice between traditional and new protocols. Generally speaking, in security, a new protocol is equivalent to an untested protocol. Given the nature of security, accumulated years of public scrutiny are more important than any theoretical benefits that might be attributed to new protocols. Proprietary protocols are also a bad choice because they never receive public scrutiny, and history shows that proprietary protocols are almost always flawed and readily broken.

How Do You Know Your Users?

What are the pros and cons of each of these strategies?

- Passwords
- Central authentication facility: strong authentication
 - More choices of authentication mechanism are available
 - Many network authentication services support single sign-on.
 - Examples: Kerberos, Token cards, Biometrics, PKI
- Proxy Authentication and Authorization for multitier environments
- Single sign-on



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

Lesson Agenda

- Security technologies and regulatory constraints
- Impact of security in distributed computing
- **Security in the Java EE technology**
- Web services security



ORACLE

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

Examining Security in the Java EE Technology

- Built on the security capabilities of Java SE, the Java EE security model provides a flexible architecture to leverage existing security services.
- It allows the business logic of application components to be separated from the security logic.
- This improves the portability of the Java EE application, and it also encourages Java EE application server vendors to provide value-add security services.



ORACLE

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

Defining Java EE Security Terminology

The following terminology is used to describe the Java EE security model:

- Security service provider
- Principal
- Security attributes
- Security credential
- Security policy domain



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

- **Security Service Provider:** A built-in component or a third-party product that is used to enforce security policies. An example of a security service provider is an LDAP server configured for the Java EE application server.
- **Principal:** An entity (a user or an application) that can be authenticated by a security service provider. Once authenticated, a principal is identified by a unique name.
- **Security Attributes:** A set of security-specific data associated with a principal. You can use security attributes for multiple purposes, for example, to allow or deny access to protected resources.
- **Security Credential:** Information that is used by a security service provider to authenticate a principal.
- **Security Policy Domain:** A scope over which a common security policy is defined and enforced by the security service provider. Also known as Security Domain.

Authentication

- Java EE allows a principal's identity to be authenticated in all the tiers and components.
- The authentication mechanism is typically implemented by a third-party security service provider (sometimes called a security realm).

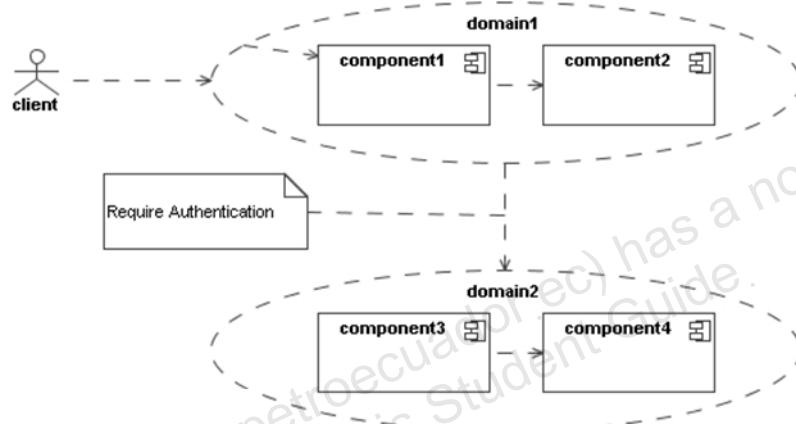


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The authentication mechanism is typically implemented by a third-party security service provider (sometimes called a security realm), such as an LDAP server, plugged into the application server as a Java Authentication and Authorization Service (JAAS) module. Authentication occurs when a client accesses the first component that is protected. When the client is authenticated, its identity and credentials can be propagated to subsequent accesses to the application.

Defining Protection Domains

- In Java EE, a protection domain is a set of components that are assumed or known to trust each other.
- Components within a protection domain do not need to authenticate to one another.
- When the interaction is across the boundary of a protection domain, authentication is required.



ORACLE

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

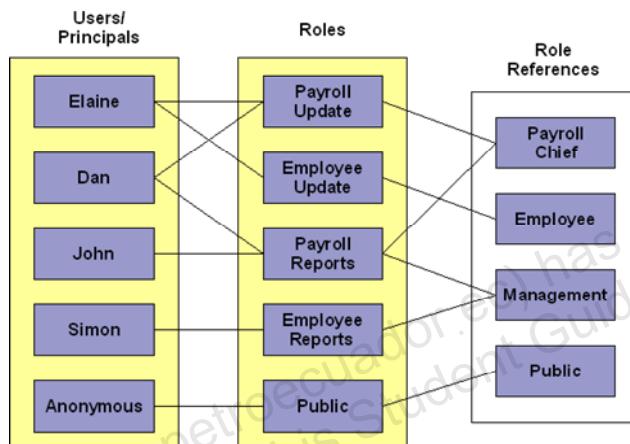
The slide illustrates the concept of a protection domain. When the client is authenticated by domain1, a client can access the components in this domain with re-authentication. However, if the access spans to domain2, then authentication is required.

A protection domain-based approach to security simplifies the issues of maintenance, authentication, and verification. However, there is a modest, but real, penalty to overall security. If an attack gains control of one component in the protection domain, then all components within the protection domain are compromised.

Keep in mind that the scope of a protection domain typically depends on the application server product, and the configuration of the application. Application server vendors can also provide single sign-on capabilities to extend the scope.

Authorization

- Java EE uses a role-based authorization model to control the access permissions.
- A role defines a logical grouping mechanism of users with respective to the target application.
- User and group information physically maintained by the security service provider can be mapped to roles.



ORACLE

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

A role is similar to a job function, but you determine it primarily in terms of need-to-know or need-to-perform. For example, you might decide that payroll clerk is a useful role. You can use this role to identify the information needed to perform the job and the functions within the system that are needed.

Once you have identified all of the roles, list the associated privileges. Then, document the roles and privileges so that an appropriate individual, often a member of the Human Resources department, can assign roles to individuals. One individual might be permitted any number of roles. The slide illustrates the concept of mapping users to roles.

Data Confidentiality and Integrity

- The Java EE platform facilitates secure communication by adopting transport layer integrity and confidentiality mechanisms based on SSL/TLS protocols.
- The SSL/TLS security properties are configured at the container level, and they can be applied to web components and EJB components, including their web services endpoints.
- At the time of assembly and deployment, components are declared to be protected for communication integrity or confidentiality.
- The component's deployment descriptor is used to represent this information.



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

Enabling SSL communication between EJB components and application clients is not standard, because it typically requires you to consult the application server vendor, and configure the vendor-specific deployment descriptors.

Selecting Cipher Suites

- SSL uses cipher suites to implement algorithms for key exchange, symmetric encryption, and message digest.
- For example, the cipher suite RSA_WITH_RC4_128_MD5 uses RSA for key exchange, RC4 with a 128-bit key for bulk encryption, and MD5 for message digest.
- During the SSL handshake, the strongest encryption accepted by the two peers is negotiated.
- To improve the security of SSL communication, you might want to configure the cipher suites supported by the server by removing some less secure cipher suites.

ORACLE

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

The protocol that you select should have the following performance characteristics:

- The protocol should be available to all parties that might need to take part in conversations.
- The protocol should provide an appropriate level of security.
- The protocol should impose a minimum impact upon the capacity and throughput of the system.

The decision about what level of security is appropriate for network communication is a difficult and continuously varying problem. In general, you need to have just enough security so that by the time someone breaks the protection on your message, the value of the message is gone. For example, a message detailing orders to troops in a battlefield loses its value after the troops have executed those maneuvers.

It is hard to know how long it might take to break any given encryption. Of course, the time it takes continues to decrease as computers get faster and as code breaking techniques improve.

The main, or perhaps only, cost of using longer keys is the time that it takes to encipher and decipher the messages. The time to encipher or decipher a message is related to the size of the message and the length of the key. A small message has a short encryption time, even if you are using a large key. So, if the messages are small, this time is not a major factor in the overall performance equation.

Declarative and Programmatic Security

- In the Java EE Technology, the separation of the business logic and security logic of application components is achieved by the container-based security model.
- The security service implemented and configured for the container can be injected to application components using a declarative or programmatic approach.

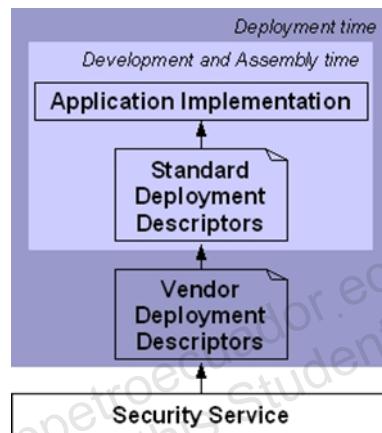


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

In Java EE, the component containers are responsible for providing application security. A container provides two types of security: declarative and programmatic.

Declarative Security

- Use annotations (preferred) or deployment descriptors.
- Standard deployment descriptors declare the logical security requirement.
- Vendor-specific deployment descriptors map security requirements to the physical security services.



ORACLE

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

In declarative security, an application's security structure, such as authentication requirement, authorization or access control, and security roles is defined outside of the application's code-level implementation. For Java EE applications, the deployment descriptor is the primary vehicle for declarative security. You can use the standard deployment descriptors to declare a logical security model for the application, and vendor-specific deployment descriptors to map the model to the physical security services provided by the application server product. In runtime, the security service is injected to the applications. This approach is illustrated in the slide.

Starting from Java EE 6, you can also use annotations, such as `javax.annotation.security.RolesAllowed`, defined in the JSR 250: Common Annotations for the Java Platform to declare the security features in the application. These annotations allow the application developers to provide the default security (and other) information about the application. At assemble and deployment time, you can use deployment descriptors to override these values.

Securing Enterprise Beans

Declarative security is the preferred method for EJBs. Declarative security expresses an application component's security requirements using either deployment descriptors or annotations. The presence of an annotation in the business method of an enterprise bean class that specifies method permissions is all that is needed for method protection and authentication in some situations.

Programmatic Security

- Used to implement a fine-grained security model that cannot be defined using declarative security
- Allows you to use APIs to build security-aware applications
- The standard Java EE API for programmatic security consists of two methods available for EJBs, HTTP Servlets, Web Service contexts and JAX-RS security contexts:
 - `isCallerInRole`
 - `getCallerPrincipal`
 - `isUserInRole`
 - `getUserPrincipal`



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

These methods allow components to make business logic decisions based on the security role of the caller or remote user. For example, they allow the component to determine the principal name of the caller or remote user to use as a database key.

Securing Servlets

Servlet 3.0 provides the `authenticate`, `login`, and `logout` methods of the `HttpServletRequest` interface. With the addition of the `authenticate`, `login`, and `logout` methods to the Servlet specification, an application deployment descriptor is no longer required for web applications but may still be used to further specify security requirements beyond the basic default values.

Comparing Declarative Security and Programmatic Security

- Declarative security should be used whenever possible.
- Programmatic security can make the application harder to port, change, and extend.
- Programmatic security should be used only to overcome the limitations of declarative security.
- Typical scenarios that require programmatic security include:
 - Implementing identity-based business logic
 - Implementing identity-based access control
 - Implementing component-managed EIS sign-on
 - Implementing programmatic login



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

Declarative security should be used whenever possible because it provides a complete separation between application logic and the underlying security service. On the other hand, because programmatic security builds security awareness directly into the application, it can make the application harder to port, change, and extend. Because of this, programmatic security should be used only to overcome the limitations of declarative security.

Implementing identity-based business logic

For example, when a user logs on to an online banking application, the application displays a list of accounts for the user. To achieve this, you can use an implementation similar to the following code snippet:

```
Principal user = request.getUserPrincipal();
String userName = user.getName();
// Search for the accounts for this user
Principal user = request.getUserPrincipal();
String userName = user.getName();
// Search for the accounts for this user
```

Note: The name of a Principal could have different content depending on the security service provider that authenticates the user. Because of this, the application should be aware of the security service provider, and the application should be carefully designed to associate the authenticated principal with the business logic, for example, by parsing the content of the principal's name. This can limit the portability of the application.

Implementing identity-based access control

The declarative security model defines a role-based authorization model. For example, you can allow users in the Customer role to call the getAccount method implemented in an EJB component. However, the declarative security does not allow you prevent one user in the Customer role from accessing other account balances. To achieve this, you can implement the getAccount method using programmatic security as follows:

```
public Account getAccount(int accountId) { Principal caller =  
    sessionCtx.getCallerPrincipal(); String name = caller.getName(); //  
    find this caller's account IDs if(/* accountId belongs to the caller  
    */)  
    {  
        //allow access }else{  
        // deny it  
    }
```

Implementing component-managed EIS sign-on

EIS sign-on refers to the mechanism for an application component to get a connection to a resource, such as a database. In most cases, EIS sign-on should be managed by the container. This is achieved by configuring the authentication information at the container level. If it is required to sign on to the resource with different credentials depending on the identity or role of the component caller, you can use component-managed sign-on.

Implementing programmatic login

If the standard authentication mechanisms in Java EE are not sufficient, you might have to implement a programmatic authentication solution to provide the security credentials directly to the security service provider. However, the capability of programmatic login varies significantly among application server vendors. Therefore, a programmatic login solution is never portable.

Servlet 3.0 specifies the following methods of the HttpServletRequest interface that enable you to authenticate users for a web application programmatically:

- **authenticate**, which allows an application to instigate authentication of the request caller by the container from within an unconstrained request context. A login dialog box displays and collects the user name and password for authentication purposes.
- **login**, which allows an application to collect username and password information as an alternative to specifying form-based authentication in an application deployment descriptor.
- **logout**, which allows an application to reset the caller identity of a request.

Lesson Agenda

- Security technologies and regulatory constraints
- Impact of security in distributed computing
- Security in the Java EE technology
- Web services security



ORACLE

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

Understanding Web Services Security

- Many of the security requirements in web services are similar to those in Java EE components, for example, authentication, authorization, data integrity, and confidentiality.
- However, several characteristics of web services have presented additional challenges in securing web services, including:
 - The loosely coupled nature of web services
 - The end-to-end security requirements of web services
 - The interoperable requirements of web services



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

Even though the Java EE security model can be applied to secure web services implemented as a Java EE component, several characteristics of web services have presented additional challenges in securing web services, including:

The loosely coupled nature of web services

The invocation of a web service is often initiated by consumers who do not have a previous relationship with the service provider. It can be difficult to establish trust between these two parties.

The end-to-end security requirements of web services

Messages exchanged between the web service consumer and the ultimate service provider are often processed by intermediary nodes. In this case, transport-layer security does not provide an end-to-end solution because while the message is being processed by an intermediary, the message is not protected. Because of this, it is necessary to implement message-level security to secure the web service in its entire process path.

The interoperable requirements of web services

Web services are commonly invoked by business partners and clients from different platforms and devices, using different protocols.

You must secure the web services in such a way that the security solution does not limit their interoperability. For example, the security mechanism should be independent of the transport protocol. Otherwise, it can severely limit the accessibility.

Note: The Basic Security Profile (BSP) was defined by the WS-I organization as an extension to the Basic Profile of web service interoperability. BSP 1.0 provides a set of guidelines and requirements to use the existing security-related standards, including WS-Security, XML Signature, XML Encryption, security tokens, and transport-layer security. Standardizing web service security is still an evolving process. Because of this, the support for web service security varies significantly among vendors. Currently, the most widely adopted web service security standards include:

- XML Signature
- XML Encryption
- Web Services Security (WS-Security)
- SAML

Types of Attacks:

- Spoofing
- Taking advantage of bugs
- Denial of Service
- Repudiation

XML Signature

- Defined by the World Wide Web Consortium (W3C)
- Used to achieve message authentication and integrity
- Allows any content to be digitally signed, and the signed content is expressed in XML
- When applied to XML content, XML signature allows you to sign a complete XML document or just an XML fragment.
- The JSR-105, the XML Digital Signature APIs, defines a Java API to support XML Signature.



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

The XML Signature standard defined by the World Wide Web Consortium (W3C) can be used to achieve message authentication and integrity. XML signature allows any content to be digitally signed, and the signed content is expressed in XML. When applied to XML content, the XML signature allows you to sign a complete XML document or just an XML fragment.

The process of signing an XML document or fragment involves the following steps:

- Identify the XML content or a fragment that requires digital signing.
- Apply necessary transformation and canonical rules to the target data.
- Calculate a message digest of the target data.
- Digitally sign the message digest using a private key.

For example, the following XML document represents a purchase order:

```
<?xml version="1.0" encoding="UTF-8"?>
<PurchaseOrder>
<Item number="12345678">
<Description>SL425 ILT</Description>
<Price>2500.00</Price>
</Item>
<Buyer id="8492340">
<Name>Student</Name>
</Buyer>
</PurchaseOrder>
```

The following XML document represents an enveloped-signature format that embeds the XML signature within the original document. In this example, the signer's X.509 digital certificate is included in the result so that the receiver of the signed document can use the public key in the certificate to validate the signature.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<PurchaseOrder>
<Item number="12345678">
<Description>SL425 ILT</Description>
<Price>2500.00</Price>
</Item>
<Buyer id="8492340">
<Name>Student</Name>
</Buyer>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
<Reference URI="">
```

```
<Transforms>
<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>LWHfaw8miJl9F+2RWXcy/YjzrHY=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>fys04Zdamo0uiVumPzlRU6BAqfenJCOe8zzWXNLbQ==</Signatu
reValue>
<KeyInfo>
<X509Data>
<X509SubjectName>CN=USER, OU=EDU, O=COM, C=US</X509SubjectName>
<X509Certificate><!-- BASE-64 encoded X509 Cert --
></X509Certificate>
</X509Data>
</KeyInfo>
</Signature>
</PurchaseOrder>
```

Using XML Signature, message authentication can be achieved by validating the XML data using the sender's digital certificate. Message integrity is achieved using the message digest.

The JSR-105, the XML Digital Signature APIs defines a Java API to sign digital content and validate the digital signatures. It provides a Java technology-based implementation of the XML Signature specification. The JSR-105 API can be used in any situation where you need to digitally sign XML data.

The JSR-105 API is bundled in the Java SE 6.0 SDK software.

XML Encryption

- Also defined by W3C
- Can be applied to any type of digital content, and the result of encryption is expressed in XML
- For XML content, the encryption can be applied at the XML document and fragment-level.
- Multiple encryptions can be applied to different portions of an XML message.
- XML Signature and XML Encryption combined provides the foundation of message level security.
- The JSR-106, XML Digital Encryption APIs, defines a Java API that implements XML Encryption.



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

XML Encryption is another standard defined by the W3C to address the issue of message confidentiality. Like XML Signature, you can apply XML Encryption to any type of digital content, and the result of encryption is expressed in XML. For XML content, you can apply the encryption to the XML document and fragments of the XML document. You can apply multiple encryptions to different portions of an XML message. XML Signature and XML Encryption combined provide the foundation of message-level security.

The following example shows the results of encrypting the purchase order content. In this example, the original XML document is encrypted using the AES-128 algorithm, while the AES-128 encryption key is encrypted using the TRIPLEDES algorithm with a secret key.

```

<PurchaseOrder>
<xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Type="http://www.w3.org/2001/04/xmlenc#Content">
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" />
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" />
<xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:CipherValue xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
21DTtDR+c6CedvE32sP9cy4QE/cvMGN3DpWxExocaOY=
</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedKey>
</ds:KeyInfo>
<xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:CipherValue xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
ID71vE7UArO7RLgSip9C8puppnZsky4Uc0iILsqmRwFS/194YVwGOr5Vwyflg6K9TCtq
3jHXmiNQ
o7Tx1XBj+L8W6eomJzBEP LspCqoSyl3wSaZO3iinXMr+KWGzy5o3rCFVPIMYecxnia4D
sBwj tWum
LcnNB/TK+8qQPqsc3M0MVgqxupQeMqO3YAfa/FU27ZWmnMhpPXbt7Qjz946mSdtbtKYj
OKtybmtw
wJ1Q1Jk=
</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
</PurchaseOrder>

```

The JSR-106, XML Digital Encryption API is a specification. The goal of JSR-106 is to provide a standard Java API that implements the XML Encryption standard. Currently, JSR-106 is still under development. Therefore, the Java APIs you use for XML encryption can vary among vendors.

A popular open source implementation that supports XML Encryption is the Apache XML Security library. You can find more information about the library at <http://xml.apache.org/security>.

Web Service Security (WS-Security)

- The Web Services Security: SOAP Message Security (WS-Security or WSS) specification defined by OASIS addresses the common security requirements for web services, including message authentication, message integrity and confidentiality, and security token propagation.
- Defined as a security-oriented extension to SOAP, WS-Security relies on existing security standards and technologies to maintain the interoperability.



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

Essential Definitions in WS-Security

Claim: A claim is a declaration made by an entity (such as name, identity, key, group, privilege, and capability).

Claim Confirmation: A claim confirmation is the process of verifying that a claim applies to an entity.

Security Token: A security token represents a collection of claims. Examples of security tokens include user names or X.509 certificates. WS-Security also defines the mechanisms for using XML-based security tokens.

Signed Security Token: A signed security token is a security token that is asserted and cryptographically signed by an authority. An X.509 certificate is an example of a signed security token, while a user name token is not.

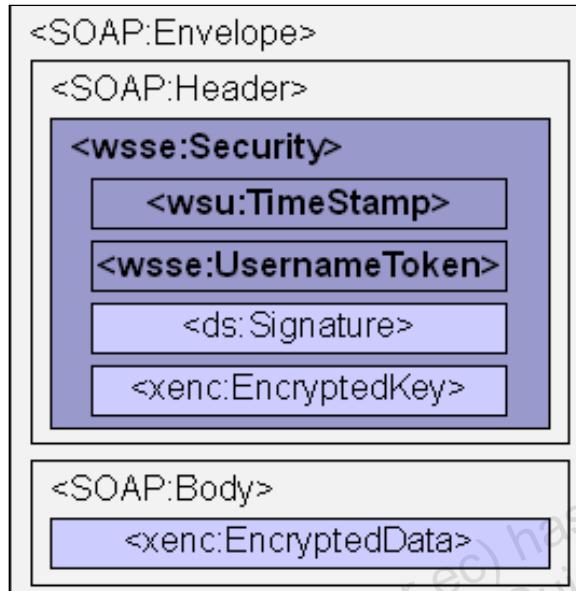
Trust: Trust is the characteristic that one entity is willing to rely upon a second entity to execute a set of actions or to make a set of assertions or both about a set of subjects or scope or both.

Addressing Web Services Security Requirements

WS-Security provides a standard-based and transport-protocol independent approach to address the following requirements:

- **Message integrity:** achieved through XML Signature
- **Message confidentiality:** achieved through XML Encryption
- **Authentication and authorization:** achieved through security tokens

Message Structure in WS-Security



ORACLE

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

As an extension of SOAP, WS-Security defines a security header schema to describe the structure and processing rules for the SOAP messages. It illustrates a high-level structure of a WS-Security enabled SOAP message. A brief explanation of each element is illustrated in the slide.

A SOAP message can contain multiple Security header blocks. Each header block represents a complete WS-Security-enabled SOAP message targeted to a recipient or a SOAP intermediary actor.

The `TimeStamp` element is used to specify the timeline and the validity of the security semantics, for example, the creation and expiration time.

In this example, the `UsernameToken` element is used to attach a user name security token to the security semantics. Other tokens can be used, including X.509 certificates and XML-based tokens.

The `Signature` element represents an XML Signature-based digital signature of the message, or a portion of the message.

The **EncryptedKey** element represents the secret key used to encrypt the data in the SOAP envelope. This secret key is encrypted using the recipient's public key, so only the actual recipient can decrypt the key then use the key to decrypt the message data. The **EncryptedKey** element typically contains a list of references to the message portions that are encrypted with the specific key.

The **EncryptedData** element represents the encrypted data in the SOAP message using XML Encryption.

Web Services Security and Java EE

- Java EE security features can be incorporated into the Web service endpoint component.
- The Java EE specification does not address the issue of message level security.
- Implementing an end-to-end security solution for Web services typically depends on value-add services provided by the application server vendor.

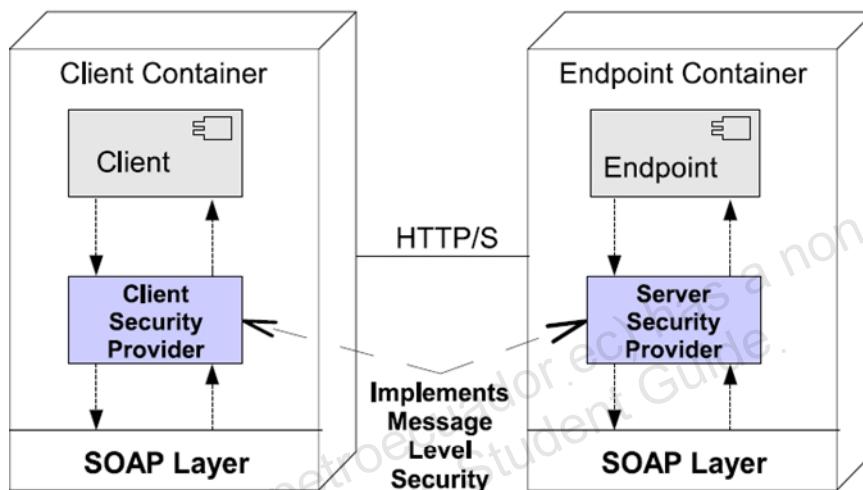


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Java EE provides an ideal platform to develop and deploy Web services. When you develop and deploy web service endpoints using the Java EE technology, many security features provided by the Java EE container can be leveraged, such as authentication, authorization and transport-level security. Unfortunately, securing web services at the message level using previously mentioned technologies is not yet part of the Java EE technology. Because of this, at this moment, implementing an end-to-end security solution for web services typically depends on value-add services provided by the application server vendor.

Web Service Security in Oracle WebLogic

- The WS-Security support is integrated in web service client and server-side containers.
- Enabling message-level security does not require changes to the web service implementation.



ORACLE

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

In WebLogic, the support for the WS-Security standard is integrated in Web-service client and server-side containers. This container-based enforcement allows message-level security to be applied to a web service application without requiring changes to the implementation. The container-based message-level security feature is illustrated in the slide.

WebLogic provides facilities to bind SOAP-layer message security providers and message protection policies to containers and to applications deployed in containers. You can configure SOAP-layer message security providers in the client and server-side containers. During installation, the providers are configured with a simple message protection policy that, if bound to a container, or to an application or client in a container, causes the source of the content in all request and response messages to be authenticated by an XML digital signature.

The WebLogic application server also supports the Web Service Interoperability Technologies (WSIT). WSIT was also tested in a joint effort by Sun Microsystems, Inc. and Microsoft with the expressed goal of ensuring interoperability between web services applications developed using either WSIT or the Windows Communication Foundation (WCF) product.

Web Service Interoperability Technology (WSIT)

- The goal of WSIT is to ensure interoperability between Web services applications developed using either WSIT and the Windows Communication Foundation (WCF) product.
- WSIT also enhances security by implementing:
 - WS-Secure Conversation
 - Web Services Security Policy
 - Web Services Trust



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

WSIT enhances security by implementing WS-Secure Conversation, which enables a consumer and provider to establish a shared security context when a multiple-message-exchange sequence is first initiated. Subsequent messages use derived session keys that increase the overall security while reducing the security processing overhead for each message.

WSIT also implements two additional web service security features:

Web Services Security Policy: Enables web services to use security assertions to clearly represent security preferences and requirements for Web service endpoints.

Web Services Trust: Enables web service applications to use SOAP messages to request security tokens that can then be used to establish trusted communications between a client and a web service.

WSIT implements these features in such a way as to ensure that web service binding security requirements, as defined in the WSDL file, can interoperate with and be consumed by WSIT and WCF endpoints.

For more information about WSIT, visit the website
<http://java.sun.com/Webservices/interop>.

Security Assertion Markup Language

SAML 1.1:

- Is a popular standard from OASIS
- Describes an interoperable XML framework for exchanging security information between systems
- Uses *assertions* to express trusted statements about a subject wanting to use a service (user ID, group ID, company, and so on)
- Supports SOAP over HTTP
- Defines *profiles* for common communication patterns, such as Web single sign-on (SSO) and WS-Security
- Recommends that assertions be digitally signed to avoid “man-in-the-middle” attacks



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

The SAML standard defines a framework for exchanging security information between software entities on the Web. SAML security is based on the interaction of asserting and relying parties. An assertion is a package of information that supplies one or more statements made by an SAML Authority. The following types of statements are supported:

- Authentication statements that say when and how a subject was authenticated
- Attribute statements that provide specific information about the subject (for example, what groups the Subject is a member of)
- Authorization statements identify what the subject is entitled to do. SAML authorization is not supported in this release of WebLogic Server.

SAML defines a request/response protocol for obtaining assertions. A SAML request can ask for a specific known assertion or make authentication or attribute decision queries, with the SAML response providing back the requested assertions. The XML format for protocol messages with their allowable extensions is defined in an XML schema. In WebLogic Server, request/responses are used for POST and Artifact profiles.

Profiles are technical descriptions of particular flows of assertions and protocol messages that define how SAML can be used for a particular purpose. WLS supports the web services SAML Token profile, both as a Web services client and a web services server. Web services supports the use of SAML assertions as security tokens in messages and generates/validates them by calling the SAML Credential Mapper and Identity Asserter

SAML Terminology

An *asserting party*:

- Acts as a central authority
- Confirms that some subject has been authenticated and has specific security attributes

A *relying party*:

- Relies on an asserting party for any security assertions associated with a given request
- Can make additional authorization decisions based on the supplied security attributes
- Must have a trusted relationship with the asserting party (using PKI, for example)



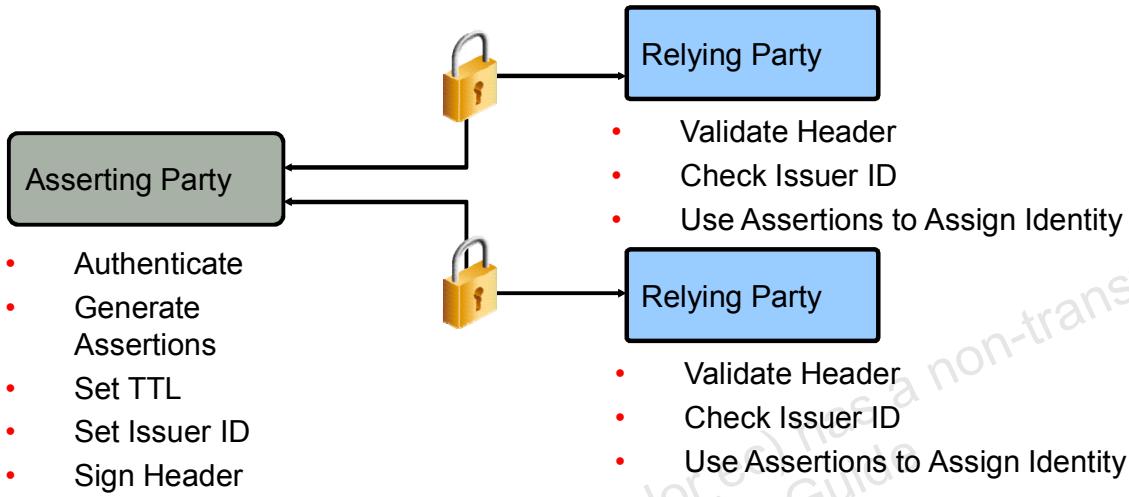
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The asserting party asserts that a user has been authenticated and given associated attributes. For example, there is a user Dan Murphy with the email address “dmurphy@company.com,” and he is authenticated to this domain using a password mechanism. Asserting parties are also known as SAML authorities.

The relying party determines whether it trusts the assertions provided to it by the asserting party. SAML defines a number of mechanisms that enable the relying party to trust the assertions provided to it. Although a relying party may trust the assertions provided to it, local access policy defines whether the subject may access local resources. Therefore, even if a relying party trusts that a user is Dan Murphy, it does not mean Dan Murphy can access all the resources in the domain.

Just providing assertions from an asserting party to a relying party may not be adequate for a secure system. How does the relying party trust what is being asserted to it? In addition, what prevents a “man-in-the-middle” attack that grabs assertions to be illicitly “replayed” at a later date? SAML defines a number of security mechanisms that prevent or detect such attacks. The primary mechanism is for the relying party and asserting party to have a preexisting trust relationship, typically involving a PKI. Although use of a PKI is not mandated, it is recommended. Use of particular mechanisms is described for each profile.

SAML WS-Security Architecture



ORACLE

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

SAML assertions added by the asserting party to the WS-Security header include a field named Issuer. This field can have any textual value, but is typically formatted as a URI, such as “www.xyz.com/saml.” Relying parties such as WLS can be configured to accept or reject SAML assertions from specific issuers.

To ensure that relying parties can have confidence that received assertions have not been forged or altered since they were issued, SAML assertions appearing in WS-Security header elements should be signed by their issuing authority (the asserting party). Transport layer security may be used to protect the assertions from modification while in transport, but signatures are required to extend such protection through intermediaries.

Replay attacks can be detected by replying parties if asserting parties include additional message identifying information, such as time-to-live (TTL) identifiers. Relying parties can check this information against previously received values.

SAML WS-Security Profiles

| Directory | Description |
|--------------------|--|
| WSS/Sender-Vouches | Relying parties must have an existing PKI trust relationship, so that they can validate digitally signed assertions. |
| WSS/Holder-Of-Key | Asserting parties include an X.509 certificate in the header that can be used to validate digital signatures. |



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

Subject confirmation refers to the method that is used to verify the subject of a SAML subject statement in a SAML assertion. An attesting entity, which is presumed to be different from the subject, vouches for the verification of the subject. The receiver of the SAML token has already established a trust relationship with the attesting entity. By confirming the attesting entity, the receiver, therefore, trusts the subject. This method of subject confirmation is called “Sender-vouches.”

“Holder-of-key” is for the attesting entity to insert an X.509 public key into a SAML assertion. In order to protect the embedded certificate, the SAML assertion must be signed by the attesting entity. The signature that covers the certificate will be verified by the receiver. The purpose of SAML token with “holder-of-key” subject confirmation is to allow the subject to use an X.509 certificate that may not be trusted by the receiver to protect the integrity of the request messages. The signature verification is done as part of the SAML assertion validation process. After everything is verified, the receiver uses the embedded X.509 from the KeyInfo node of the SAML assertion to validate the actual message that is signed by the subject. The receiver will not further validate the embedded X.509 certificate because it has an already established trust relationship with the attesting entity that trusts the subject and signs its certificate.

Additional Resources

- Kanneganti, Ramarao. *SOA Security*, Manning Publications, 2008.
- Shah, Shreeraj. *Web 2.0 Security - Defending AJAX, RIA, and SOA*, Charles River Media, 2007.
- Bertino, Elisa, Lorenzo Martino, Federica Paci, and Anna Squicciarini. *Security for Web Services and Service-Oriented Architectures*, Springer, 2009).
- Steel, Christopher, Ramesh Nagappan, and Ray Lai. *Core Security Patterns: Best Practices and Strategies for J2EE™, Web Services, and Identity Management*. Upper Saddle River: Prentice Hall PTR, 2006.



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

Web Services Interoperability Organization (WS-I). Basic Security Profile Version 1.0.

[<http://www.ws-i.org/Profiles/>

BasicSecurityProfile-1.0.html], accessed February 27, 2007.

OASIS Standards. [<http://www.oasis-open.org/specs>], accessed February 27, 2007.

Brunette, Glenn. Toward Systemically Secure IT Architectures.

[<http://www.sun.com/blueprints/0206/819-5605.pdf>], accessed February 27, 2007.

National Institute of Standards and Technology (NIST) Special Publications 800 Series.

[<http://csrc.nist.gov/publications/nistpubs/index.html>], accessed February 27, 2007.

SANS Institute. Top-20 Internet Security Attack Targets. [<http://www.sans.org/top20>], accessed February 27, 2007.

The Open Web Application Security Project (OWASP). [<http://www.owasp.org>], accessed February 27, 2007.

Quiz

Key to a security architecture is limiting who has access to what. Which of the following models of security best addresses the requirement that every file and application is labeled with a specific security access level?

- a. Mandatory Access Control
- b. Discretionary Access Control
- c. Role-based Access Control



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

Answer: a

MAC is based on label relationships.

Quiz

Which of the following forms of security breaches are preventable using better software and architectural design principals? Choose all that apply.

- a. SQL Injection
- b. Social Engineering
- c. Pretexting
- d. Cross-site scripting



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

Answer: a, d

These attacks can be prevented through technology.

The other two require training of people to prevent these vulnerabilities.

Quiz

Choose the best solution to prevent an attack on the company systems by a system tunneling in on a open port.

- a. Patches and system upgrades
- b. A well-established virus scanning email server system
- c. Firewalls, routers, and switches
- d. Frequent communication and training



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

Answer: b

Closing down susceptible ports

Quiz

Least privilege:

- a. Describes the process of assigning passwords to users
- b. Provides just enough access to users and applications to perform a task
- c. Defines the level of privilege granted to guest users
- d. Is implemented in Windows Operating systems

 ORACLE

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

Answer: b

Quiz

In Java EE, a protection domain is a set of components that are assumed to trust each other. Protection domains:
(choose all that apply)

- a. Provide that components within a protected domain do not need authenticate to one another
- b. Allow interaction across boundaries for some types of users
- c. Do not distinguish between legitimate authentication and authentication gained by attack
- d. Rely on least privilege



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

Answer: a, c

Summary

In this lesson, you should have learned how to:

- Analyze the impact of security in distributed computing
- Describe the security services in Java technology
- Define security requirements for web services

Practice 3 Overview: Identifying Security Risks

This practice covers the following topics:

- Identify security risks in the architecture.

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.

Understanding Nonfunctional Requirements



ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Describe the nonfunctional requirements of an enterprise application
- Describe common practices for improving systemic qualities
- Prioritize quality-of-service (QoS) requirements
- Inspect for trade-off opportunities
- Study the impact of dimensioning on nonfunctional qualities



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

Discussion Questions

- What are the general approaches that you can use to scale a system? How do the different approaches affect the NFRs of the resulting system?
- Because you cannot reasonably expect a system to deliver all nonfunctional requirements, how can you decide which requirements to trade off and which to keep?



ORACLE®

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

Lesson Agenda

- Examining Nonfunctional requirements (NFRs)
- Common practices for improving qualities
- Prioritizing Quality-of-Service requirements
- Inspecting QoS requirements for trade-off opportunities



ORACLE

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

Functional and Nonfunctional Requirements

- A functional requirement defines a function of a software system or its component.
- A function is described as a set of inputs, the behavior, and outputs (see also software).
- A nonfunctional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors.
- In general, functional requirements define what a system is supposed to do whereas nonfunctional requirements define how a system is supposed to be.
- In this course we focus on nonfunctional requirements.



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

Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. Behavioral requirements describing all the cases where the system uses the functional requirements are captured in use cases. Functional requirements are supported by nonfunctional requirements (also known as quality requirements), which impose constraints on the design or implementation (such as performance requirements, security, or reliability). How a system implements functional requirements is detailed in the system design.

Examples of NFRs include:

- Get a bid result in ten seconds or less
- Meet an SLA
- 99.99% uptime
- How up to date the data displayed to the user is
- Most forms of constraints on the system
- The system must be unavailable from midnight until 1:00 am for backups

Nonfunctional Requirements

“A system has properties that emerge from the combination of its parts. These emergent properties will surely be a matter of accident, not design, if the nonfunctional requirements, or system qualities, are not specified in advance.”

- Ruth Malan and Dana Bredemeyer



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

Nonfunctional requirements are properties of a system that establish the quality of service that the system can deliver. Different nonfunctional requirements impose various pressures on the overall system design. Therefore, you should evaluate and prioritize the requirements so that you can design the architecture to minimize constraints on those qualities.

Before you begin to work with nonfunctional requirements, get a written consensus on the meaning of any of the terms defined in this section that you also use in your projects. The meanings given in this section reflect the usage of each term in this course.

Nonfunctional requirements describe the end goal of building an enterprise application to meet the service level agreement (SLR). Examples of NFRs include performance, throughput, security, and availability.

Nonfunctional Requirements Categories

- Manifest requirements are obvious to the user:
 - Performance (For example, Response Time)
 - Reliability (includes Consistency)
 - Availability
 - Usability
- Operational requirements are tied to the running system:
 - Throughput
 - Manageability
 - Security
 - Serviceability
 - Testability



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

The nonfunctional requirements are sometimes grouped into categories or families according to how they are observed or what they affect. The families for the nonfunctional requirements are manifest, operational, developmental, and evolutionary.

Manifest requirements are those qualities that are obvious to the user:

- **Performance:** A measure of the response time as seen by the user
- **Reliability:** A measure of the probability that data are correct, both in the backing storage and in the results presented to the user
- **Availability:** A measure of the probability that the system is operating at the moment a user attempts to use it and that the system continues to operate to the completion of the user's request
- **Usability:** A measure of the ability that a user can learn and use the system to accomplish a task

Operational qualities are evident when the system is running but are not immediately evident to a single user. Operational qualities include:

- **Throughput:** A measure of the total amount of work that can be performed in a given time

Throughput is a good example of a quality that is evident when the system is running, but not immediately evident to a single user. Throughput is closely related to performance, but a single user cannot see it. In fact, throughput is aggregate performance for many users.

- **Manageability:** A measure of the amount of human intervention, such as backups required to keep the system running smoothly
- **Security:** A measure of the confidence that data are protected from unauthorized disclosure or modification
- **Serviceability:** A measure of the amount of work that is required to perform nonroutine maintenance, such as the replacement of a component
- **Testability:** A measure of the amount of work that is required to identify and isolate a fault or error in the system

Nonfunctional Requirements Categories

- Developmental requirements affect how the system is built:
 - Realizability
 - Planability
- Evolutionary requirements relate to how the system behaves when it is altered or upgraded:
 - Scalability
 - Maintainability
 - Extensibility
 - Flexibility
 - Reusability
 - Portability



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

Developmental qualities affect how the system is built. These qualities are typically focused on the project during its development and become irrelevant as the system evolves.

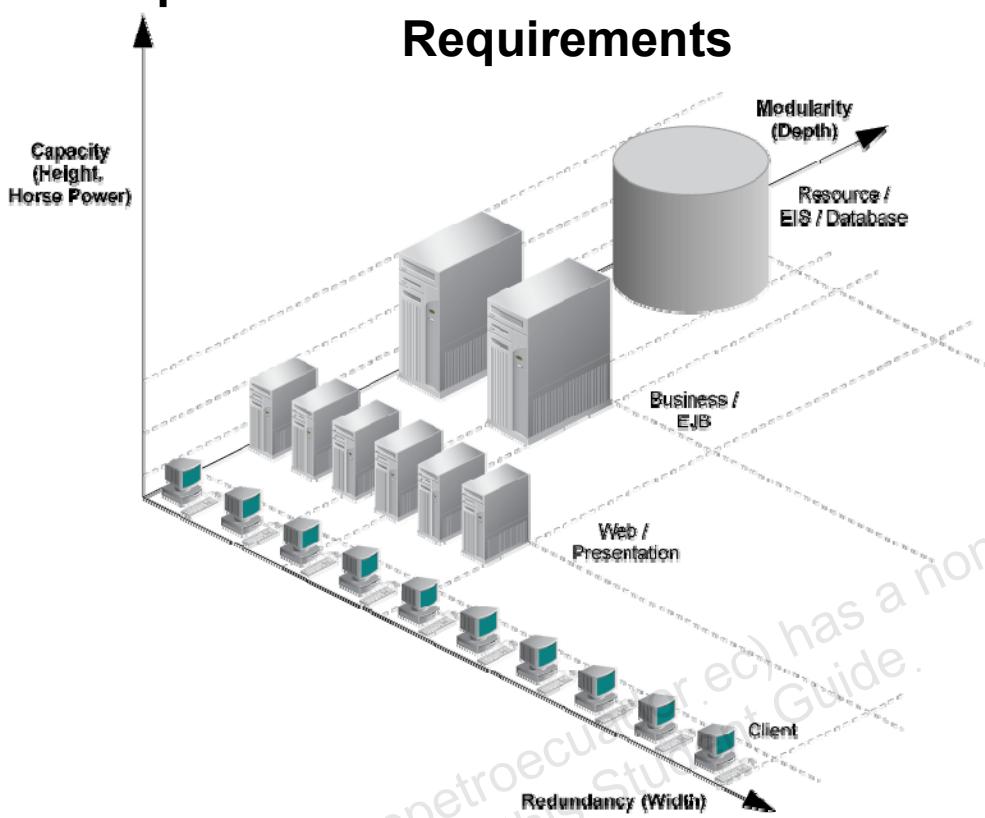
- **Realizability:** A measure of the probability of the successful implementation of a feature or product
- **Planability:** A measure of the confidence in a predictable plan and cost estimates

Evolutionary qualities relate to how the system behaves when it is altered or upgraded. Scalability is perhaps the most compelling of the evolutionary qualities, which include:

- **Scalability:** The inverse of a measure of the work and cost required to modify the system to provide higher throughput
- **Maintainability:** The inverse of a measure of the work involved in making routine bug fixes
- **Extensibility:** The inverse of a measure of the work and cost of adding new features of functionality to the system
- **Flexibility:** The inverse of a measure of the work and cost of making changes to the system

- **Reusability:** A measure of the probability that, given a new project with comparable requirements, a part of the system can be successfully integrated into the new system
- **Portability:** The inverse of a measure of the work or cost of migrating components to a new platform

Impact of Dimensions on Nonfunctional Requirements



ORACLE

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

As part of your evaluation, and from a system computational view point, you can think of the gross layout of an architecture (tiers, layers, and so on) as having six independent variables that are expressed as dimensions:

- **Capacity**: The dimension of raw power in an element, perhaps a powerful CPU, fast network connection, or a capacious storage medium. Capacity is increased through vertical scaling and is sometimes referred to as height. Another commonly used term is horsepower.
- **Redundancy**: The dimension in which multiple systems work on the same job, such as load balancing among several Web servers. Redundancy is increased through horizontal scaling and is also known as width.
- **Modularity**: The dimension in which you divide a computational problem into separate elements and spread those elements across multiple computer systems. Modularity, also referred to as depth, indicates how far into a system you have to go to get the data you need. Keep in mind that as you go deeper into the system, you gain additional working components. So, modularity works like a bucket brigade transferring water to a fire.

- **Tolerance:** The dimension of time available to fulfill a request from a user. Tolerance is closely bound with the overall performance of the system. Higher tolerance allows the system more time to complete operational requests without penalty.
- **Workload:** The dimension of computational work being performed at a particular point within the system. Workload is closely related to capacity in that workload consumes available capacity, which leaves fewer resources available for other tasks.
- **Heterogeneity:** The dimension of diversity in technologies that is used within a system or one of its subsystems. Heterogeneity comes from the variation of technologies that are used within a system. This might come from a gradual accumulation over time, inheritance, or acquisition of other systems. Heterogeneity can also be planned for the purpose of satisfying other requirements.
- **Note:** Certain dimensions tend to increase a NFR quality (+), some dimensions tend to decrease it (-), some dimensions might either increase or decrease an NFR quality (+/-), and some dimensions have no effect.

The following definitions relate to NFRs:

- **Height:** Capacity of an individual node or horsepower, which is increased by vertical scaling
- **Width:** Number of paths through a tier, which is increased by horizontal scaling
- **Depth:** Number of tiers involved in processing a request, which is decreased by caching
- **Time:** Amount of flexibility or tolerance in the time required to fulfill a request
- **Density:** Amount of computing effort required by a particular point in the request path
- **Diversity:** Degree of technology variation in a partition of the system

Impact of Dimensions on Nonfunctional Requirements

| Category | Quality | Capacity | Redundancy | Modularity |
|---------------|-----------------|----------|------------|------------|
| Manifest | Performance | + | +/- | +/- |
| | Reliability | | + | - |
| | Availability | + | + | - |
| | Usability | | | |
| Operational | Throughput | + | + | + |
| | Manageability | | - | - |
| | Security | | - | + |
| | Serviceability | | - | - |
| Developmental | Realizability | | | |
| Evolutionary | Scalability | + | + | + |
| | Maintainability | | | + |
| | Flexibility | | | + |
| | Reusability | | | + |



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

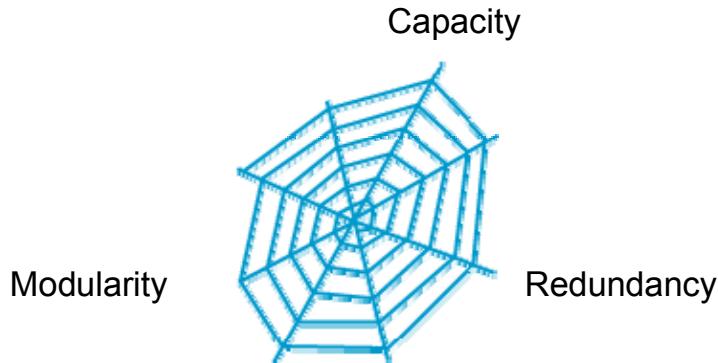
Capacity (or raw compute resources) can clearly affect the performance-related qualities, such as increasing throughput and reducing response times. Typically, capacity does not affect other qualities. However, if increased capacity results from more powerful hardware and this hardware is less reliable, then capacity can affect other qualities.

Redundancy (or the use of multiple of a given service to work in parallel) improves reliability because if one machine fails, there is another machine that can still provide some level of service. Introducing redundancy to the system can improve the performance because of the additional processing power. However, if designed poorly, redundancy can degrade the performance due to overhead, such as additional network traffic. Another effect of redundancy is that it tends to make the system more complex. Complexity reduces manageability, serviceability, and security.

Modularity (or dividing a job into separate parts and splitting those parts across multiple systems) is a significantly more complex proposition. Modularity usually increases throughput, but the performance, as seen by individual users, does not necessarily improve. You can improve security because each piece of the system can be protected individually, which makes it more difficult for an attack to compromise the whole system at once.

However, reliability and availability decrease because now multiple machines must be working at the same time to complete the job. Manageability, serviceability, and realizability are reduced precisely because the system is more complex. On the other hand, if you break up the system into carefully considered elements, you can modify each element in relative isolation. The result is that modularity tends to increase all the qualities in the evolutionary group, including scalability, maintainability, flexibility, and reusability.

Making Trade-Off Decisions on System Dimensions



ORACLE

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

As demonstrated, when you improve a particular dimension of the system, it rarely has a positive impact on all the nonfunctional requirements. Because of this, when you design the system, you should prioritize QoS requirements of the system, and improve the most important dimension or dimensions to satisfy the QoS requirements in a cost-effective manner.

Typically, the three dimensions listed do not have a direct impact on the usability of the system. However, as more and more software systems are designed to face a large number of Internet-based users with different experience, the importance of usability has increased significantly. To achieve a high degree of user satisfaction, at the architecture level, you should consider issues, such as the types of user interfaces, user error handling, and internationalization.

Capture and Examine NFRs

Capture:

- Interviews
- Meetings
- Facilitated workshops
- Consider a wiki for capturing the NFRs
- Feedback forms
- Complaints from customers or users

Examine:

- Cost-benefit analysis
- Prioritize NFRs
- Localize NFRs to the problem domain



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

- **Facilitated workshops:** Fast, high quality, strong buy-in, less rework, and enhancement requests
- **Cost-Benefit Analysis:** The process involves, whether explicitly or implicitly, weighing the total expected costs against the total expected benefits of one or more actions in order to choose the best or most profitable option.
- **Localize NFRs to the Problem Domain:** Different NFRs are specified for different parts of the system.

Lesson Agenda

- Examining nonfunctional requirements (NFRs)
- Common practices for improving qualities
- Prioritizing Quality-of-Service requirements
- Inspecting QoS requirements for trade-off opportunities



ORACLE

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

Introducing Redundancy to the System Architecture

- Redundancy can be applied to either the vendor products or the server systems themselves.
- Redundancy is typically implemented with replication.
- Common techniques include:
 - Load balancing
 - Failover
 - Cluster



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

Over the years, software and system engineering practices have developed many best practices for improving NFRs. By applying these practices to the system at the architecture level, you can gain a higher level of assurance for the success of the system development.

Many infrastructure-level practices for improving systemic qualities rely on using redundant components in the system. You can apply these strategies to either the vendor products or the server systems themselves. The choice depends primarily on the cost of implementation and the requirements, such as performance, throughput, and scalability.

Load Balancing

Load balancing Strategies:

- Load balancers in network switches
- Load balancers in cluster management software and application servers
- Load balancers based on server instance DNS configuration

Load balancing algorithms:

- Round-robin algorithm
- Response-time or first-available algorithm
- Least-loaded algorithm
- Weighted algorithm
- Client DNS based algorithm
- Sticky round-robin



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

To implement load balancing, you usually select a load-balancer implementation based on its performance and availability. Consider the following:

Load balancers in network switches: Load balancers that are included with network switches and are commonly implemented in firmware, which gives them the advantage of speed.

Load balancers in cluster management software and application servers: Load balancers that are implemented with software are managed closer to the application components, which gives greater flexibility and manageability.

You can implement load balancing to address the architectural concerns such as throughput and scalability. Load balancing is a feature that allows server systems to redirect a request to one of several servers based on a predetermined load-balancing algorithm. Load balancing is supported by a wide variety of products, from switches to server systems, to application servers. The advantage of load balancing is that it lets you distribute the workload across several smaller machines instead of using one large machine to handle all of the incoming requests. This typically results in lower costs and better use of computing resources.

Load balancers based on the server instance DNS configuration: Load balancer is configured to distribute the load to multiple server instances that map to the same DNS host name. This approach has the advantage of being simple to set up, but typically it does not address the issue of session affinity.

Round-robin algorithm: Picks each server in turn

Response-time or first-available algorithm: Constantly monitors the response time of the servers and picks the one that responds the quickest

Least-loaded algorithm: Constantly monitors server load and selects the server that has the most available capacity

Weighted algorithm: Specifies a priority on the above algorithms, giving some servers more workload than others

Client DNS-based algorithm: Distribute the load based on client's DNS host and domain name information

Sticky round-robin: Requests for a given session are sent to the same application server instance. With a sticky load balancer, the session data is cached on a single application server rather than being distributed to all instances in a cluster. Therefore, the sticky round robin scheme provides significant performance benefits that normally override the benefits of a more evenly distributed load obtained with pure round robin.

In addition to these solutions, most load balancer implementations allow you to create your own load balancing strategy and install it for use.

Your selection of a load balancing strategy is largely based on the type of servers you are managing, how you would like to distribute the workload, and what the application domain calls for in performance. For example, if you have equally powerful machines and a fairly even distribution of transaction load in your application, it would make little sense to use a weighted algorithm for load balancing. This approach could result in an overloaded system that might fail. An equal distribution of workload would make more sense.

Failover

- Failover is a system configuration that allows one server to assume the [logical] identity of a failing system.
- One important aspect of failover is available capacity, which can be handled in two ways:
 - Designing with extra capacity
 - Maintaining a stand-by server



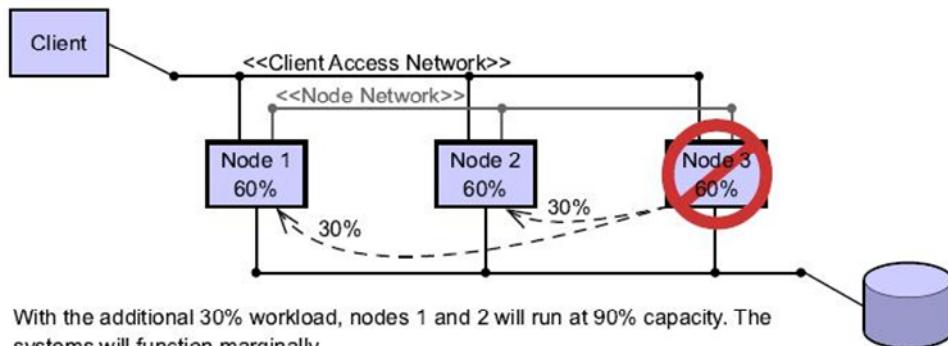
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Failover is another technique that you can use to minimize the likelihood of system failure. Failover is a system configuration that allows one server to assume the identity of a failing system within a network. If, at any point in time, a server goes down due to overloading, internal component failure, or any other reason, the processes and state of that server are automatically transferred to the failover server. This alternative server then assumes the identity of the failed system and processes any further requests on behalf of that system. One important aspect of failover is available capacity, which can be handled in two ways:

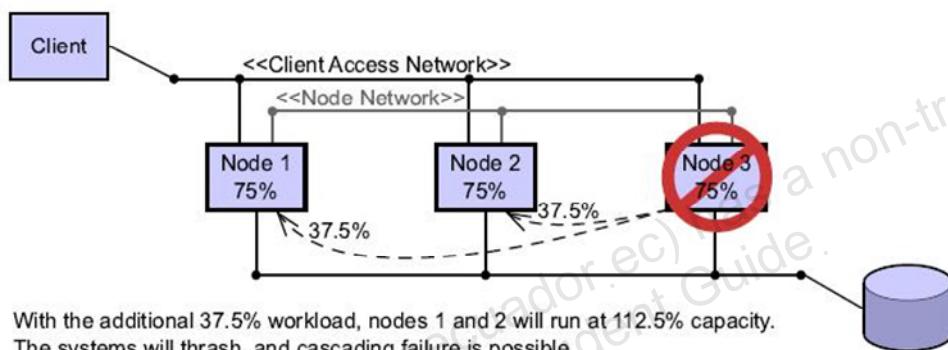
Designing with extra capacity: If you design a server group with extra capacity, all of the systems work for you, but at low usage levels. This means that you are spending money on extra computing resources that will not be used under normal load and operation conditions.

Maintaining a stand-by server: If you design a server group to have a stand-by server, you are spending money on a system that does no work whatsoever, unless or until it is needed as a failover server. In this approach, the money spent on unused computing resources is not the important thing to keep in mind. Instead, you should view the expenditure as insurance. You pay for the stand-by server and hope that you will never have to use it, but you can rest easier knowing that the stand-by server is there in case you ever need it.

Effects of Failover



With the additional 30% workload, nodes 1 and 2 will run at 90% capacity. The systems will function marginally.



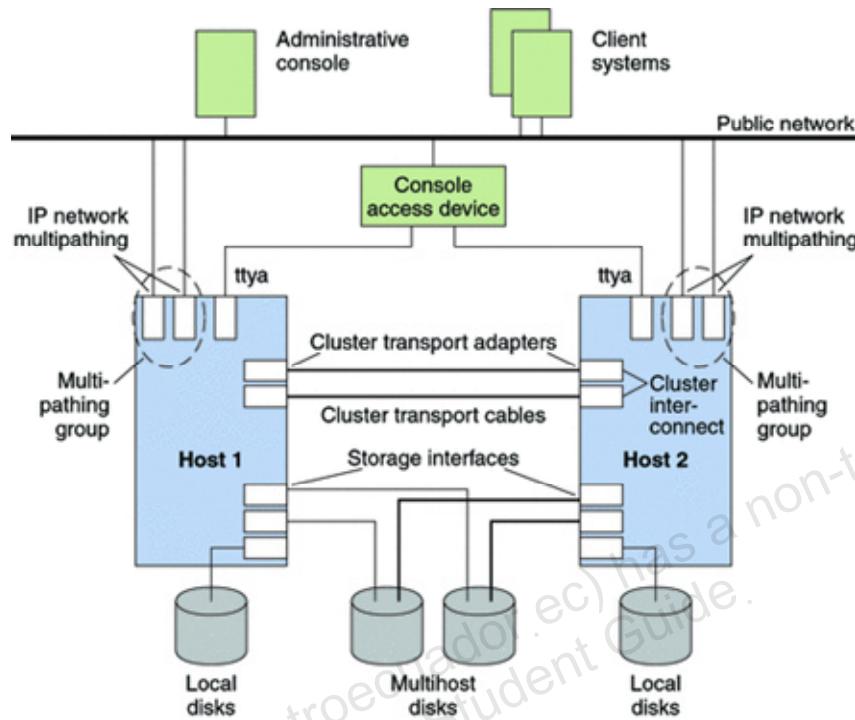
With the additional 37.5% workload, nodes 1 and 2 will run at 112.5% capacity. The systems will thrash, and cascading failure is possible.

ORACLE®

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

The slide shows how failover and capacity planning work together so that failover does not cause a cascading failure of the entire server collection.

Clusters



ORACLE

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

A cluster is a collection of loosely coupled computing nodes that provides a single client view of network services or applications, including databases, web services, and file services. In a clustered environment, the nodes are connected by an interconnect and work together as a single entity to provide increased availability and performance.

Highly available clusters provide nearly continuous access to data and applications by keeping the cluster running through failures that would normally bring down a single server system. No single failure, hardware, software, or network, can cause a cluster to fail. By contrast, fault-tolerant hardware systems provide constant access to data and applications, but at a higher cost because of specialized hardware. Fault-tolerant systems usually have no provision for software failures.

Clusters minimize the likelihood of system failure. Clusters provide high availability to system resources. Cluster software allows group administration, detects hardware and software failure, handles system failover, and automatically restarts services in the event of failure.

Common Cluster Configurations

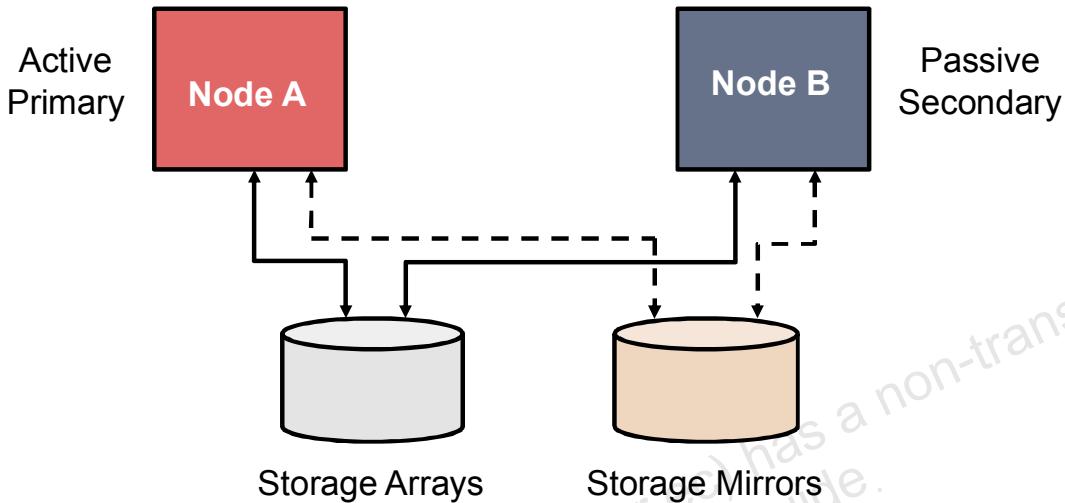
- Two-node clusters (symmetric and asymmetric)
- Clustered pairs
- Ring
- N+1 (Star)
- Scalable (N-to-N)



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

- **Two-node clusters (symmetric and asymmetric):** A configuration for which you can either run both servers at the same time (symmetric), or use one server as a stand-by failover server for the other (asymmetric).
- **Clustered pairs:** A configuration that places two machines into a cluster, then uses two of these clusters to manage independent services. This configuration allows you to manage all four machines as a single cluster. This configuration is often used for managing highly coupled data services, such as an application server and its supporting database server
- **Ring :** A configuration topology that allows any individual node to accept the failure of one of its two neighboring nodes.
- **N+1 (Star):** A configuration that provides N independent nodes, plus 1 backup node to which all the other systems fail over. This system must be large enough to accept the failover of as many systems as you are willing to allow to fail.
- **Scalable (N-to-N):** A configuration that has several nodes in the cluster, and all nodes have uniform access to the data storage medium. The data storage medium must support the scalable cluster by providing a sufficient number of simultaneous node connections.

Two Node Cluster: Asymmetric



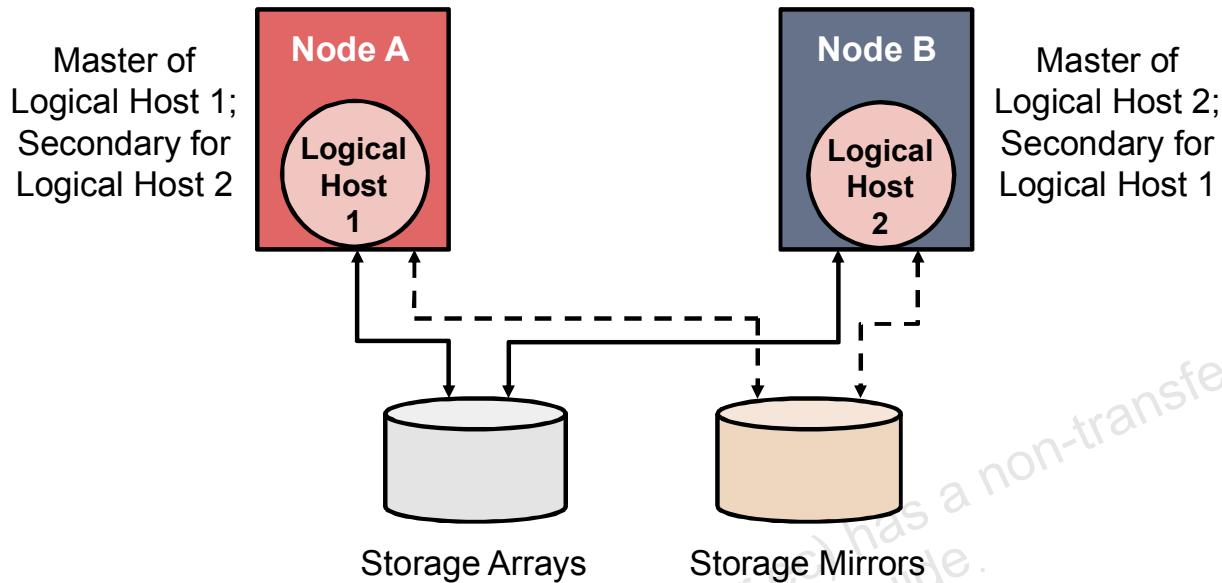
ORACLE®

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

One of the nodes will be primary and one will be secondary (backup). The backup node will become active only if there is a failover – in this situation, the backup node will master the logical host, start the application(s) and provide access to the application.

The advantage of an asymmetric configuration is the simplicity of its configuration, which makes administration easier. Asymmetric capacity planning is usually more straightforward because all that is required is to replicate the system requirements of the primary server onto the standby (backup) node.

Two Node Cluster: Symmetric

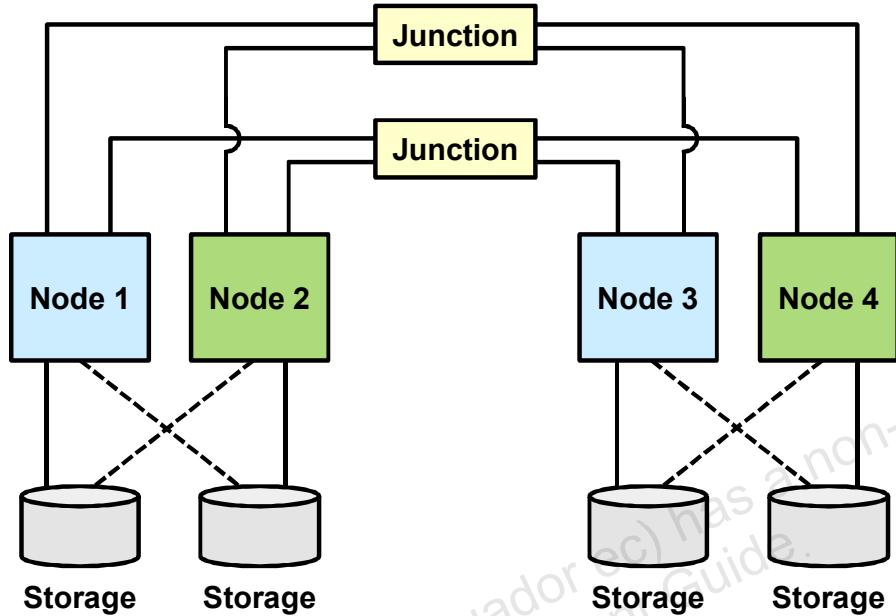


ORACLE®

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

Each server is configured with its own logical host(s), and each provides access to either of the databases hosted by the cluster. Each node is the master of its own logical host, and also acts as its as a standby for the other node. Configuring a symmetric HA configuration can be more challenging than configuring an asymmetric configuration. Application behavior must be fully understood because any node could be required to host multiple applications. The capacity of each node should be sufficient to handle the entire cluster workload. If this is not possible, users will have to settle for reduced levels of service. Decisions of this nature should be driven by the business no the technical requirements.

Clustered Pairs Topology



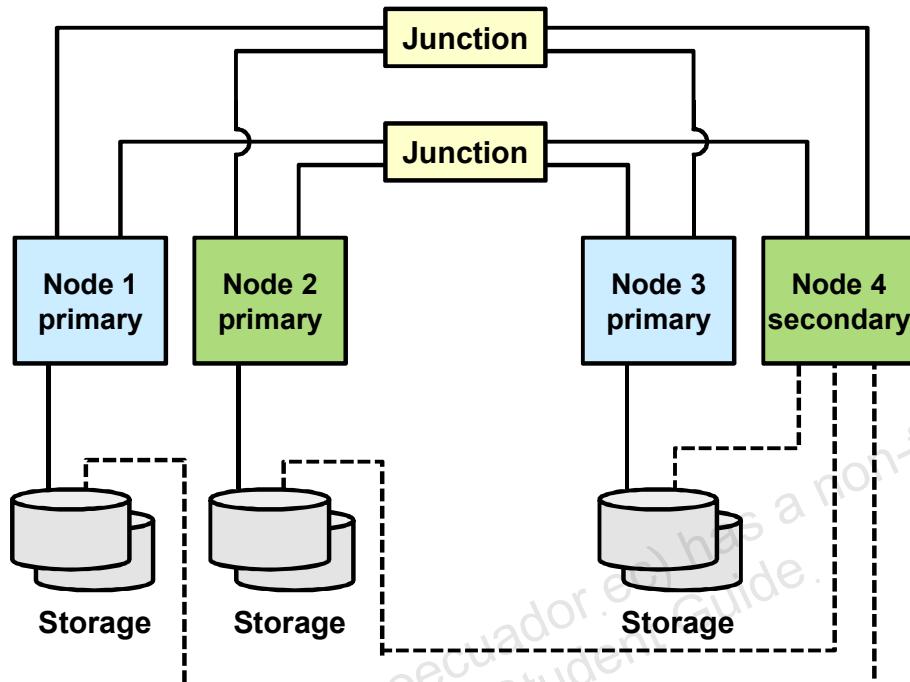
ORACLE

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

A clustered pair topology is two or more pairs of nodes that operate under a single cluster administrative framework. In this configuration, failover occurs only between a pair. However, all nodes are connected by the cluster interconnect and operate under Sun Cluster software control. You might use this topology to run a parallel database application on one pair and a failover or scalable application on another pair. Using the cluster file system, you could also have a two-pair configuration. More than two nodes can run a scalable service or parallel database, even though all the nodes are not directly connected to the disks that store the application data.

This configuration allows you to manage all four machines as a single cluster. This configuration is often used for managing highly coupled data services, such as an application server and its supporting database server.

N+1 Topology

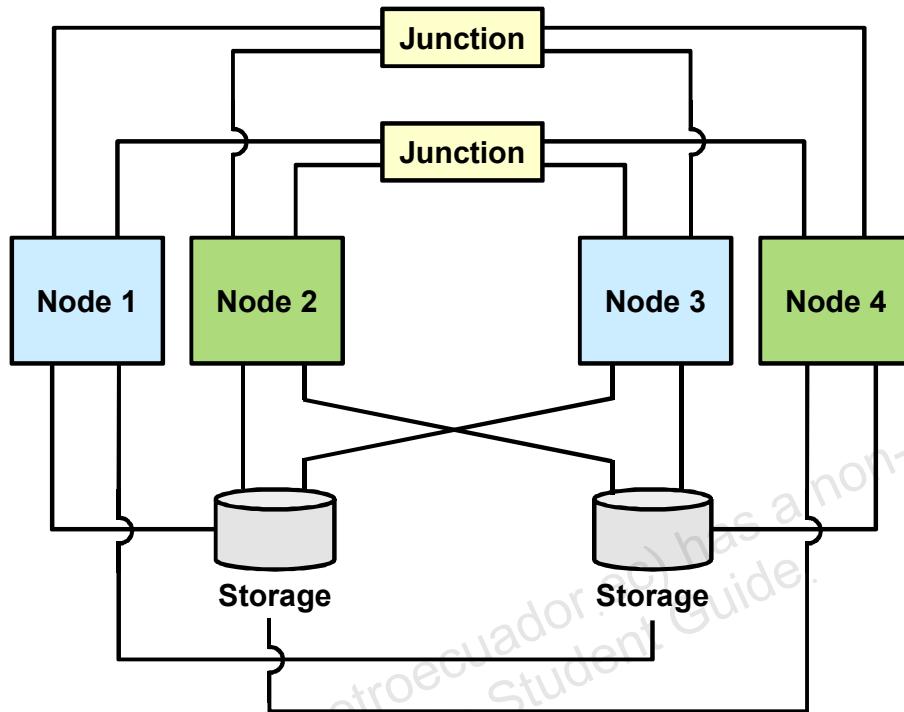


ORACLE®

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

An N+1 topology includes some number of primary nodes and one secondary node. You do not have to configure the primary nodes and secondary node identically. The primary nodes actively provide application services. The secondary node need not be idle while waiting for a primary to fail. The secondary node is the only node in the configuration that is physically connected to all the multihost storage. If a failure occurs on a primary, Sun Cluster fails over the resources to the secondary, where the resources function until they are switched back (either automatically or manually) to the primary. The secondary must always have enough excess CPU capacity to handle the load if one of the primaries fails.

N*N Topology



ORACLE®

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

An N*N topology enables every shared storage device in the cluster to connect to every node in the cluster. This topology enables highly available applications to fail over from one node to another without service degradation. When failover occurs, the new node can access the storage device by using a local path instead of the private interconnect.

Evaluating Replication Strategies

- You must weigh architectural concerns and find the most cost-effective approach to solving the problem.
- You might require feedback from the stakeholders.
- Consider the following factors:
 - Machine equivalence
 - Network design
 - Cost consideration
 - Cost of ownership
 - Initial outlay costs
 - Machine equivalence
 - Serializable issues with Java EE components

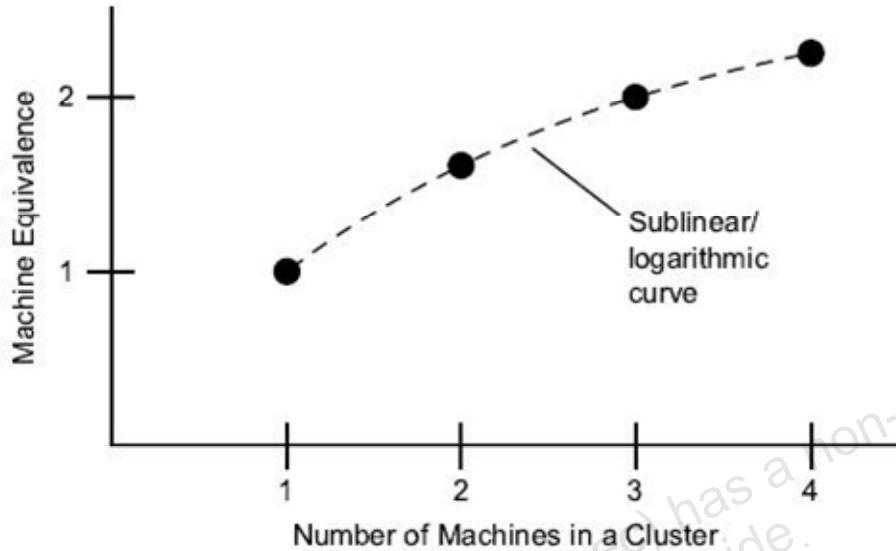


ORACLE

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

To properly select a replication strategy and implementation, you must weigh architectural concerns and find the most cost-effective approach to solving the problem. You might easily discern the most effective option, or you might require feedback from the stakeholders to achieve service levels that require additional project funds. Most of the time, load balancing and failover decisions are ones of performance and administration effort, so they do not require input from the stakeholders. It is usually the larger strategies that are associated with selecting server group and cluster configurations that require input from the stakeholders. Stakeholders should be considered as to the cost of the service, the time delay, and fees for licensed software required for replication.

Machine Equivalence



ORACLE

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

When you consider a replication strategy, you must account for machine equivalence. This slide illustrates that the machine equivalence effect can be modeled as a logarithmic curve. The sub-linear growth shown in the graphics is typically caused by additional overhead involved in horizontal scaling, including:

- Overhead of synchronizing the server instance state
- Overhead of additional network traffic
- Overhead of synchronizing the access to shared resources

As the slide demonstrates, achieving a certain capacity requires a larger number of physical machines than the capacity requirement specifies. You can use this relationship between capacity requirement and physical requirements to map performance goals to a physical deployment configuration.

Network Design

- Once a replication strategy is implemented, additional network traffic is typically involved.
- To accommodate this additional traffic, you should consider segmenting the network so that different types of network traffic are isolated into their own networks or subnets.
- Without additional consideration in the network design, this additional network traffic might have a negative impact on the performance of the system.



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

For example, the load balancer might need to perform a health check on each server instance to verify whether the server instance is still running.

- This type of additional network traffic does not occur in a single server instance deployment.
- You might consider a private network for replication or consider channels like T3 protocol.

Cost Consideration

- Each cluster configuration has a cost of ownership associated with it that is based on its reliability, availability, and serviceability.
- Together, these three architectural requirements are known as the RAS requirements for the cluster.
- The RAS requirements not only drive the cluster configuration that you select, they also contribute to the total cost of ownership and return on investment for the configuration you pick.



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

In addition, the cluster has an initial cost outlay that is greater than the cost of the individual systems. These costs include the additional hardware required to support the cluster and the cost associated with the additional skills that a cluster administrator requires beyond those of a normal system administrator. Although these costs add to the total cost of the cluster, they can be far outweighed by the costs incurred by not selecting a cluster solution.

Another cost issue associated with the cluster configuration is machine equivalence. When a group of machines are assembled into a cluster, a small amount of the cluster's resources are dedicated to matters other than processing application requests. The end result of this effect is that a cluster of two machines has lower utilization than a pair of machines working independently.

Improving Performance and Throughput

- Two key factors determine the system performance:
 - Processing time
 - Blocked time
- Common practices to improve performance:
 - Increase the system capacity.
 - Increase the computation efficiency.
 - Introduce cached data to reduce computation overhead.
 - Introduce concurrency.
 - Limit the number of concurrent requests.
 - Introduce intermediate responses.
 - Apply a timeout to the long-lasting operations.



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

Processing time: The processing time includes the time spent in computing, data marshaling and unmarshaling, buffering, and transporting over a network.

Blocked time: The processing of a request can be blocked due to the contention for resources, or a dependency on other processing. It can also be caused by certain resources not available, for example, an application might need to run an aggressive garbage collection to get more memory available for the processing.

Computation efficiency can be improved through the use of better software designs and patterns, a better JVM like JRocket, using a profiler and optimizing compiler.

While caching data can improve performance, there are associated issues of cost and uncertainty of success.

Improving Availability

- Factors affecting the system availability:
 - System downtime
 - Long response time
- Common practice to improve the system availability is through one of the following types of replication:
 - Active replication
 - Passive replication



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

System downtime: The system downtime can be caused by a failure in hardware, network, server software, and application component.

Long response time: If a component does not produce a response quick enough, the system can be perceived to be unavailable.

The most common practice to improve the system availability is through one of the following types of replication, in which redundant hardware and software components are introduced and deployed:

Active Replication: The request is sent to all the redundant components, which operate in parallel, and only one of the generated responses is used. Because all the redundant components receive the same request and perform the same computation, they are automatically synchronized. In active replication, the downtime can be short because it involves only component switching.

Passive Replication: Only one of the replicated components (the primary component) responds to the requests. The state of other components (secondary) are synchronized with the primary component. In the event of a failure, the service can be resumed if a secondary component has a sufficiently fresh state.

Improving Extensibility and Flexibility

- The need for extensibility and flexibility is typically originated from the change of requirements.
- Consider the following practices for the system extensibility and flexibility:
 - Clearly define the scope in the Service Level Agreement.
 - Anticipate expected changes.
 - Apply patterns to the system architecture.



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

The need for extensibility and flexibility is typically originated from the change of a requirement. One of the most important goals of the architecture is to facilitate the development of the system that can quickly adapt to the changes. When you create the architecture, consider the following practices for the system extensibility and flexibility:

- **Clearly define the scope in the Service-Level Agreement:** Scope change is one of the most common reasons for project failure. Defining a clear scope is the first step to limiting unexpected changes to a system.
- **Anticipate expected changes:** You should identify the commonly changed areas of the system, for example, the user interface technology, isolate these areas into coherent components. By doing this, you can prevent ripple effects of propagating the change across the system.
- **Design a high quality object model:** The object model of the system typically has an immediate impact on its extensibility and flexibility. Therefore, you should consider applying essential object-oriented (OO) principles and appropriate architectural and design patterns to the architecture. For example, you can apply the MVC pattern to decouple the user interface components from the business logic components.

Improving Scalability

Scalability can be configured in two ways:

- Vertical Scalability:
 - Add more processing power to an existing server system.
 - Replace an existing server with more capable system.
 - Transparent to system architecture.
 - Bound to the physical limitation of a server system.
 - Cost of buying more powerful hardware is high.

Horizontal Scalability:

- Add additional runtime server instances.
- Does not have the physical limitation.
- Affects the system architecture.
- You must account for machine equivalence.



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

Vertically scaling a system is transparent to system architecture. However, the physical limitation of a server system and the high cost of buying more powerful hardware can quickly render this option impractical. On the other hand, horizontally scaling a system does not have the physical limitation imposed by an individual server's hardware system.

Another consideration you must take into account is the impact that horizontal scaling has on the system architecture. Typically, to make a system horizontally scalable, not only do you need to use a software system that supports the cluster-based configuration, you also need to design the application such that the components do not depend on the physical location of others.

Consider the trade offs of multiples boxes versus having multiple JVMs on the same box.

Virtualization

- Advantages:
 - Server consolidation
 - Fewer physical boxes
 - Reduced costs of hardware
 - More flexible and agile
 - Easier deployment and testing
- Issues:
 - License and software costs
 - Capacity planning
 - Training and support
 - Management
 - Initial hardware investment
 - Support for legacy systems



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

Virtualization decouples the hardware from the software being used. Therefore, many different server environments can be run on a single physical machine. Virtualization adds a thin layer over the physical server itself. The underlying physical hardware is seen as a pool of resources that all the virtual machines can share. Each virtual server is seen as its own entity, with the exception of sharing the pooled resources. With this separation, problems with one application do not affect the other virtual servers.

Advantages:

- **Server consolidation:** Leverage the power of a piece of hardware more fully. InfoTech estimates that distributed physical servers may use only 20% of their capacity.
- **Fewer physical boxes:** Virtualization decreases the number of physical devices. This results in a smaller data center, less power consumption and heat produced.
- **Reduced costs of hardware:** Because of consolidation, less physical devices are required.
- **Quick creation of new OS environments:** New guest OSs can be created quickly for running applications in parallel, testing, migration and restoring systems in a disaster recovery scenario.

- **Easier deployment and testing:** Using virtual servers can speed up development and testing because it is easier and faster to create new OS environments. It also enables developers to test and compare application performance across different server environments and configurations. You can stress test on more machines than you physically have.

Issues:

License and software costs: Each server environment requires its own license.

Capacity planning: Matching the right number of servers with the capacity of the hardware

Training and support: These costs must be added into the plan.

Management: Virtual servers still require management.

Initial Hardware investment: Consider the trade-off between the cost of new hardware and impact of using less powerful hardware.

Support for legacy systems: There may be issues associated with virtualization. For example, a firewall may expect a single IP address for specific piece of hardware.

The virtual machine is still a “machine” and has constraints. Ensure there is a good fit of the virtualized OS with the underlying hardware. Does the host hardware support the guest OS capabilities and requirements? For example, the guest OS may support 64 gig of RAM, but if the underlying host does not have this much memory available it will constrain the guest. Or, if the host machine has only one network card for several guest OSes, then the single network card must handle all the traffic.

CASE STUDY: ORACLE UNIVERSITY

Oracle University is responsible for delivering education and training classes on Oracle products. Most Oracle classes require one or more servers configured with the product under study for demonstration purposes and hands-on activities.

The result was significant problems with power and space. The Oracle University data center simply could not add any more physical servers to meet growing classroom demands. An additional problem was that it could take as much as 48 hours to fully provision a new classroom environment. That is a long time when you are hosting multiple events per week in locations all over the world. Provisioning limits were actually hampering the ability of Oracle University to grow.

MOVING TO VIRTUALIZATION

The solution for Oracle University was a fully virtualized Grid environment utilizing Oracle VM for server virtualization and NetApp for storage virtualization. Oracle University relies on a “stateless” model in which no particular physical device is essential. Virtual machines (running under Oracle VM connect through virtual private network (VPN) connections to back-end storage.

The first step in migration was to move to reliable network storage. Having the whole server environment on network storage made it possible to quickly move that environment from one physical server to another. The addition of server virtualization with Oracle VM supercharges that capability by making it possible to carve up the resources of a single physical server for use by multiple VMs at one time, leading to a significant reduction in overall server count.

The combination of Oracle VM and network storage makes it possible to move a running VM from one physical machine to another while the VM is in use. Oracle chose NetApp storage for this deployment because of key technical capabilities. Also, the level of service that NetApp was able to provide was head and shoulders above other vendors, and was key to selling the concept of virtualized storage in the Oracle University organization.

JVMs

Why 64-bit JVMs?

- The 64-bit JVM allows for a heap greater than 4 GB.
- There should be fewer (though longer) garbage collections.
- There is more address space, so there can be significantly more threads and a larger stack.

Why not 64-bit JVMs?

- Certain client-oriented features like Java Plug-in and Java Web Start are not supported.
- Native code needs to be compatible (For example, JNI for things like Type II JDBC drivers).

ORACLE

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

32 versus 64-bit JVMs

There may be either a slight improvement (due to certain hardware optimizations on certain platforms) or minor degradation (due to increased pointer size). In memory databases or search engines that can use the increased memory for caching and avoid IPC or disk accesses will see the biggest application level improvements.

The 64-bit JVM can require higher levels of application tuning to reduce garbage collection times.

Questions to ask:

- What problems do you think you will solve with a 64-bit JVM? For example, it will not solve memory leaks.
- Does my application really need, that is will it really benefit from, a 64-bit JVM?
- Does the underlying hardware have enough RAM to take advantage of the 64-bit JVM larger memory space?

Oracle JRockit

The Oracle JRockit JVM is a high-performance JVM. The JRockit JVM provides improved performance for Java applications deployed on Intel 32-bit (Xeon) and 64-bit (Xeon and SPARC) architectures at significantly lower costs to the enterprise. Further, it is the only enterprise-class JVM optimized for Intel architectures, providing seamless interoperability across multiple hardware and operating system configurations. The JRockit JVM enables your Java applications to run optimally on Windows and Linux operating systems (both 32-bit and 64-bit architectures). The JRockit JVM is especially well-suited for running Oracle WebLogic Server.

Note: Oracle will merge JRockit JVM and Hotspot JVM in 2011.

Packaging and Deployment Considerations

- Consider a packaging and deployment scheme that will support your installation and scalability requirements.
- For example, Oracle recommends for WebLogic:
 - For development purposes, use a split development directory structure that separate source files and output files.
 - For production purposes, use an exploded (unarchived) directory format. This format enables you to update files without having to redeploy the application.



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

Consider the scope and maturity of the organizations that are impacted by deployment plans. Assess the gaps and define a resolution approach. Clearly define roles and responsibilities for implementation and deployment. Identify any deployment issues and be prepared to make recommendations on how to handle them. Identify the discrete packages of functionality that will address identified gaps. Consider how best to leverage these packages across projects.

Suggestions for your deployment plans:

- Define an overall deployment solution.
- Ensure deployed solutions conform to the architecture.
- Define and activate operations to support the deployment solution.
- Initiate a phased deployment schedule in alignment with business needs and priorities.
- Identify the resources and skills needed to support the deployment plans.

Testing

- Consider developing an enterprise-wide testing strategy.
- Issues of enterprise software testing:
 - Complex technology landscape: what to test?
 - Broader scale
 - Integration challenges
 - Product upgrades and interim releases
 - Regulatory compliance
 - Need to test functionals as well as nonfunctionals
 - Consider building or buying a testing framework
- Stress testing:
 - Do you actually perform it?
 - How do you know the system will respond as planned for sure?



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

The needs of enterprise testing are different than application testing. Issues of collaboration, agility for constant changes, coping with lack of clear requirements, distributed environments, fuzzy roles and responsibilities all contribute to the challenge.

Some things to keep in mind:

- The QA team needs agility to respond to and support (sudden) changes.
- Two-thirds of US companies are only at CMM1.
- Testing needs to support distributed and parallel efforts.
- Frameworks can be difficult to test; to isolate the aspects to be tested.
Enable the testers to leverage new technologies
- Create mixed talent teams.

Oracle e-TEST Suite:

Application testing is becoming increasingly important as business seek to deploy applications faster, at lower cost and with higher quality of service. Business are looking for automated application testing solutions that help prevent costly application performance problems, avoid unplanned outages of business-critical applications, and automate the manual steps involved in application testing.

The Empirix e-TEST suite products address this need by helping validate the functionality and scalability of applications, enabling businesses to rapidly deliver higher quality applications at reduced cost.

Lesson Agenda

- Examining nonfunctional requirements (NFRs)
- Common practices for improving qualities
- **Prioritizing Quality-of-Service requirements**
- Inspecting QoS requirements for trade-off opportunities



ORACLE

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

Prioritizing Quality-of-Service (QoS) Requirements

- The quality-of-service requirements specify the required or desired values for nonfunctional requirements:
 - Evaluate QoS requirements on some common basis.
 - Work with the stakeholders to prioritize the requirements.
- Variations in priority can give dramatically different solutions to system architecture.



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

Evaluate QoS requirements on some common basis, such as monetary value. However, the monetary value of a QoS requirement is not the only factor that impacts your decisions. Project stakeholders might provide input based on their desires and expectations, rather than the cost of mitigating risk. This input can change the perspective on QoS requirements. These stakeholder expectations affect the value of the QoS requirements, but they do not map neatly into cost the way risk mitigation does.

After you identify and define the QoS requirements, work with the stakeholders to prioritize the requirements. Some requirements have high priority due to the nature of the application domain. Other requirements have high priority because of the desire of the stakeholders to achieve a certain marketability level or to obtain an advantage over competitors. As always, there is risk associated with realizing a system that can meet the requirements. Prioritization incorporates the risk of satisfying stakeholders into the risks of managing complexity.

Variations in priority can give dramatically different solutions to system architecture. For example, consider two systems (X and Y) designed to provide users with a Web portal to similar applications. System X is designed with a higher priority on performance, while the system Y is designed with a higher priority on security.

The extra services of managing security in system Y forces it to either perform at a lower level than system X, or requires more horsepower to achieve a performance level similar to system X.

In either case, the priority of QoS requirements affects the final architecture in each system so that each system is distinctly different from the other.

System Design Considerations

- Examine a project to determine which requirements, or categories of requirements, are most important.
- Examples:
 - A project is computationally demanding but has a fixed number of users.
 - A project expects the user base to grow at a large but difficult-to-determine rate.
 - As the application evolves, users get more remote, thus effecting latency and performance
- Consider using feedback forms and support logs to gain feedback from users.

ORACLE

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

You should examine a project to determine which qualities, or categories of qualities, are most important. Then use the dimensions to generate a design outline of the system that maximizes the benefit to these qualities.

For example, if a project is computationally demanding, but you expect it to have a fixed number of users, capacity would be an appropriate way to provide the needed computational power. Consequently, the system design is driven by making performance and throughput a higher priority.

In contrast, if a project has a few users at first, but you expect the user base to grow at a large but difficult-to-determine rate as the service gains popularity, a more appropriate approach would be redundancy, modularity, or possibly a combination of the two. In this case, because scalability has the higher priority, it drives the design.

As you can see, priority of a particular requirement yields dramatically different system configurations. Consequently, prioritization of requirements becomes an important step in creating the architectural prototype. The prioritization of requirements is driven by several factors, including problem domain, stakeholder interest, and risk.

Ranking QoS Requirements

- Based on the business domain:
 - Online shopping
 - Online banking
 - Online brokerage
- Based on customer expectations:
 - Stakeholders require a level of quality from the application that is based on their experiences with similar applications.
 - If the quality of the system does not meet customer expectations, the project can be considered a failure.
- Consider using Must Have Should Have Could Have and Won't Have (MOSCOW) to assess user requirements.



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

Each business domain places constraints on QoS requirements, which forces some to be invariant (*must haves*) and others to be more flexible (nice to have, but can get by without). You can make initial priority assignments based on your experiences from similar problem domains and by consulting business domain experts. For an example of how the business domain impacts QoS requirements, consider the differences between the domains of online shopping, banking, and stock trading.

In the online shopping application, reliability and security are not as critical as they are in online banking or brokerage. In the online shopping application users typically browse through catalogs, examine detailed views of items, and add items to a shopping list. This data is not highly sensitive, nor are the operations in need of high reliability because they can be easily reproduced. Reliability and security only become important in the shopping application when the user is executing a financial transaction to make a purchase.

In the banking application, reliability and security become more important than in online shopping. Most, if not all, user transactions contain highly sensitive financial information regarding customer accounts. User actions usually involve the movement of money, and failure of application operations must not be allowed to cause a loss of customer funds.

Performance, on the other hand, is not as critical as it is in online brokerage applications because banking transactions are usually processed only once per day, although some banks offer instantaneous service for some transactions, such as transfer of funds.

In the online brokerage application, performance is just as important as reliability and security. Not only are you dealing with sensitive financial information, but stock trading is handled in near-real time. Seconds count and can make a difference of thousands of dollars. Also, user actions must be performed with some guarantee of completion because when a trade is believed to be executed, it must be executed. Failure to execute trades can result in huge financial losses in the form of legal action.

Prioritization of QoS Requirements

| Category | Quality | Priority | Risk | Mitigation |
|---------------|-----------------|--------------|-----------------------|-------------------|
| Manifest | Performance | Show stopper | System is too slow | Increase capacity |
| | Reliability | | | |
| | Availability | | | |
| | Usability | High | System is too complex | Simplify UI |
| Operational | Throughput | Show stopper | Insufficient capacity | Add cluster nodes |
| | Manageability | | | |
| | Security | | | |
| | Serviceability | | | |
| Developmental | Realizability | | | |
| Evolutionary | Scalability | Low | Faster growth rate | Plan scalability |
| | Maintainability | | | |
| | Flexibility | | | |
| | Reusability | | | |

Rank QoS Requirements Based on Customer Expectation



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

Stakeholders have a vested interest in the outcome of the project. The desires and expectations of the stakeholders impact the priority of the QoS requirements. Stakeholders require a level of quality from the application that is based on their experiences with similar applications. If the quality of the system does not meet expectations, the project can be considered a failure.

For example, consider the impact that the stakeholders' performance expectations have on a system infrastructure. Performance has a limit that is not only associated with the capabilities of a given hardware system, but also with the tolerance of the user to wait for results to arrive. If the tolerance of the user is exceeded by the system, and the system is operating at its maximum capacity, then the quality of the system is deemed to be inadequate. In this case, the priority of performance outweighs the system infrastructure, and more funds must be allocated to increase system capacity and to improve performance, even though other QoS requirements allowed for a smaller infrastructure.

This scenario is exemplified in the partial risk list, shown in the slide.

Reviewing Quality Estimation

- Present the estimation to the stakeholders for validation.
 - Allows the stakeholders to make choices to proceed with requirement values or to change them based on current architecture cost.
 - Aligns the stakeholder perspective of the project and the project goals with the actual project deliverables.
- Use the architectural prototype to validate the ability to achieve QoS goals.
- Consider using an Oracle Reference Architecture for the architectural prototype.

ORACLE

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

Review is an important part of any iterative process. As the quality estimates are made more concrete, you need to present them to the stakeholders for validation. This empowers the stakeholders to make choices to proceed with requirement values or to change them based on current architecture cost. The review cycle aligns the stakeholder perspective of the project and the project goals with the actual project deliverables.

The evolutionary prototype not only validates architectural decisions, it also validates the ability to achieve QoS goals. The results obtained from implementation and testing are used to clarify or validate QoS requirements during the next iterative cycle. This feedback from actual system development to requirements specification demonstrates the importance of an iterative process as a risk management tool.

Revising QoS Values

- Adjust requirement values to more realistic values when you have one or more of the following issues:
 - Budget constraints
 - Time constraints
 - Excessively high complexity
- Adjust the architecture when you have one or more of the following issues:
 - Lack of available technology
 - Need for redistribution of services
 - Improvements required for flexibility or manageability



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

After the stakeholders review the estimates of QoS requirements, you record any changes in the project documentation. Use these changes to drive the next iterative cycle, and eventually, the evolutionary prototype. As you modify the prototype, the ability of the system to fulfill requirements becomes known, and you can identify and control the project risks. Use test programs and test harnesses to obtain values for the prototype. Measure values, such as performance, scalability, and availability, directly from the prototype. Measure values, such as flexibility and manageability, when you make changes to the prototype.

Lesson Agenda

- Examining nonfunctional requirements (NFRs)
- Common practices for improving qualities
- Prioritizing Quality-of-Service requirements
- Inspecting QoS requirements for trade-off opportunities



ORACLE

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

Inspecting QoS Requirements for Trade-Off Opportunities

- The architect should have the ability to select an optimal solution from several options.
- To make trade-off decisions
 - Identify invariant requirements: requirements that cannot be compromised
 - Compare remaining requirements for trade-offs



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

Much of the art of architecture is really the art of engineering trade-offs, or the ability to select an optimal solution from several options. This requires knowledge of the economics of engineering and the ability to optimize the quality characteristics of the system. Because system optimization is a problem with multiple variables, and these variables have many interdependencies, finding an optimal solution is difficult. Awareness of which requirements are variable and which are invariant can improve your efforts to find an optimal solution.

Identify Invariant Requirements: Some QoS requirements cannot be compromised. Therefore, it is a good idea to create a list of requirements that have invariant values. If the system fails to meet these operation levels, the project will fail. These requirements might fulfill a critical business need, or they might provide a serious market advantage over competitors. In either case, these invariant requirements establish the initial scope of the project and justify its continued development.

- **Compare Remaining Requirements for Trade-offs:** After you identify the invariant requirements, you can use any QoS requirements that remain to perform the necessary trade-offs to optimize the system. As stated previously, it is important that you quantify these requirements in equivalent terms. You can use implementation cost in present value to do this. The requirements might not always allow for a realizable, much less optimal, solution.

In these cases, one requirement must take precedence over another. These trade-off opportunities allow the stakeholders to control project costs and, at the same time, achieve as many of the project goals as possible. You can incorporate any changes to the requirement values into the iterative development cycle, which controls risks and costs simultaneously.

Additional Resources

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

- Marcus, Evan and Hal Stern, *Blueprints for High Availability, Second Edition*. New York: John Wiley and Sons, 2003.
- Cockcroft, Adrian and Bill Walker. *Capacity Planning for Internet Services*. Sun Microsystems, 2000.



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

Quiz

The key focus of an Enterprise Architect is the NFR's. Which of the following is not an NFR?

- a. A page displays in less than 2 seconds under full system load
- b. 99.99% uptime
- c. Date and time are displayed to the user in European and Military style
- d. The system must determine if the user has sufficient funds in their account



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

Answer: d

a, b, c are all examples of quality of service requirements.

Quiz

Given a situation in which the system will experience significant variability of load depending upon the time of year (like post-Thanksgiving to pre-Christmas), which of the following is the best NFR to focus on?

- a. Reducing redundancy in the architecture
- b. Implementing a load-balancer
- c. Adding additional fail-over devices
- d. Implementing an asymmetric two-node cluster



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

Answer: b

Quiz

Given a requirement from the IT department for a replication strategy, what do you need to account for?

- a. Cost
- b. Additional security
- c. Machine Equivalence
- d. Electrical power



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

Answer: c

Quiz

With a mandate to improve system availability, what is a common practice you can apply?

- a. Increase concurrency
- b. Apply patterns to the system architecture
- c. Add additional runtime server instances
- d. Active replication



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

Answer: d

Quiz

Virtualization has become a hot topic – what are some trade-offs to consider? (Choose all that apply.)

- a. License costs
- b. Disk storage required
- c. Capacity Planning
- d. Security



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 systemic qualities of an enterprise application
- Describe common practices for improving systemic qualities
- Prioritize quality-of-service (QoS) requirements
- Inspect for trade-off opportunities
- Study the impact of dimensioning on nonfunctional qualities



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

Practice 4 Overview: Consider the impact of Nonfunctional Requirements

This practice covers the following topics:

- Consider the impact of nonfunctional requirements on your architecture.
- Identify ways to address the nonfunctional requirements.



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

In this practice, you will use the architecture you created previously and consider how it can cope with risks and also how to address nonfunctional requirements.

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.

Defining Common Problems and Solutions



Part 1: Risk Factors and System Flexibility

ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Identify key risk factors in distributed enterprise systems
- Design a flexible object model

Discussion Questions

- What aspects of a project are common sources of risk, and why? How can you manage these risks?
- What aspects of object-oriented design are pertinent to architecture? How do these aspects help control risks?



ORACLE

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

Lesson Agenda

- Identifying key risk factors
- Designing a flexible object model

Identifying Key Risk Factors

- A primary focus of architecture is controlling risks.
- Although risk can arise in many ways, the following areas regularly cause difficulties in distributed enterprise computing systems:
 - System flexibility
 - Network communication and layout
 - Transaction model
 - Security model
 - System sizing and planning



ORACLE

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

The goal of the architecture is to identify problems in these areas early, and provide a implementable blueprint that provides a solution to mitigate the risks associated with these problems.

This section looks at some of the main recurrent risks in distributed enterprise computing systems and how to control those risks.

What Do You Think?

“The hard problems in distributed computing are not the problems of how to get things on and off the wire. The hard problems in distributed computing concern dealing with partial failure and the lack of a central resource manager. The hard problems in distributed computing concern insuring adequate performance and dealing with problems of concurrency. The hard problems have to do with differences in memory access paradigms between local and distributed entities.”

A Note on Distributed Computing by Samuel C. Kendall, Jim Waldo, Ann Wollrath, and Geoff Wyant



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

System Flexibility

- Why is system flexibility so important?
- How to increase the system flexibility?



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

Flexibility is important because it is inevitable that system requirements change over time. In fact, system requirements almost always change before the completion of a project. A major rework is expensive and threatens the project as a whole.

In general, applying appropriate design principles and patterns increases the flexibility of a system. For example, a component model based on the loosely-coupled principle makes changing the component implementation easier. Based on good design principles, a pattern typically provides a concrete structure and guidelines to solve a problem. Therefore, applying patterns allows you to reuse proven experiences.

Network Communication and Layout

- Why is the network model so important?
- How to design an efficient network model?



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

The network is typically the slowest part of any distributed computing system. Consequently, any unnecessary, or poorly timed network communication has a devastating effect on the system's overall performance.

To design for efficient use of networks, you must think about how the system is laid out and how the elements must talk to each other. Your goal is to minimize the number of round trip messages that are sent over the network.

Transaction Model

- Why are transactions so important?
- How to design an efficient transaction model?
- Are transactions always required?



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

Transactions are an important tool for maintaining data integrity in a system that permits concurrent use. Unfortunately, transactions also limit concurrency, and it is easy to reduce a system's throughput dramatically by inappropriate use of transactions.

Use transactions with care to ensure that they have sufficient, but not excessive scope. In some situations, you should avoid the use of transactions, or reduce their scope to the point that they do not entirely protect the data. In this situation, known as optimistic locking, be sure to include code that repairs the damage that might arise if concurrency happens to cause trouble.

Security Model

- What is a good security model?
- How to create a good security model?



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

Due to the increasing importance of security in a distributed system, it is essential to create a security model, at the architectural level. A good security model for the system should be cost effective, easy to implement, easy to use, and compliant with the company security policy.

The security model should be created based on both the system infrastructure and application components. You can rely on infrastructure components, such as firewalls and other security devices to provide an environment-level protection for the system. You should design the application component model to work with the infrastructure by considering appropriate component deployment layout, communication protocols, and other security mechanisms, such as user authentication and authorization.

Java EE technology addresses problems in all of these areas. You can use either declarative or programmatic approaches to specify flexible transaction and security behavior of a Java EE technology system. The declarative model simplifies the overall development and proof of these aspects of a system. Because the Java EE technology is component based, all the flexibility of good object-oriented design is available to Java EE technology projects. Further, Java EE technology works with standard design patterns, such as the Gang of Four patterns, and additional patterns, such as the Java EE patterns described by Alur et al., have been developed to address Java EE technology-specific issues.

System Sizing and Planning

- The practice of system sizing typically requires you to make a lot of trade-off decisions, because given a projected budget, it is typically impossible to design a system that can satisfy all the QoS requirements.
- You should be able to prioritize the requirements, and design the size of the system that can satisfy show-stopper requirements.

 ORACLE

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

Many nonfunctional qualities, such as performance, availability, and scalability depend on the capacity and redundancy dimensions of the system. To deliver the acceptable quality of service, you must address the issues of system sizing and deployment at the architecture level. Another factor you should consider in planning the system is to select the appropriate application servers and other service provider components. A variety of application server products are available in both the proprietary and open-source forms. You should select these products based on factors, such as cost, product quality, and the vendor's support services.

Note: Even though the Java EE technology does not mandate an application to provide enterprise-level features, such as clustering and load balancing, application server products typically support these. These features allow you to start with sizing the system to meet the initial load requirement, and scale up the system as the system load increases.

Estimation Best Practices

“The most unsuccessful three years in the education of cost estimators appears to be fifth-grade arithmetic.”

- Norman R. Augustine

- How do you estimate for software architecture and design?
- The best architects and developers should be part of the metrics teams.



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

“Numerous aspects of estimation are not what they seem. Many so-called estimation problems arise from misunderstanding what an “estimate” is or blurring other similar-but-not-identical concepts with estimation. Some estimation practices that seem intuitively useful don’t produce accurate results. Complex formulas sometimes do more harm than good, and some deceptively simple practices produce surprisingly accurate results.” *McConnell, Steve. Software Estimation: Demystifying the Black Art.* Microsoft Press 2006.

Lesson Agenda

- Identifying key risk factors
- Designing a flexible object model

Designing a Flexible Object Model

“Software systems perform certain actions on objects of certain types; to obtain flexible and reusable systems, it is better to base their structure on the objects types than on the actions.”

- Bertrand Meyer

To design a flexible system, you should have a good understanding of the object models used in a distributed system.

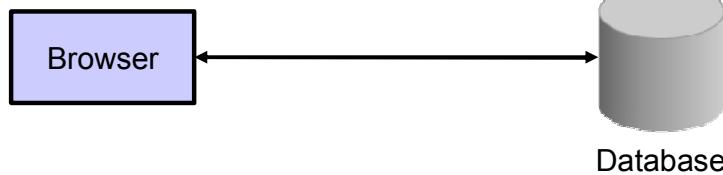


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

The elements that make up the architecture are components and systems. The components might be considered as large-scale, high-level objects. Consequently, to design a flexible system, you should have a good understanding of the object models used in a distributed system. You can apply the usual principles, patterns, and frameworks to the components. In the remainder of this lesson, we'll look at how to design a flexible system, using patterns and good object-oriented design principles.

Using Abstractions

Example 1



Example 2



Example 3



ORACLE

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

When you create an architecture, you also create symbols that represent the elements of the architecture to the development team. Abstraction allows the team to form the big picture before they discuss the details. Apply context and perspective to your drawings and then choose how to communicate the architecture to other members of the team.

The process of abstraction can take many forms. It can be a whole system, such as a Web server, or it can have finer granularity at the component level. The idea is to introduce appropriate abstractions that have small, simple, and clear interfaces to decrease dependency and tight coupling between entities.

Applying Object-Oriented Principles

- Open-Closed principle
- Dependency Inversion principle
- Interface Segregation principle
- Composite Reuse principle and Separation of Concerns principle

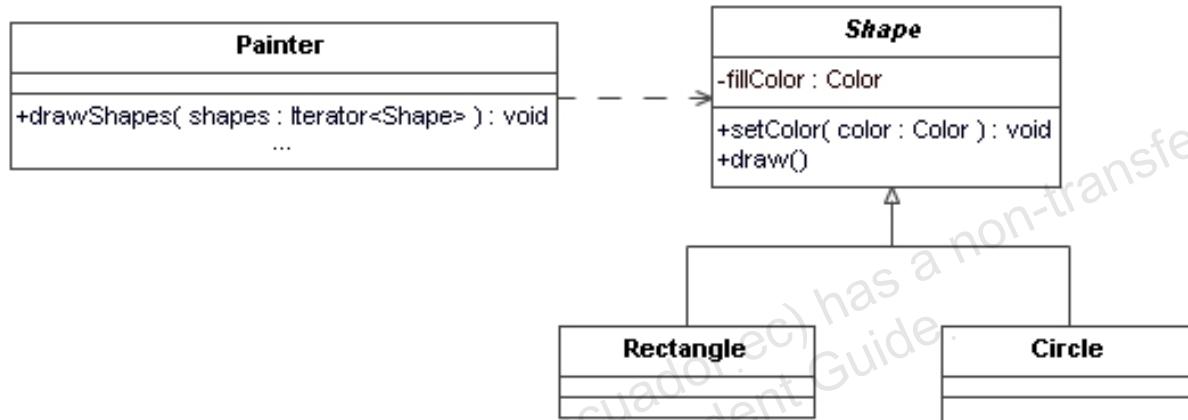


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Many software design principles underline the object-oriented design heuristics. Applying these principles at the architectural level generally improves the flexibility of the system.

Open-Closed Principle

Classes or components should be open to extension but closed to any modification that breaks clients.



ORACLE

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

In object-oriented design, common design heuristics, such as making member variables of a class private and designing dependencies on interfaces are examples of applying this principle.

The slide shows an example of applying the Open-Closed principle. In this example, the `fillColor` attribute of the `Shape` class is declared as private, so that changing its implementation is limited to the `Shape` class only, and it does not affect all the subclasses of `Shape`. If the `fillColor` attribute is visible to subclasses, the subclasses could be implemented to use the attribute directly. In this case, changing the implementation of the `fillColor` attribute in the `Shape` class could immediately have a ripple effect to its subclasses.

Additionally, the `drawShapes` method of the `Painter` class depends only on the `Shape` class. This allows additional subclasses of `Shape` to be created without changing the implementation of the `Painter` class.

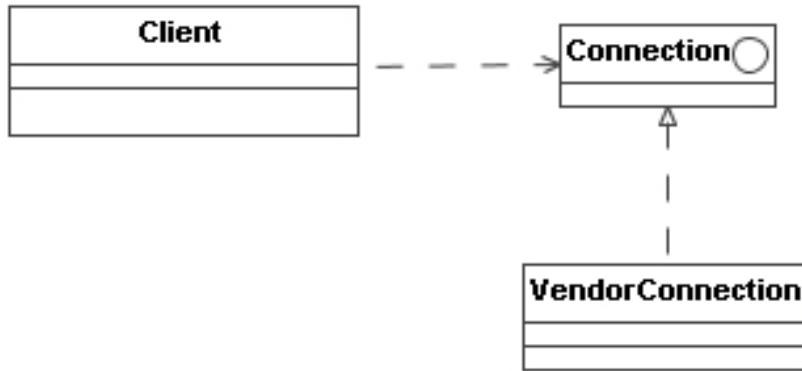
You can achieve much of the benefit that the Open-Closed principle gives by programming to a carefully designed interface. The interface takes into account the essential nature of the task, and so it is reasonably likely to remain stable.

In most programming systems, the Open-Closed principle can be implemented using the following approach:

- Create well-encapsulated functionality
- Limit (by discipline if not by compiler rules) access to particular methods chosen for publication

When you enforce this approach, you help to ensure that changes in one part of a system have minimal impact on other parts of the system, which is the primary goal of encapsulation.

Dependency Inversion Principle



ORACLE

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

You should design and code in terms of abstractions, not in terms of concrete implementations. For example, when you design components that are accessible by clients, you should consider designing a layer of component interfaces (see the Open-Closed principle again) to enable their use. In this type of an approach, it is easy to change the implementation without changing the interface. Therefore, the client is unaffected by the change.

An example of applying the Dependency Inversion Principle is to access a database using a JDBC connection, as shown in the slide. The client can access a database by using the Connection interface, without knowing a particular vendor's connection implementation details.

Interface Segregation Principle

- Interfaces should be small and specific rather than all encompassing.
- Almost all distributed computing systems can allow multiple interfaces to be supported by network accessible services.
- Two key benefits that you can accrue from applying this principle include:
 - Restriction of the impact of change
 - Avoidance of unnecessary baggage



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

Interfaces should be small and specific rather than all encompassing. It is easiest to apply this principle when program functionality can implement multiple interfaces. This enables you to use many small, focused interfaces in place of larger, less-focused interfaces. Again, almost all distributed computing systems can allow multiple interfaces to be supported by network accessible services.

Imagine a small change made to a large multi-purpose interface. Clearly, any client of the entire interface needs to be recompiled at the very least. However, if a change is made to a small interface, but that interface is implemented by a server alongside many other interfaces, then only clients that use the changed interface need be altered or recompiled.

Also, consider what happens if a system is built in terms of a single, large interface. If it becomes appropriate to reimplement some part of the functionality on a system that is better able to support just that part of the functionality, you have a problem. There are two possible courses of action. Both the original and the new systems must implement the entire interface, or the interface must be split, which requires extensive rewriting of the client programs.

Composite Reuse Principle and Separation of Concerns Principle

- More often than not, inheritance is a poor way to achieve reuse.
- Instead, when you analyze a system, take the following approach:
 - Look for elements that are reasonably stable and parts that are less stable.
 - Look for parts that are closely related and parts that are not closely related.
 - Separate the unrelated parts.
 - Separate the unstable from the stable parts.
 - Use composition to reassemble the elements into the whole.



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

This approach reduces the impact of change and allows much more flexibility in the resulting system. You see examples of this kind of approach in many of the presentation-tier Java EE patterns.

Common Closure Principle and Common Reuse Principle

- The Common Closure principle and the Common Reuse principle relate to packaging, but they are still of value in this presentation. These principles provide the following guidance:
 - Keep closely related elements (elements that change together) together.
 - Keep unrelated elements (elements that change independently) apart.
- Consider using the Common Closure principle and the Common Reuse principle with the Composite Reuse principle.

 ORACLE

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

Java EE technology supports all these principles and uses most of them in its design and implementation.

Applying Patterns

- Solving every problem afresh is not efficient.
- Instead, familiar problems that have been solved successfully in the past should be solved the same way when they recur.
- People commonly use patterns that they are familiar with at the expense of solving the problem correctly.
- Every pattern has a list of associated consequences.

ORACLE

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

Software engineers love to solve problems, it is what makes their world interesting, and it is what gives them satisfaction at the end of the workday. The general idea of reusing proven concepts falls under the heading of heuristics. Heuristics that are sufficiently well known and documented have become patterns. This section describes how to effectively use patterns.

Because patterns contribute to the solution of a particular problem, you need to select and use patterns in the correct way. People commonly use patterns that they are familiar with at the expense of solving the problem correctly. Every pattern has a list of consequences associated with it. You must examine the consequences of using a pattern and decide whether those consequences are a problem. If so, you need to discover how the drawbacks of the pattern affect the subsystem, and then formulate a plan to compensate for those drawbacks if they are significant

- If the drawbacks are minimal, you might be able to ignore them.
- If the drawbacks contribute significantly to risk, you should design a solution to address the risk, or consider removing the pattern altogether.

Common Pattern Catalogs

- Hardware configuration patterns
- Software design patterns:
 - The GOF patterns
 - Java EE patterns
- Architecture patterns
 - Buschmann, et al. patterns
 - Enterprise architecture integration patterns
 - Enterprise application architecture patterns
- Anti-patterns seek to avoid common design traps
 - Examples: analysis paralysis, cash cow, silos, God object



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

The pattern community has cataloged a large set of patterns for use in application architecture. Pattern catalogs exist for the following areas. Note that these pattern catalogs are not mutually exclusive. You can use these patterns together to provide a solution to a larger problem. Most pattern catalogs address software issues, but the concept of patterns is more general than that. Hardware systems is another domain in which patterns can be effective and in which catalogs are beginning to appear.

Common problems and their solutions are well defined in hardware domains, but the ability to compare the solutions in the language of patterns is also of importance. As pattern language moves into this realm, the problems of scalability, reliability, availability, and serviceability can be more easily addressed by looking at the forces and consequences that are associated with each pattern. Solutions, such as load balancing, failover, clustering, and the use of a single server with multiple domains have variations and implementation options that fit well into a pattern catalog.

Gang of Four (GoF) Design Patterns

“A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design. The design pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities. Each design pattern focuses on a particular object-oriented design problem or issue. It describes when it applies, whether it can be applied in view of other design constraints, and the consequences and trade-offs of its use.”

- Gamma, Helm, Johnson, and Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*



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

The book *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides represented the first publication of several design patterns organized together. Often called the Gang of Four or GoF book, this book includes 23 design patterns that provide common object-oriented solutions to common object-oriented design problems. There are many other object-oriented patterns, but the GoF patterns are the most well known. Many J2EE patterns have been built upon the GoF patterns. There are three groupings of design patterns: behavioral, structural, and creational.

Java EE Patterns

- The Java EE patterns bring the features of design patterns to Java EE technology applications.
- The Java EE patterns catalog is grouped into tiers:
 - Integration tier
 - Business tier
 - Web/Presentation tier



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

Java EE-specific design patterns identify the minimal set of known problems that an application architecture should solve. These patterns are based on experiences of the Java EE community and use Java EE technologies to solve problems.

These patterns focus on two primary efforts in application design:

- The patterns promote flexibility.
- The patterns improve performance.

These two characteristics are present throughout the Java EE pattern catalog. You can help to ensure the proper use of patterns by evaluating their flexibility and performance capabilities.

The advantages of applying a design pattern are: proven solutions to the problems, reusability, improved performance, increased maintainability, and so on. The entire Java EE patterns catalog is available at <http://java.sun.com/blueprints/corej2eepatterns/index.html>.

Most GoF design patterns are abstract solutions that are not intended strictly as recipes. Instead they suggest solutions that should be varied as the needs of the implementation require. GoF patterns can be used in any object-oriented programming language.

In contrast, J2EE patterns are intended to be used in the Java EE platform, although many of them can be adapted for use outside the Java EE platform and the Java programming environment. These pattern's relation to Java EE technology leads many people to conclude that Java EE patterns are really idioms. Idioms are patterns that rely on specific features of a programming language or platform.

Unlike the Gang of Four book, which divides its catalog into creational, structural, and behavioral design patterns, the Java EE patterns catalog is grouped into tiers. These tiers are similar to tiers in the Java EE architecture. However, keep in mind that this partitioning is logical rather than physical. Each pattern will be covered later in the course in its corresponding lesson.

The J2EE Patterns Catalog

Integration Tier:

Service Activator Allows a client to asynchronously invoke an EJB component, using the JMS API

Data Access Object Abstract and encapsulate data access mechanisms

Domain Store Creates a robust persistence mechanism that is transparent to the business objects without using entity beans

Web Service Broker Makes business services available as web services

Business Tier:

Service Locator Simplify client access to enterprise business services

Session Façade Coordinate operations between multiple business objects in a workflow

Transfer Object Transfer business data between tiers

Application Service Centralizes business logic between the service façades and the business objects

Business Delegate Reduce coupling between Web and Enterprise JavaBeans tiers

Transfer Object Reduces the number of remote method calls by encapsulating multiple business data values into one serializable component based on JavaBeans component architecture (JavaBeans component) that can be returned to the client

Transfer Object

Assembler Assembles transfer object data from multiple business objects

Composite Entity Model a network of related business entities

Value List Handler Provides a mechanism for executing queries that might return a large number of objects, and provides browsing through the results in an efficient manner

Presentation Tier:

Intercepting Filter Preprocess and postprocess application requests

Front Controller Centralize application request processing

Application Controller Separates the action invocation management and view dispatching management from the front controller component

| | |
|-------------------|---|
| Context Object | Passes data from context-specific objects without passing those objects out of their context |
| View Helper | Simplify access to model state and data access logic |
| Composite View | Separately manage layout and content of multiple composed views |
| Dispatcher View | Combines the Front Controller and View Helper patterns. View helpers can use business delegates to communicate with the business tier |
| Service to Worker | Similar to the Dispatcher View pattern, except that the front controller takes more responsibility for view selection and business process invocation |

Architecture Patterns by Buschmann, et al.

- The collection of architecture patterns in the book *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns* by Buschmann, et al. provides a foundation for building systems.
- For example, the Model-View-Controller (MVC) pattern has been shown to be useful in creating architectures that separate presentation logic, user view, and application model.
- The use of the Layers pattern helps to ensure flexibility and portability.



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

Buschmann, et al. also describe various communication patterns. These patterns promote flexibility and portability by reducing coupling between software components and the communication mechanisms they use. Location transparency is another benefit of communication patterns. When the communication dependency is removed, the software components no longer depend on their deployment environment. You can then reconfigure the underlying infrastructure to meet architectural requirements without a need to rewrite component communication logic. For example, The Forwarder-Receiver and the Client-Dispatcher-Server patterns promote flexibility, decoupling, and location transparency.

The Publisher-Subscriber pattern covers asynchronous communication. This pattern is a variation on the Observer pattern, but adds the additional element of the Event Channel to further decouple the publishers and subscribers in a distributed environment. The Publisher-Subscriber pattern is often used in conjunction with a Broker pattern, such as in the implementation of CORBA systems.

The Layers Pattern

- Helps establish logical tiers and technology layers
- Defines and enforces abstraction boundaries
- Examples:
 - The tiered architecture of Java EE applications
 - An abstract data access layer in the form of object wrappers



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

The Layers pattern helps establish logical tiers and technology layers in a distributed system and ensures that abstraction boundaries are defined and enforced. However, you can use the Layers pattern at other levels in the design of architectures to promote flexibility.

As an example, consider a system that must integrate with a moderately volatile database. This scenario could arise when a company is in a period of acquisition and merger, and an application must maintain a level of integrity in the face of changing data representations. If the architect uses a database abstraction layer in the form of object wrappers, the application is insulated from database changes.

Enterprise Integration Patterns

- 65 patterns “harvested” by Gregor Hohpe
- Focus on enterprise integration using messaging:
 - Message-oriented middleware
 - JMS
 - Microsoft's Message Queuing
 - WS-ReliableMessaging



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

In his book, *Enterprise Integration Patterns*, Addison-Wesley, 2003, Gregor Hohpe, software engineer and architect at Google, presents 65 patterns that make the design and implementation of integration solutions easier. The book provides a consistent vocabulary and visual notation to describe large-scale integration solutions across many implementation technologies. It also explores in detail the advantages and limitations of asynchronous messaging architectures.

Enterprise Application Architecture Patterns

- Written by Martin Fowler and contributors
- Details over 40 architectural patterns
- Topics include:
 - Layering of enterprise applications
 - Structuring domain logic
 - Structuring a web user interface
 - Linking in-memory modules to a RDBMS
 - Handling session state in stateless environments
 - Principles of distribution



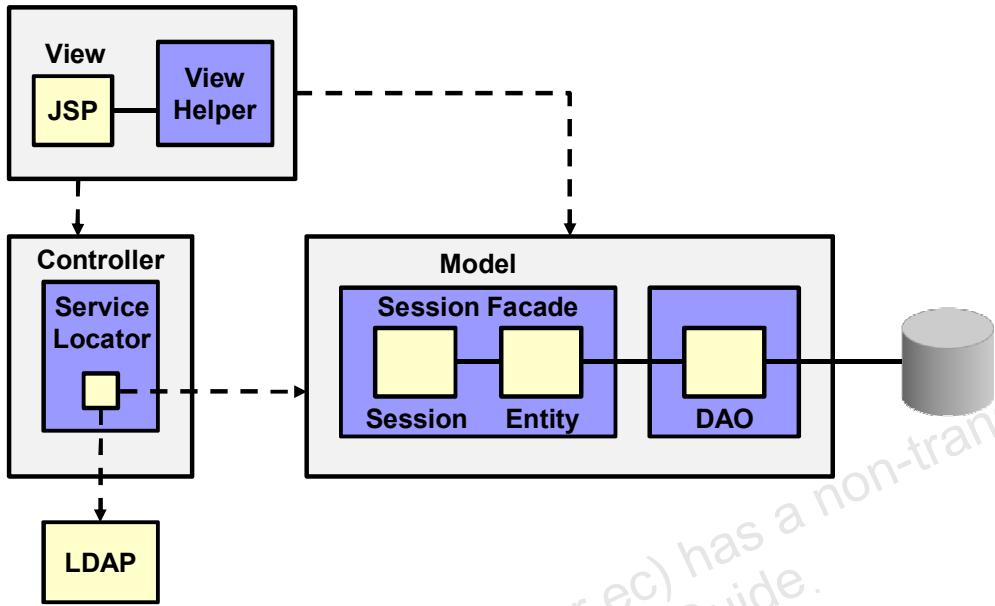
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Martin Fowler noticed that despite changes in technology, the same basic ideas can be adapted and applied to solve common architecture problems. Over 40 different patterns are presented. The book is organized in two parts. The first section is a short tutorial on developing enterprise applications. The second section is a detailed reference to the patterns themselves.

Patterns covered include:

- Domain Logic Patterns
- Data Source Architectural Patterns
- Object-Relational Behavioral Patterns
- Object-Relational Structural Patterns
- Object-Relational Metadata Mapping Patterns
- Web Presentation Patterns
- Distribution Patterns
- Offline Concurrency Patterns
- Session State Patterns
- Base Patterns

Pattern Integration



ORACLE

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

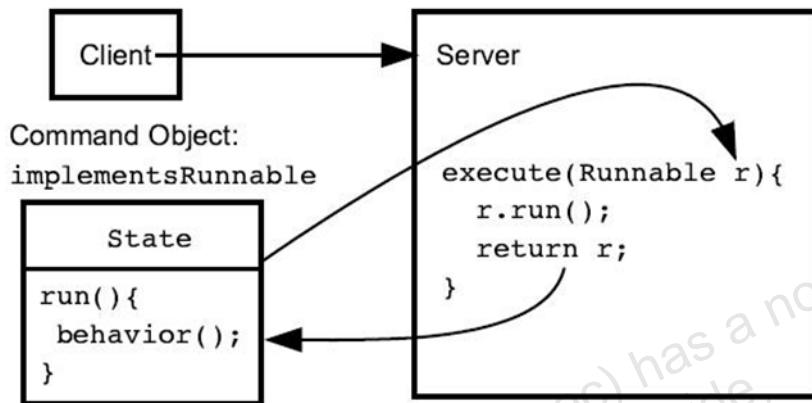
The architect uses a top-down approach to defining systems and their interactions. This top-down approach allows you to use patterns to solve high-level problems, such as distribution, application partitioning, and service levels. After you have solved the high-level problems, you can address the problems of individual subsystems. Use patterns to solve the smaller problems without compromising the larger system architecture.

For example, you can use the MVC pattern to establish the top-level specifications of an application server. Then, as shown in the slide, you can implement the individual components inside the application according to patterns, such as Session Façade, Service Locator, or so on.

Using Patterns Across a Network

Example: the Command pattern

- Does this work as well in an object-based system?



ORACLE

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

Consider the Command pattern as an example. It has the following characteristics:

A is encapsulated along with its arguments in an object.

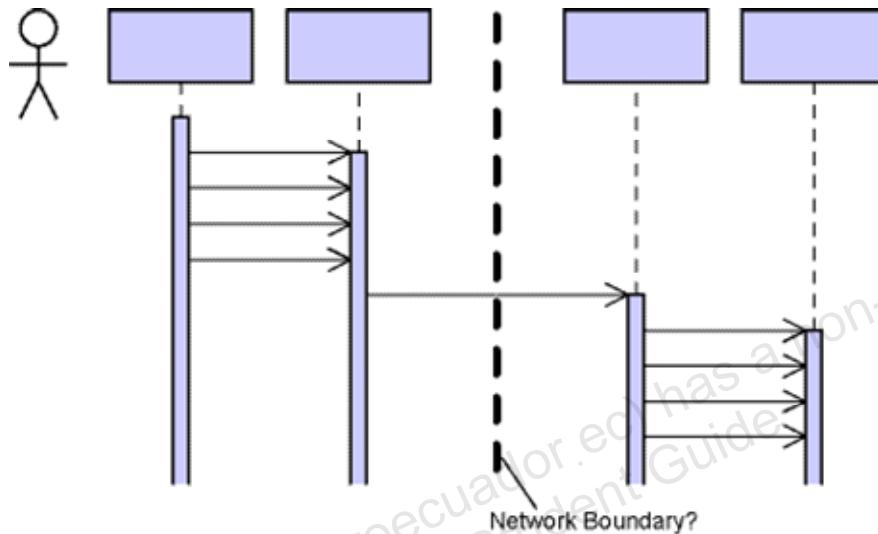
The object has a well-known interface—for example, the `java.lang.Runnable` interface—so that it can be executed easily. The command and arguments, encapsulated as an object, are passed from the client to the server. Because the object carries state behavior, when the server executes the Command object, it is actually executing behavior it might never have seen before. The Command pattern is powerful because it enables a client to extend server functionality without the drawbacks of the following three approaches:

- Reprogramming the server to provide the new functionality directly
- Building the server with fine-grained behavior, then making multiple calls from the client to aggregate these behaviors into the required functionality
- Having coarse-grained behavior that passes unnecessarily large amounts of data to the client and allowing the client to filter what is actually needed

For a Command pattern to work over a network, the distributed computing infrastructure must be able to transmit both state and behavior over the network (see the slide). Most systems do not do this because they are object-based at best and cannot support software that uses the Command pattern. In contrast, RMI, with its native network protocol JRMP and with an environment that supports dynamic class downloading, is a fully OO distributed system that can implement the pattern directly.

Object-Based Patterns

You can distribute these object-based patterns over networks, even if the underlying network system is not object-oriented.



ORACLE

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

In many cases, OO design patterns do not require the transfer of behavior with arguments and return values. These patterns might be described as object-based, rather than object-oriented. The use of object-based patterns gives the same benefits in a distributed environment as in a local system. However, you must take care to avoid potentially substantial performance penalties. The approach that minimizes problems with performance penalties includes the following tasks:

- Examine the frequency of calls between each component of the pattern.
- Examine the data in the arguments and return values of those calls.
- Place network boundaries only at low-frequency, low data-volume interfaces, as shown in the slide.

Using Reliable Frameworks

- A framework is partially complete system that can be extended and instantiated.
- Frameworks are essential in creating reusable software and systems.
- Frameworks are typically pattern based.
- Example: The EJB technology and other application frameworks.



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

According to Buschmann et al., a framework is, “A partially complete software (sub-) system that is intended to be instantiated. It defines the architecture for a family of (sub-) systems and provides the basic building blocks to create them. It also defines the places where adaptations for specific functionality should be made.”

This idea is important when you create reusable software and systems. You can build frameworks by assembling patterns. Frameworks then become patterns for assembling systems. When you extend or instantiate the elements of a framework, you can apply the framework to a particular problem domain.

One relevant example of a framework is the EJB technology. The EJB architecture framework is a collection of patterns that are applied to the domain of enterprise applications.

Open Source Issues

- Being used more and more.
- Half the battle is mind-set.
- Beware of patent trolls suing the OSS vendor.
- Who will support it? Is there an expectation from the business that the OSS support will always be there?
- How long lasting will the OSS be? Will it be around in the future?
- Are there software packaging restrictions?



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

Consider using the Open Source Initiative list of licenses as a source.

Additional considerations with OSS:

- Produced by individuals with no intent to “monetize” their product.
- Large companies can be skeptical about using open source.
- May be no solid support, documentation, community, or education structure.
- Is there tooling available?

Suggestions:

- Develop a policy for testing and implementing open source software.
- Decide if you will contribute back to the OSS community.
- Define a policy for OSS procurement. This should be no different than procuring other software.
- Manage OSS as a company asset.

Note: Oracle's OSS offerings typically are not subject to similar concerns due to Oracle's support, industry presence, and commitment.

Service-Based Architecture

- The proper implementation of services is the key to producing architectures with good security, availability, and scalability.
- A service provides access to application components as a small, coherent set of APIs.
- Clients are strongly decoupled from the service implementation.
- Services typically encapsulate the business domain.
- Services are sharable.
- A service-based architecture can be more reusable and extensible.

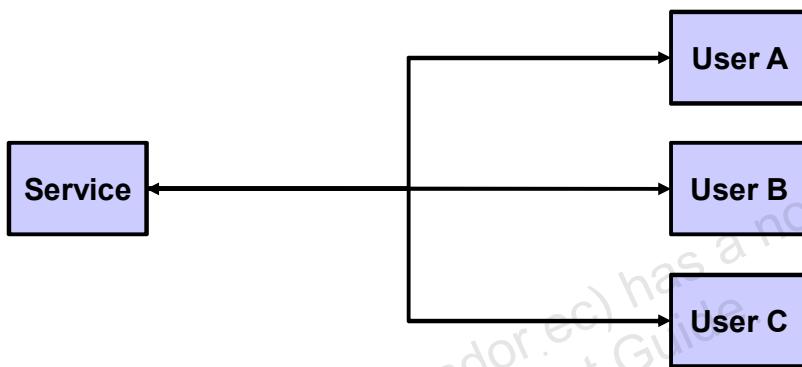


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Oracle's senior architects suggest that scalable, available architectures are typically service-based, with collections of cooperating and communicating services. To fully define the architecture, the architect should consider the infrastructure, as well as the application architecture. For high-capacity systems, the infrastructure works closely with the application.

Developing Service-Based Architectures

- A service allows many concurrent users to access resources that it manages.
- Services map to enduring business themes, or map top-level domain objects.



ORACLE

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

Service-based architectures are composed of a number of high-level services. Each service provides access to application components as a small, coherent set of APIs. The resultant architecture captures the enduring themes that correspond to the business.

A service is the software analog to a server in a client-server architecture. A server is a single point of control that manages a limited resource for use by multiple clients. A service allows many concurrent users to access resources that it manages, while it simultaneously maintains the integrity of the data.

Clients are strongly decoupled from the implementations. This makes modifications and client updates easier. Services act as outlets in the sense that they provide a means of connecting to the rest of the application. This feature makes clients insensitive to the implementation of the service, which improves extensibility. The services can be modified without the client's knowledge.

Services encapsulate the business domain. Business components can ask for services rather than interact with them directly. This loose coupling means that service-based architectures are less brittle. They respond easily to changes in business demands, and you can tune each system for the capability it provides. You can easily move around well designed services because they are location transparent.

Many components and applications can share the same services, so the reuse of complicated services makes the architecture more extensible and easier to manage.

You can partition by tiers or partition by special functions as alternatives to service-based architectures. Although these are not good approaches for partitioning the entire architecture, they are still useful for some parts of the application. For example, you can partition so that one machine has a Kerberos authentication that is physically protected.

If you partition by tier or by special function, and you distribute across many machines, then you are likely to have one big monolithic application. If you want to split functionality or modify the distribution of the parts, you need to rewrite this application.

Types of Services

- Vertical services:
 - Based on the content of the system
 - Reflect the business model and are specific to a particular domain
 - Example: an inventory service
- Horizontal services:
 - Based on the infrastructure of the system
 - Are accessed at many layers and across different domains
 - Example: an authentication security service

ORACLE

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

The architect considers two types of services that are orthogonal to each other:

Vertical services: Based on the content of the system, these services reflect the business model and are specific to a particular domain. For example, the inventory service can present an object that represents an item in a warehouse inventory.

Horizontal services: Based on the infrastructure of the system, these services are accessed at many layers and across different domains. Horizontal services are items, such as security, which are carried throughout the application and touch the application at many levels. An authentication security service is one example, where the client requests some form of proposed authentication information and hands back a credential that states the permissions for the client.

A similar term frequently mentioned is Service-Oriented Architecture (SOA). SOA has a strong focus on using service-based components to encapsulate high-level business operations. By designing the system in this fashion, it improves the ease of integrating heterogeneous systems. SOA is not a brand new concept, because it is possible to implement an SOA system using many existing technologies. However, as web services become more commonly used to implement distributed systems, the implementation of an SOA-based system typically relies heavily on web services. In this regard, SOA can be considered as a style of the Service-Based architecture with more constraints. For more information on SOA, refer to the lesson titled “Developing an Architecture for the Integration and Resource Tiers.”

Granularity of Services

- Striving for an appropriate level of granularity will maximize ease of use, reuse, and manageability.
- Ideally, your SOA portfolio should consist primarily of coarse-grained service interfaces, which map directly to business semantics.
- Low-level services, in contrast, are tightly coupled to underlying infrastructure or APIs and cannot easily be modified to suit changing business requirements.
- Ultimately, it is important to remember that an appropriate service is not necessarily either fine or coarse, but one that maximizes business value.



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

A top-level, business-driven analysis will typically uncover high-level (coarse-grained) services that map to existing needs. As an example, a coarse service might represent “Process Purchase Order,” which clearly maps to a business process.

Because it is focused on IT infrastructure and existing APIs, the bottom-up analysis will typically yield a significant set of coarse-grained services as well as a number of low-level (fine-grained) services. An example might be a function enabling a lower-level task such as “Insert Order Line Item.” Coarse interfaces also allow you to build applications and processes using other services from heterogeneous environments without concern for the details and differences. High-level services are much more agile in a dynamic business environment, as they are not tightly coupled to underlying infrastructure. Low-level services, in contrast, are tightly coupled to underlying infrastructure or APIs and cannot easily be modified to suit changing business requirements. In fact, wrapping a service around an existing business object, such as an Enterprise JavaBean (EJB), will inevitably yield a fine-grained interface exposing each method that can be called on the bean.

Services that will be used to address very specific use cases within the organization may be implemented with fine-grained interfaces, which provide more flexibility to client applications that use these services. Bear in mind, however, that this increased flexibility brings with it the cost of increased complexity.

In general, you should avoid exposing low-level interfaces as part of a service portfolio intended to meet business needs. Consider instead combining a set of fine-grained services and composing them as coarse-grained services.

Versioning Services

A change in a Web services implementation may affect its consumers depending on a number of factors:

- A change in the operation parameters of a web service.
- A change of the name of an operation (this will affect the current consumers).
- The addition of an operation (this might affect the current consumers).
- A deletion of an operation (this will affect the current consumers).
- Using a Service bus can address versioning of services.
- Consider the service's lifecycle: what happens over time? At the end?



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

Web services are bound to change and evolve over time. The loose coupling principles of service-oriented architecture (SOA) imply that service providers can release a new version of a shared service without waiting for consumers to adapt, and that service consumers should test and certify on a new shared service version before switching. Consequently, you might need to have multiple versions of a shared service running concurrently and simultaneously accessible by different service consumers. Some service consumers might need to continue using an old version of a service until migration of the consumer code occurs. Therefore, web services versioning is an important subject that should be considered carefully in all enterprise SOA approaches.

A typology of change in web services can be created in relation to the impact on the current consumers of those services. One approach is to qualify a change that will not affect the current consumers as a minor release and a change that will affect the current consumers as a major release.

Minor Release

A minor release can be one of two types. The first is a correction of a bug or a performance enhancement. This type will not affect the Web Services Description Language (WSDL) of the web service. The second type consists of adding new methods to a Web service, wherein the WSDL is changed with no impact on service consumers. A distinction can be made between these two types when labeling those versions. For example, for the first type you can change the second decimal place of the version number (1.0X), while for the second type you change the first decimal place of the version number (1.Y0).

Major Release

A major release involves a change that will break backwards compatibility. In this case the consumers must be modified. A release that only affects the functionalities of a Web service, without affecting the WSDL, is also considered a major release. This is because the current consumers cannot invoke the new version without considering the web service's modified functionalities. Now that we have identified the various types of changes and their impact on current consumers, let us take a look at different patterns for web services versioning.

What Do You Think?

What is the difference between an object, a component, and a service?

- As a designer?
- As an architect?



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:

- Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Readings: Addison Wesley Publishing Co., 1995.
- Cockcroft, Adrian and Bill Walker. *Capacity Planning for Internet Services*. Sun Microsystems, 2000.
- Fowler, Martin. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.
- Hohpe, Gregor. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2003.



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

Lofstrand, Mikael and Jason Carolan, *Sun's Pattern-Based Design Framework: The Service Delivery Network*. Sun Microsystems, 2005.

Kakadia, Deepak, Sam Halabi and Bill Cormier. *Enterprise Network Design Patterns: High Availability*. Sun Microsystems, 2003.

J2EE Patterns Catalog: <http://www.oracle.com/technetwork/java/catalog-137601.html>

Quiz

The primary focus of architecture is controlling risks. Which of the following typically presents the greatest overall performance impact on a distributed system?

- a. Transaction model
- b. Network communications and layout
- c. Security model
- d. System size



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

Answer: b

Quiz

Designing for flexibility enables a system to change over time without significant rework. What are some best practices for increasing flexibility? (Choose all that apply.)

- a. Selection of a good database table model
- b. Loosely-coupled component models
- c. Implementing a Model-View-Controller pattern
- d. Limiting the number of transactions



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

Answer: b, c

Quiz

Given a situation where you are asked to design components that will be extended or modified with the guarantee that the new components will not break other implementations, which principle should you apply?

- a. Dependency Inversion
- b. Interface Segregation
- c. Open-Closed
- d. Separation of Concerns



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

Answer: c

Quiz

Presented with the problem of working with an existing database that has been highly customized with triggers and stored procedures, what pattern could you recommend to reduce the impact that changes to the database have on the enterprise application you are charged with designing?

- a. Layers Pattern
- b. Model-View-Controller
- c. Business Delegate
- d. Session Façade



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

Answer: a

Summary

In this lesson, you should have learned how to:

- Identify key risk factors in distributed enterprise systems
- Design a flexible object model

Practice 5 Overview: Create a Flexible Object Model

This practice covers the following topics:

- Use patterns to make the Domain model more flexible.
- Identify and model services to support a more flexible architecture.



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

In this practice you will further iterate your architecture by applying patterns to increase the flexibility of the Domain model. You will also identify potential services, both coarse grained and fine grained.

Defining Common Problems and Solutions



Part 2: Network, Transaction, and Capacity Planning

ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Define the guidelines of creating a network model
- Justify the use of transactions
- Plan system capacity

Discussion Questions

- What aspects of network communication cause the biggest risk? How can you control network communication risks?
- What are transactions? Why are transactions needed? How do transactions degrade system performance? What steps can you take to limit the risks?
- How can you estimate the transaction load and rate values, which are significant factors that affect overall system responsiveness?

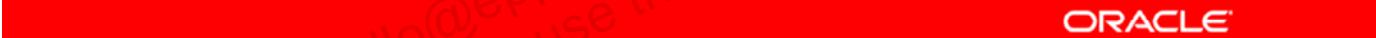


ORACLE

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

Lesson Agenda

- Describing network communication guidelines
- Justifying the use of transactions
- Planning system capacity



ORACLE

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

Network Communication Guidelines

- Network performance is a primary architectural concern in network communication.
- Three factors that determine network performance:
 - Bandwidth
 - Request latency
 - Request frequency



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

This course uses the following definitions for the terms bandwidth, request latency, and request frequency:

- **Bandwidth:** Governs the data capacity of a network connection. Only a limited amount of bandwidth is available to be shared among many users. Therefore, the effective use of bandwidth is a primary goal in network communication.
- **Request latency:** Represents the time elapsed in completing a network communication. Actual request latency (or round trip time) has a significant impact on network performance. Request latency is made up of software delays and time spent in the network itself. A delay is caused by time spent in network stacks and time taken by the system to compute a response.
- **Request frequency:** Reflects the use of the channel capacity (bandwidth) and is sometimes called chattiness.

Network Performance Guidelines

- To use bandwidth effectively, you send data in large chunks that require fewer round trips.
- This can be hard to follow for a couple of reasons:
 - All the data may not be available at once
 - It takes significant time to calculate all the data
- To minimize request frequency:
 - Careful UI design
 - Careful remote API design



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

If you do not have all the data you need available at once, it can make it mandatory that you use multiple round trips. This is particularly common with a user interface because the user might not know the possible answers to the system's next question until the response screen is displayed from the previous question.

It takes significant time to calculate all the data. In addition, the process at the other end is time-consuming and can start with only part of the data. In this case, it might still make sense to send some of the data in multiple chunks. A situation in which you might send data in multiple chunks is rare, but an example is when you need the system to handle the results of a large search, particularly if a user is browsing through the results.

You can usually minimize request frequency or chattiness with one of two approaches:

- **Careful UI design:** Use as few screens as possible, and provide as much information as possible with each screen.
- **Careful remote API design:** Ensure that the whole of a request is expressed.

Distributed Computing Fallacies

Some of the false assumptions (fallacies) for the network can have a significant impact on the creation of the architecture:

1. The network is reliable.
2. The latency is zero.
3. The bandwidth is infinite.
4. The network is secure.
5. Topology does not change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous. (Gosling)



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

The list of fallacies is considered to have originated at Sun Microsystems. Peter Deutsch, one of the original Sun “Fellows,” is credited with writing the first seven fallacies in 1994. However, Bill Joy and Tom Lyon had already identified the first four as “The Fallacies of Networked Computing.” Around 1997, James Gosling, added the eighth fallacy.

What Do You Think?

“I think a critical cloud computing issue clearly suggests what the ninth one should be: 9. *Location is irrelevant.*”

- Harry J. Foxwell, PhD.



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

Creating a Network Model

- The essential difference between local and distributed computing is the network.
- The network makes calls slower, transactions longer, and security harder to achieve.
- Java EE technology can do little to influence either of these aspects.



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

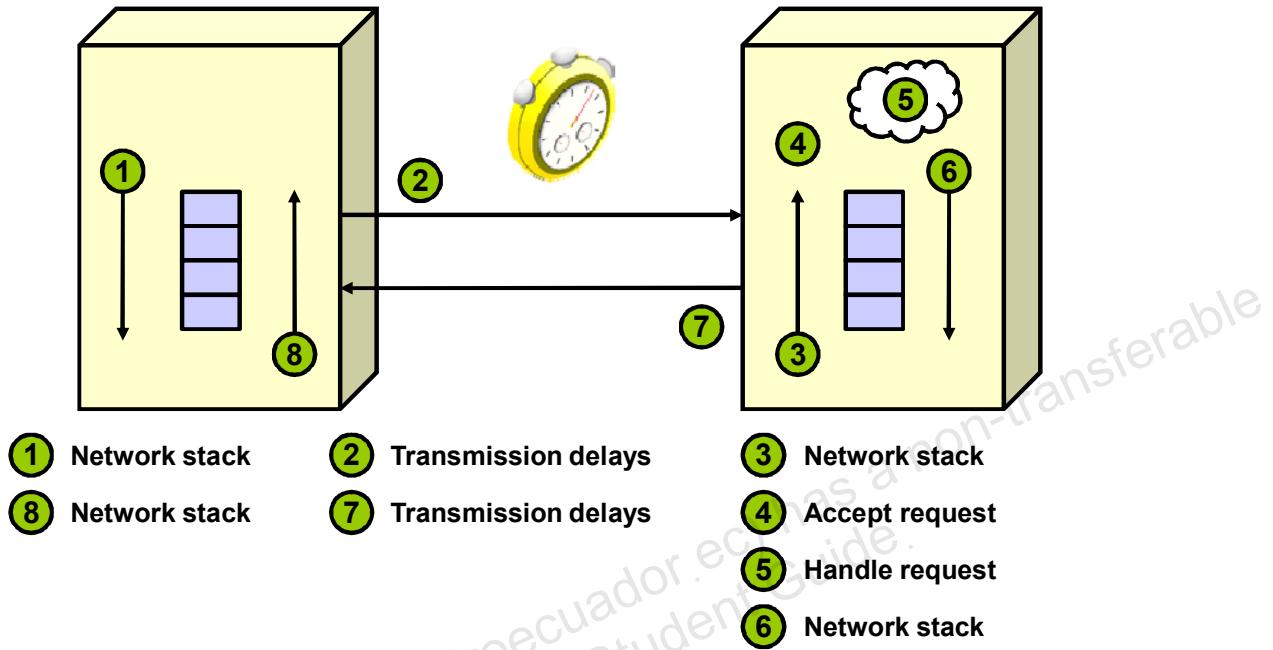
When you design a network model, carefully examine the factors that can influence the speed of a call, the length of a transaction, and the implementation of security. Some decisions are typically out of the control of the architect, such as the network bandwidth that is in use. However, other decisions are definitely within the architect's purview.

You must consider two critical aspects when you design network communications:

- Network latency or round-trip time
- Number or frequency of network calls

Java EE technology can do little to influence either of these aspects. However, Java EE technology can be used well or poorly, depending on the thought given to the individual situation by the architect. In other words, Java EE technology provides all the necessary tools to create good network solutions, but cannot enforce them in a given situation because the best approach depends on the particular application being built.

Estimating Network Latency



ORACLE

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

The accumulation of request latency can dramatically degrade network performance, as illustrated in the slide.

The time required to complete a request and response cycle depends on several factors at each stage in the cycle:

Sending the message:

- Time for the sending system to prepare the message
- Bandwidth or bandwidths of the network or networks used to send the message
- Size of the packet sent
- Signal path of the route taken
- Time lost in routers and repeaters and similar devices

Servicing the message:

- Time for the receiving system to start to service the message
- Time to service the message

Responding to the message:

- Time to assemble the response
- Size of the response packet that is returned

Servicing the response:

- Time for the receiving system to start to service the response
- Time for the receiving system to service the response

Of the many factors involved, the architect has the most control over the size of the packet and the responsiveness of the receiving machine.

- The responsiveness of the receiving machine is composed of:
- The time for the receiving system to start to service the message
- The time to service the message
- The time to assemble the response
- The responsiveness of the receiving machine mainly depends on:
- The power of the system
- The load on the system
- The response times of any systems that the receiving machine must delegate work to, such as a database server

Estimating Network Service Request Frequency

Estimate the network request frequency at both the client and the server, or on each side of any other type of link. This is because a typical web client sees only network transactions that relate to its own needs. Moreover, a web client runs over a slower link than the server, which sees all of the network transactions from all of the clients.

Obtain the following information to estimate the network service request frequency:

- What triggers a network transaction? In most cases, the trigger is a user.
- How much time does the user spend on each page, and how many network transactions does each page trigger? In most cases, one page triggers one transaction. This gives a frequency for the client.
- What is the frequency figure for the server? To calculate this frequency, multiply the estimate of the frequency for the client by the anticipated number of concurrent clients.

Constructing Efficient Data Models

- The data model that you implement to transport arguments and return values is critical to the efficient use of the network and thus, to the response times of a system.
- The following issues address how you create an efficient data model:
 - Aggregate the data.
 - Ensure services are coarse-grained.
- Additionally, you can use compression to further reduce the network usage.



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

The following issues address how you create an efficient data model:

- Aggregate the data so that a single round trip carries all of the data needed to answer a question, and the response carries all of the data needed to describe the response.
- Ensure that methods that are provided by a service are coarse-grained enough so that a single method can answer most, or all, questions.

Aggregation of Larger Data Sets From Smaller Sets

A critical element in an efficient data model is that no more round trips are made than are absolutely necessary.

Examine the triggers that cause network traffic.

These triggers are usually user actions. Consider how, within the boundaries of good user interface design, you can aggregate screens to reduce the number of separate network transactions, while still keeping all of the same data on each screen.

Increased Operation Granularity

- To minimize the number of round trips, map each question that the client might need to ask to a single method in the server.
- You can best implement this approach with either the Session Facade or Business Delegate patterns.



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

In other words, the server provides high-level answers, rather than requiring the client to collect lots of low-level answers and then aggregate the results. Use the Business Delegate pattern to enable the client to make fewer requests over the network, even when the client requires lots of little data elements. Use the Session Facade pattern to provide higher-level functionality at the server side. In some systems, you might use both patterns in the same system.

Compression

- Data might be compressed as a result of externalization or as a result of using a compressed socket.
- If the data are compressed in either of these ways, the round-trip count is unaffected, but the size of the data sent over the network connection itself is reduced.

ORACLE

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

This approach is valuable in the following situations:

- The size of the data are relatively large compared to the bandwidth of the network so that it takes a significant amount of time just to physically transmit the data.
- The compression allows the whole message to be sent in a single frame at the physical layer, rather than having to split the message across multiple packets.

Lesson Agenda

- Describing network communication guidelines
- Justifying the use of transactions
- Planning system capacity



ORACLE

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

Justifying the Use of Transactions

- A transaction is a single unit of work that is performed by a series of individual operations, where the whole series must all succeed or all fail.
- Transactions have a close relationship with two important features in a distributed system:
 - Correctness
 - Performance
- Transactions can have a significant impact on performance in a distributed system.

 ORACLE

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

Transactions are often crucial to correctness because they enforce the idea that some operations take place as an indivisible group. Either all the operations succeed, or the system is left as if none of the operations had ever taken place.

However, Martin Fowler in his post: <http://www.martinfowler.com/bliki/Transactionless.html>, describes how eBay is able to work without transactions, in many cases.

Gregor Hohpe talks about not using two-phase commit in his post:
http://www.eaipatterns.com/ramblings/53_eric.html

When Are Transactions Required

To determine if an operation is transactional, you must examine several aspects of the data storage and retrieval process for your application. Your primary concerns are as follows:

- Concurrent access
- Read-only access as opposed to read-write access
- Data reliability



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

To help identify transactional operations, start by identifying the logical transaction model for the application and its resources.

Identify whether transactions are required.

Determine the level of complexity of the transaction model.

- Is the transaction model simple enough to be managed through resource manager local transactions?
- Is the transaction model complex enough to require distributed transaction protocols, such as two-phase commit?

Not all transaction management falls into one or the other of these two boundary cases. You might need to incorporate a combination of direct access to a resource manager, interaction with a transaction manager, and use of techniques like messaging to achieve the desired result. Many times, these creative solutions require complex application code.

You also need to ensure that transactions within the model conform to the following four characteristics of transactions, known as the ACID test:

Atomicity: Operations are all or nothing.

Consistency: Resource state changes are complete.

Isolation: Concurrent transactions result in sequential resource state changes.

Durability: Resource state changes, once committed, are persistent.

CAP Conjecture

- Is it possible to achieve consistency, availability, and partition tolerance?
- Classic distributed systems focused on ACID
- Modern internet systems focus on BASE: Basically Available Soft-state (or scalable) Eventually consistent
- In a shared-data system you want: strong Consistency, high Availability and Partition tolerance (CAP).
- CAP Conjecture (Brewer):
 - You can only have two out of these three properties
 - The choice of which feature to discard determines the nature of your system
 - Which do you choose, and why?

ORACLE

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

CAP Theorem

Eric Brewer, professor at the University of California, Berkeley, and cofounder and chief scientist at Inktomi, made the conjecture that distributed systems cannot ensure all three of the following properties at once (signified by the acronym CAP):

- **Consistency:** The client perceives that a set of operations has occurred all at once.
- **Availability:** Every operation must terminate in an intended response.
- **Partition tolerance:** Operations will complete, even if individual components are unavailable. Specifically, a web application can support, at most, only two of these properties with any database design. Obviously, any horizontal scaling strategy is based on data partitioning; therefore, designers are forced to decide between consistency and availability.

ACID versus BASE

ACID

- Strong consistency for transactions highest priority
- Availability less important
- Pessimistic
- Rigorous analysis
- Complex mechanisms

BASE

- Availability and scaling highest priorities
- Weak consistency
- Optimistic
- Best effort
- Simple and fast

ORACLE

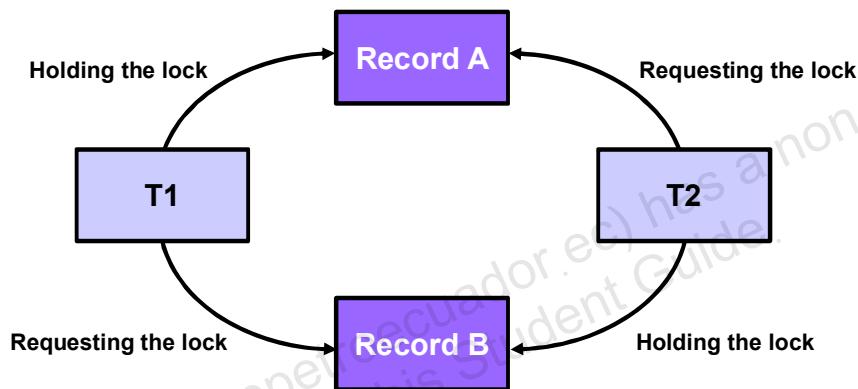
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

BASE (Basically Available, Soft state, Eventually consistent) 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.

Difficulties of Transaction Models

- The effect of a transaction is primarily to serialize access to a resource that is often one or more rows in a database.
- However, this is not a trivial operation for two key reasons:
 - Transactions can have a substantial impact on performance and throughput.
 - The deadlock situation:



ORACLE

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

In terms of difficulty, transaction design is like multi-threaded programming, which is a notoriously tricky programming issue. The effect of a transaction is primarily to serialize access to a resource that is often one or more rows in a database. However, this is not a trivial operation for two key reasons.

- Transactions can have a substantial impact on performance and throughput, sometimes for the better, but most often for the worse. Problems often arise as a result of a process being held up unnecessarily.
- A deadlock occurs if two exclusive access transactions are competing for locks that are being held by the other, as shown in the slide. When a deadlock occurs, neither transaction can proceed. Unfortunately, there are no standard mechanisms to prevent or detect deadlocks.

Transaction design, like multithreaded programming, is a potentially nondeterministic and, therefore, a difficult problem.

Types of Transactions

- Local Transactions:
 - Transactions require only one data resource
 - The resource can be accessed through several different connections
- Distributed Transactions:
 - Transactions require cooperative operation of multiple data resources
 - Transactional operations are executed in a transaction context
 - The transaction context is propagated to all participating resources. Multiple protocols exist for propagating the context.



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

Some transactions require only one data resource. You can access this resource through several different connections, but it remains only one resource. This type of transactional access is termed a local transaction. On the other hand, if a transaction requires co-operative operation of multiple data resources, it is a distributed transaction, which leads to a more complex management model.

To control and manage distributed transactions, you must typically execute operations in a transaction context. Moreover, the transaction management system must ensure the propagation of the context to all resources participating in the transaction. There are many protocols for propagating the context of a distributed transaction. Among them and the most widely used, is the X/Open XA model from the Open Group (see <http://www.opengroup.org>). This model includes the DTP and 2PC protocols.

Java EE technology provides substantial transaction management assistance. The Java Persistence API (JPA) and EJB technology containers can use transactional resources, such as databases, by using the XA distributed transaction protocols.

Impact of Transactions on Latency and Request Frequency?

- The propagation of transaction control commands through the network is resource-intensive and time consuming.
- Factors that contribute to the negative impact of transactions:
 - Transaction control
 - Transaction scoping
 - Write-Write conflict
 - Isolation levels



ORACLE

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

When system components dedicate network bandwidth and computing power to managing transactions, they take resources away from processing additional user requests. This causes an increase in latency and a decrease in application request frequency. As a result, you must take away from scalability and performance to ensure reliability. Of course, the degree of impact might be slight if the available bandwidth is great compared to the bandwidth required for the transaction management.

Transaction Control

To minimize the effects of transactions on network bandwidth and system capacity, try to keep the transaction control as close to the resource as possible. This does not mean that you should use only resource managers, because that would force the transaction management into the application components. Instead, minimize the use of context propagation and try to keep the components in close proximity, such as on the same network segment when distribution is required.

Transaction Scoping

Another issue that affects network bandwidth and system capacity is the transaction scoping problem. Address this problem by using long transactions (synchronous or blocking) and short transactions (asynchronous or non-blocking). A long transaction is a transaction that uses a single database connection and transaction context to complete a use case. Short transaction result in a sequence of connections and contexts to perform a single use case.

To properly assess the choice of long and short transactions, you must consider the trade-offs they imply. Long transactions are reliable, but negatively impact scalability. Short transactions are more scalable, but require a more complex coding style, which leads to increased manageability costs.

Note: Java EE technology allows developers to choose between carefully optimized transaction strategies that might be inflexible but fast and more generalized strategies that are highly flexible, but might not perform as well. In addition, the Java EE technology transaction mechanisms allow a degree of control over an application's transactional behavior at deployment time.

Write-Write Conflict

- Another problem that the use of short transactions creates is the write-write conflict problem.
- Also known as overwriting uncommitted data
- The write-write conflict arises when two clients attempt to change the same data at the same time and both have updated the data outside of a transaction.
- You now require extra application code to identify and resolve the write-write conflict.



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

To solve a write-write conflict, you can adopt one of two locking strategies to isolate the data:

Pessimistic locking: Creates an exclusive lock on the data during modifications. In other words, only one client at a time is allowed to check out the data for modification. This solution provides higher reliability at the expense of scalability.

Optimistic locking: Checks to see if multiple clients who are attempting to modify data are making incompatible changes. If they are not, the modifications can be executed normally. On the other hand, if incompatible changes occur, you must detect them and generate a response that allows the client to take appropriate action. However, the ability to detect whether a write-write conflict has occurred is complex. Three techniques that you can use to detect write-write conflicts under optimistic locking include timestamps, counters, and state comparison.

Deadlock occurs when a locking process is established, either pessimistic or optimistic, but multiple locks are required and the protocol for obtaining these locks is not well-defined. This situation is one of the larger concerns of concurrent programming.

Deadlock

| Pessimistic Locking | | Deadlock | |
|-------------------------|-----------------|--------------------------|--------------------------|
| Txn #1 | Txn #2 | Txn #1 | Txn #2 |
| Lock row #1 | ----- | Lock row #1 | ----- |
| ----- | Wait for row #1 | ----- | Lock row #2 |
| Read and/or Change | --- | Attempt to Lock row #2 | --- |
| Read and/or Change | --- | Wait because It's locked | --- |
| Commit = releases locks | --- | --- | attempt to lock row #1 |
| --- | Lock row #1 | --- | Wait because it's locked |
| | | Wait | Wait |

ORACLE

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

What Do You Think

- An eCommerce inventory system was initially implemented using pessimistic locking. Transactions took one second to complete.
- Since two or more users frequently would hit the same record on the database, one user would block waiting for the others. So, if 100 users tried to hit that same record, the 99th user waited 99 seconds. (There were huge discounts offered on certain items, which intensified the contention for the database record.)
- The architects changed to an asynchronous model using SOA/BPM, where the user submitted the request and was done. If the system blocked, it would retry on its own.
- Horizontal and vertical scaling would not help in this situation, since the same record was being hit concurrently.
- What would you do?
- How would you solve this without a SOA/BPM solution, using only Java EE 6 architecture?



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

Isolation Levels

SQL3 isolation levels also lead to problems that you must address. Isolation levels allow for concurrent access to data under controlled conditions. Four types of access might occur:

- Lost update
- Nonrepeatable read
- Dirty read
- Phantom read



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

Lost update: One transaction overwrites the update of another transaction. This is also known as a race condition. Repeatable reads and Serializable isolation can prevent lost updates.

Non-repeatable read: One transaction reads data, a second transaction modifies the data, and the first transaction rereads the data and sees the changes.

Dirty read: One transaction is allowed to read the uncommitted data of another transaction.

Phantom read: One transaction reads a row inserted by another transaction that has not been committed.

You handle each of the isolation problems by setting the isolation level of the connection to one of the four following levels:

READ_UNCOMMITTED: Allows dirty, nonrepeatable, and phantom reads

READ_COMMITTED: Prevents dirty reads, but allows nonrepeatable and phantom reads

REPEATABLE_READ: Prevents dirty and nonrepeatable reads, but allows phantom reads

SERIALIZABLE: Prevents dirty, nonrepeatable, and phantom reads

A more restrictive control on data reads leads to higher reliability, and a less restrictive control improves scalability, availability, and performance.

Transaction (Txn) Isolation Phenomena

| Lost Update | | Non-Repeatable Read | |
|-------------|--------|---------------------|--------|
| Txn #1 | Txn #2 | Txn #1 | Txn #2 |
| --- | Read | --- | Read |
| Read | --- | Read | --- |
| Change | --- | --- | Change |
| --- | Change | Read | --- |
| Commit | --- | --- | Commit |
| --- | Commit | Commit | --- |



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

The slide shows the flow of activities for two transactions in each of the four types of access.

Transaction (Txn) Isolation Phenomena

| Dirty Read | | Phantom Read | |
|------------|----------|--------------|-----------|
| Txn #1 | Txn #2 | Txn #1 | Txn #2 |
| Read | --- | Select... | --- |
| Change | --- | --- | Insert... |
| --- | Read | --- | Update... |
| Rollback | --- | Select... | --- |
| ---- | Rollback | | |

ORACLE

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

Transactions Already Committed

- Consider the problem of removing the effects of a previously committed transaction.
- Because the transaction is already committed, there is no way to use the transaction rollback functionality to correct the error.
- Consider using a compensating transaction as a recovery mechanism. However, this is not always possible.
(Example: sending an email to your boss saying: “I quit!”)

ORACLE

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

You can use compensating transactions when an underlying resource manager does not support transaction rollback. This approach allows nontransactional resources to participate in a transaction.

Between the time that a transaction and its compensating transaction are issued, the resource is in an inconsistent state. Clients that act on the inconsistent data could cause reliability problems for the application, or worse, for the enterprise system as a whole.

A properly executed compensating transaction requires twice the remote access of a single transaction, and this impacts scalability and performance.

Creating the Transaction Model

Use the following procedure to prepare the transaction model of the application:

1. Examine the use cases.
2. Identify activities that access transactional resources.
3. Assign a transactional scope to each activity.
4. Use patterns where appropriate.
5. Examine the read-to-write ratios of the data involved in particular use cases.
6. Integrate transactional operations into your prototype early.
7. Tune the application server parameters to make the best use of the underlying transactional resource.



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

Use the following procedure to prepare the transaction model of the application:

1. Examine the use cases just as you would with other aspects of the system development process.
2. Use details in activity diagrams or scenario descriptions to identify activities that access transactional resources.
3. Assign a transactional scope to each activity.
4. Use patterns where appropriate. If the resource access is a sequence of fine-grained transactional operations, use a pattern, such as Session Façade to limit the propagation of transactional context within the EJB container.
5. Examine the read-to-write ratios of the data involved in particular use cases. High values of read to write ratios (10 to 100, or even more) are good candidates for data caching techniques. For example, an online catalog service allows customers to browse through inventory to find items that interest them. The catalog data has a high volume of read operations, a low volume of write operations, and a relatively low value of data view change. As a result, you can cache the catalog views in the application. This saves on network capacity and improves end-user performance by eliminating unnecessary database calls.

6. Integrate transactional operations into your prototype early to monitor effects on performance and scalability. When you integrate transactional operations into the architecture prototype early, you can examine the effects of the transaction model on performance and scalability early in the project. Use a good sample of different types of transactional operations (high read ratio, high write ratio, even read-to-write ratio) to gain perspective on network capacity and system capacity use. Then you can focus on tuning the model to meet performance and scalability requirements.
7. Tune the application server parameters to make the best use of the underlying transactional resource .Use the resource's performance profile to tune the parameters. You can also optimize the application by selecting values, such as connection pool size, data cache size, and transaction timeout.

Analyzing Application Transaction Requirements

When you design an application transaction model, you also need to analyze the application transaction requirements, which include looking at:

- Transaction scope
- Container-managed transactions
- Transaction performance characteristics



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

Transaction scoping

Recall that long operations lead to long transactions. As a general rule, you want to avoid long transactions because they can time out, cause exceptions, degrade performance and scalability, and create deadlock. This does not mean that all long transactions are bad. Sometimes an operation is computationally intensive and must be performed as a transaction. In this case, a long transaction is unavoidable. The point is, some options you can use to avoid long transactions include:

- Analyzing the operation to see if it passes the ACID test
- Refactoring the operations and data to allow for a sequence of short transactions
- Using an alternative approach, such as transactional messages, to propagate transaction context

Container-managed transactions (CMT)

Use a CMT service to integrate transaction control into an application. Although this approach is limited to the granularity of a single method, it provides the following benefits to application development:

- The granularity of CMT in Java EE is defined at the method level.
- There is no transaction code to write.
- You can integrate and reuse components without writing additional code.
- Better reusability of component implementation.
- You have a declarative approach to transaction management.
- Containers can use standard transaction management protocols to work cooperatively.

What Do You Think?

- When are container-managed transactions appropriate?
- When are they inappropriate?
- How do you decide?



ORACLE

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

Creating an Application's Transaction Model

Follow these key heuristics to create effective transaction models:

- Use transactions only when necessary.
- If you must use transactions, use the simplest model that you can.
- Increase the complexity of the transaction model only when the increase is required for reliability, scalability, or performance.



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

One of the most difficult aspects of any application is specifying a transaction model that scales, performs well, and yields reliable data. This is especially hard in the enterprise application realm, where many different applications might be using the same data. In addition, these applications can have incompatible transaction models. With proper analysis, you can create transaction models that provide reliability without compromising scalability and performance.

When you examine a transaction model for an application, you must first know what resources are involved. Identify all of the resources that an application requires to fulfill all of its use cases. Then identify which resources are required by each individual use case. Not all of an application's required resources are used in every single use case. When you take advantage of this fact, you can create application transaction models that are efficient and reliable.

After you know which resources an application uses, identify the capability of the resources to participate in transactions. Ask the following questions:

- Are transactions supported on the resource? If so, what are the transaction control APIs?

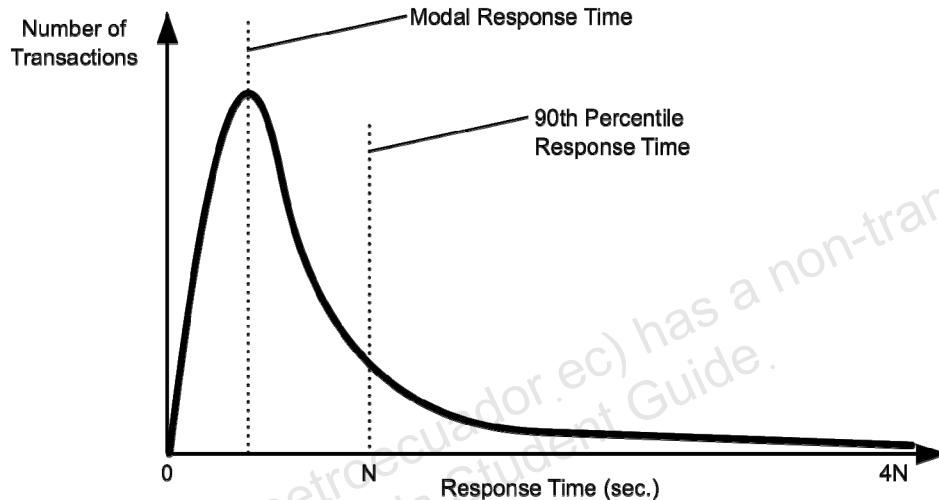
- Is the transaction management system compatible with the application technology? For example, is the transaction management system compatible with the Java EE technology?
- If the resource is not compatible with the application technology, you can manage the resource through a wrapper or another technique, such as a messaging system or a DAO pattern. If you use a programmatic technique to provide transactional access to the resource, you might require support of a compensating transaction mechanism.
- What usage policies are already in place for these resources? Does the system use an optimistic or a pessimistic policy for maintaining data consistency?
- The type of policy has a direct impact on resource availability and application performance and scalability.
- What existing isolation levels and read-write locks does the resource require to keep data consistent?
- This information is valuable when you formulate the isolation levels and data use policies, such as data caching, for your application.

Java EE technology provides for the integration of SQL databases and a number of other resource types directly into the transaction system. Not all external resources are supported, however, and a number of additional techniques are required to handle those situations.

Transaction Performance Metrics

The three major performance metrics for transactional systems:

- Response time
- Throughput
- Price to performance ratio



ORACLE

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

The response time is defined as the time interval for a transactional request and response cycle to complete. It illustrates a typical load and response curve. As small or moderate load level increases the response time is constant, statistically speaking. However, a degradation begins to set in as the load exceeds the capacity of the system.

The throughput of a transactional system is the number of transactions the system can execute per unit time. This metric gives an indication of the load capacity of the system. The throughput is calculated by dividing the number of completed transactions by a time interval over which the transactions are executed.

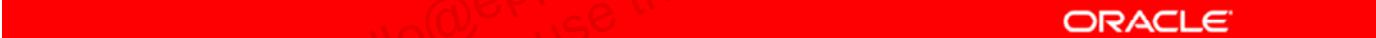
The price-to-performance ratio is the ratio of the cost of ownership of the transactional system and the number of transactions the system can complete. This value is based on a time interval so that you can reasonably estimate the cost of ownership and the transaction volume. The end result of this calculation is an average cost per transaction to provide transactional support.

To plan a suitable transaction model, you must know or assume the average and peak concurrent loads, average transaction response time, and worst-case transaction response time. You can guess at these values, estimate them based on market information, or even calculate them from empirical data.

Another important value for transaction performance is the transaction timeout value. A timeout value on a transaction helps improve performance and avoid application killers, such as deadlock. The timeout value you select must be reasonable given the load and performance characteristics of the underlying system.

Lesson Agenda

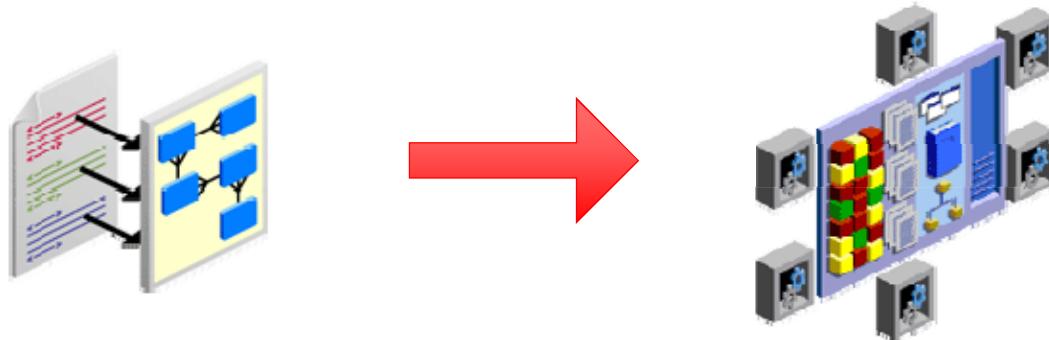
- Describing network communication guidelines
- Justifying the use of transactions
- Planning system capacity



ORACLE

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

Planning System Capacity



High-level Requirements

Detailed Components



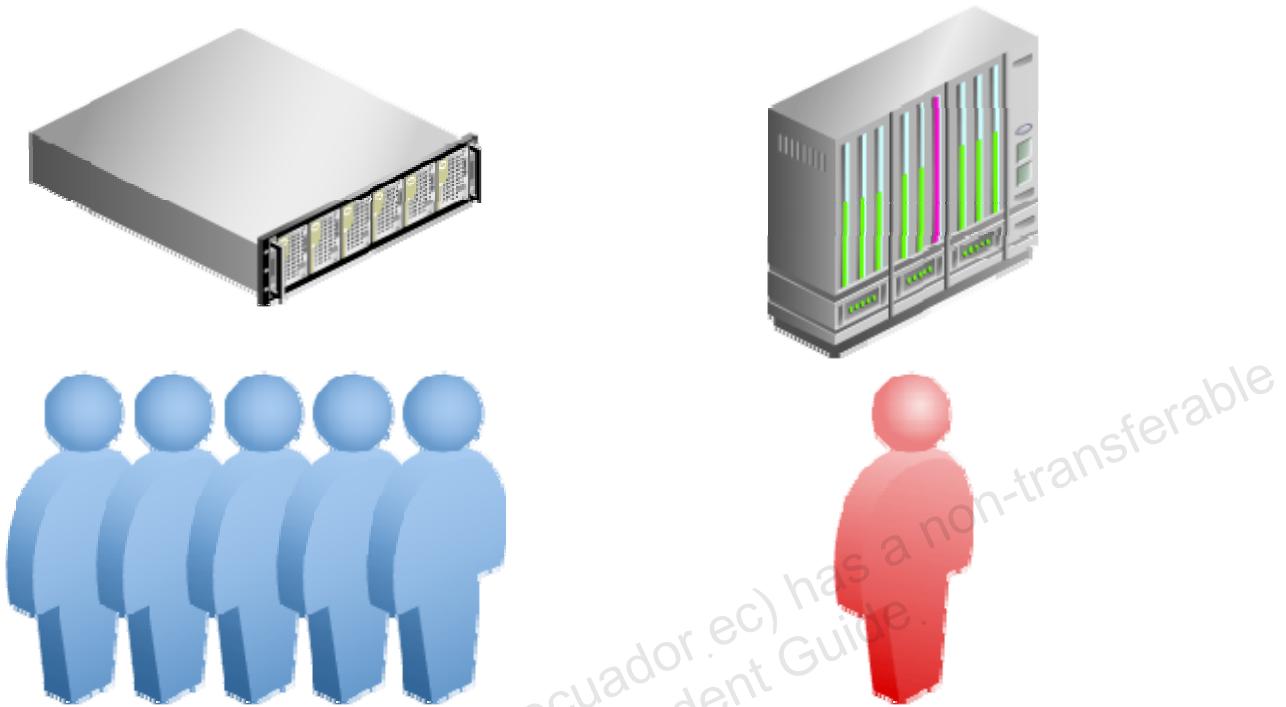
ORACLE®

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

As the system architecture moves from a top-level, conceptual, requirements specification to a collection of realizable components, the perspective changes.

Planning a system's capacity is another example of balancing the cost requirements of quality against the available funds within the project budget. In other words, like most other problems in architecture, capacity planning and management is an optimization problem.

Two Extremes for System Capacity



ORACLE®

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

Individual systems must be the right size for the business need. You could consider two extremes for system capacity:

One server serves everyone

This solution is the minimal system implementation. This solution suffers from a complete lack of additional capacity to accommodate items, such as spikes, in-usage plans, and long-term growth plans.

Everyone gets their own mainframe

This solution is extremely cost prohibitive for the project and the company as a whole.

With good techniques and the proper set of tools, an architect can specify the requirements of the individual systems. These systems are the result of good engineering processes that are based on profile data and good management techniques that are based on sound economics.

System Load Characteristics

Several factors that contribute to the quality of service that a system delivers:

- Transaction load
- Transaction rate
- Number of concurrent requests
- Request response time
- Utilization allowance
- Message size and rate (may matter more than transaction load/rate)



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

In their book on planning Internet services, Cockcroft and Walker outline several factors that contribute to the quality of service that a system delivers. Examples include load factors, utilization profiles, system architecture, and data resource layout. Of particular interest at this point are load factors, which you use to plan the capacity requirements of a particular service within the system.

Java EE technology provides a degree of insulation from the problems of scalability. This is because the specification allows individual container vendors to implement replication and failover techniques within the container, which relieves the application developers from the effort. However, it is still relevant to determine an appropriate hardware capacity for the initial deployment of the system, otherwise immediate scaling, or wasteful overcapacity might result.

Estimating Transaction Load

Transaction load is a measure of the following information:

- How many requests are being processed?
- How many of those requests require transaction management?
- How much of the request is transactional?



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

When you analyze the transaction over the entire application, the data provides you with the approximate percentage of incoming user requests that actually require transaction management. When you use this value as a basis for workload, you focus the computing resources of the system on managing the transactions instead of on processing the user requests.

Estimating Transaction Rate

- Transactional rate is based on the average number of transactions processed in a sample time period.
- The transaction rate value is statistical in nature.
- You must also address the peak value as part of your analysis.



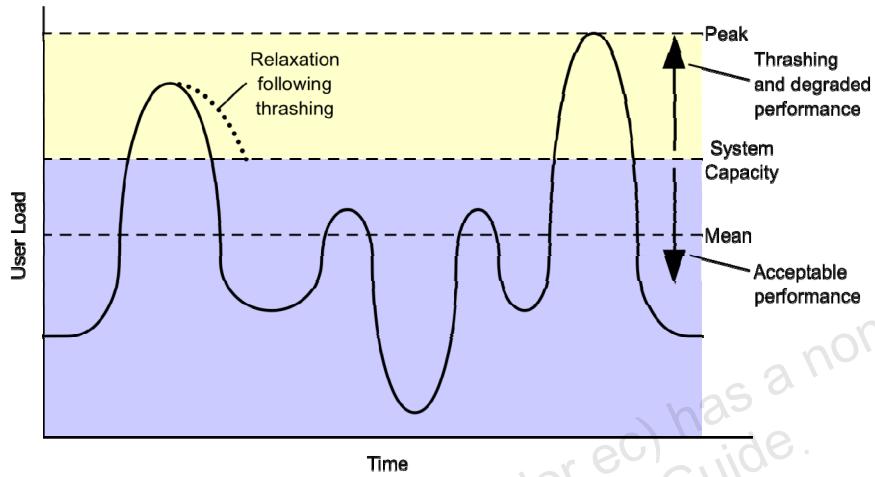
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Capacity requirements also depend on the transactional rate of the system. Transactional rate is based on the average number of transactions that are processed in a sample time period. Many vendors publish performance data on their servers that is based on this number. The TPC-C benchmark test (see [http://www\(tpc.org\)](http://www(tpc.org))) is an example of the performance of a server, based on transaction rate. TPC-C uses a metric of transactions per minute, with 90 percent of the transactions processed in under five seconds (tpmC).

The value of the transaction rate is statistical in nature and is not meant to capture the instantaneous maximum value or peak value of the transaction processing rate. You must also address the peak value as part of your analysis. If a system is designed to address only average loads and not peak loads, a burst of activity that occurs regularly could lead to a recurring loss of availability.

Estimating Client Load

A profile curve that shows average and peak values.



ORACLE

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

The client load describes

- The number of concurrent user requests that are processed
- The number of incoming requests that must be processed
- The number of requests that are currently being processed

You need to examine both the average and peak values of concurrent requests.

The average number of concurrent user (client) requests that are processed is another important factor in determining capacity requirements. This average is based on two numbers:

- The number of incoming requests that must be processed
- The number of requests that are currently being processed

These numbers vary over time and, as with the transaction rate, you need to examine both the average value of concurrent requests and the peak value of concurrent requests.

When you consider the average value of concurrent requests, you can plan capacity that is sufficient to carry your load over time. Consequently, you do not overspend trying to buy a system that would carry the peak load 100 percent of the time. On the other hand, when you consider the peak load, you ensure that the system is more responsive during periods of high use, even though it might sit idle during periods of low use.

It is a bad choice to invest in a system capable of carrying only the average load because the system provides only the bare minimum of capacity. On the other hand, operating at peak load might turn out to be a dramatic waste of resources and budget funds. As an architect, you can outline the different system capacity options and their cost impacts to the project.

- Consider the initial outlay of funds to implement all of the options.
- Profile the loss of revenue incurred if the system is over-used and does not perform.

If a system is overloaded, it spends most of its time performing resource management tasks, such as context switching and memory paging. This behavior is known as *thrashing* and seriously degrades the quality of service provided by a server (refer again to the graphic on the previous page). However, if you can resolve thrashing and restore service levels within a reasonable amount of time after the peak load occurs, a decision to accept a reasonable amount of thrashing can be more cost effective in some applications. The choice to allow some thrashing is a business decision that you need to present for the stakeholders to decide.

Sizing the System

- 70 Percent CPU Utilization Rule: A 70 percent CPU utilization indicates that the system is supporting a workload that might be considered excessive.
- 90 Percent CPU Utilization Rule: A 90 percent CPU utilization indicates that the system is supporting a workload that is critical.
- Other system components also exhibit warning and critical limit values.
- The values used for warning and critical limits can vary based on the type of system used and the types of tasks it is performing.



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

After you know the load requirements of the system, you have a good idea of how much horsepower you need to make that system function. There are some general, well-accepted heuristics that ensure that the system remains healthy and the CPU capacity utilization remains good without being excessive. These heuristics are known as the 70 percent CPU utilization rule and the 90 percent CPU utilization rule.

The 70 percent CPU utilization rule states that a system is supporting a workload that might be considered excessive.

This load value is based on the environment in which the individual server is deployed, but 70 percent CPU utilization is a good warning sign that the system is approaching its maximum utilization levels.

If this value is sustained for any length of time, you might need to consider adding more resources to the system.

The 90 percent CPU utilization rule states that a system is supporting a workload that is critical. In this situation, small utilization spikes could cause the system to overload and begin to offer degraded service. Systems that operate at 90 percent CPU utilization indicate that a major system need is not being met. For example, a system might have taken over the

workload of a failed system in its cluster, which resulted in a raised utilization rate. Another example might be that all the systems are operating at 90 percent CPU utilization, which means that there is a serious need for horizontal scaling or the purchase of a higher-grade server system.

Other system components also exhibit warning and critical limit values, but they might not be the same as the 70 percent and 90 percent values commonly used for CPUs. Also, the actual values used for warning and critical limits can vary based on the type of system used and the types of tasks it is performing.

Moreover, even though the 70 percent and 90 percent CPU utilization rules are a good gauge for when a system is approaching its upper limit of capacity, you should also design a measure of growth capability into your system size. If a project team purchases a system that is going to begin its life cycle at the upper operational limits, they have laid the groundwork for project failure.

Although not part of Java EE technology, a number of load testing systems are available for Java technology software. These systems can simulate a realistic mix of client activity, so they help you to determine the behavior of the system under a realistic load.

Also, consider other forms of sizing like Disk usage and swapping.

What Do You Think?

“Capacity planning is complicated by your brain thinking linearly about a computer system that operates *nonlinearly*.

Capacity planning is about setting expectations. Even wrong expectations are better than no expectations!

Hardware is cheaper today, but a truck-load of PCs won't help one iota if all or part of the application runs single-threaded.

If the network is out of bandwidth or has interminable latencies, fix it! Then we'll talk performance of your application.

You never remove a bottleneck, you just shuffle it around.

All benchmarking is institutionalized cheating.”

- Dr. Neil J. Gunther, *Guerrilla Capacity Planning*



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

Planning Scalability

- After you have determined the capacity requirements for current operation levels, you can use these initial requirements to design a prototype.
- Then you can review this prototype and modify it to allow for growth at minimal cost.
- The essence of scalability focuses on the question, “Can the use of your system increase without demanding too much revision effort or without dramatically increasing the cost of the system?”



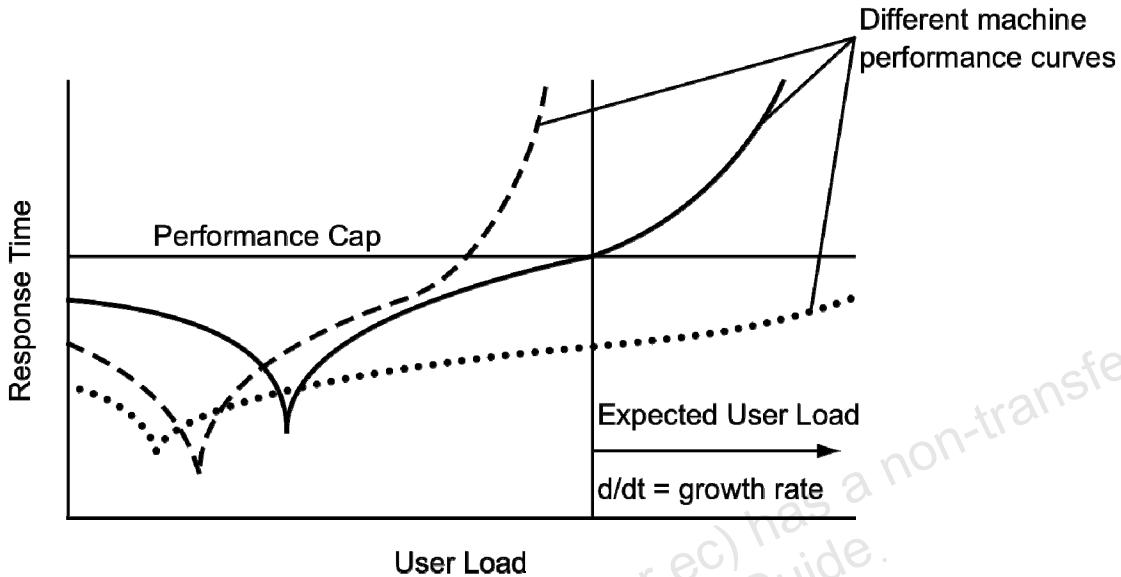
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

To design a scalable system, you must know the long-term growth plans for the system, including:

- What are the customer expectations?
- What are market conditions?
- What growth is expected or desired?

Answers to these questions might not coincide, but it is not your job as an architect to make them coincide. Understand the alternatives and present options. Say to the stakeholders, “Here is what you need to operate at current levels. Here is what I can give you with the money you have allocated. Here is where you want and expect to be. Do they fit together?” That is your job as the architect.

Machine Performance Profiles



ORACLE

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

The slide shows the effects that user load has on a system's performance. Initially, the curve shows improvement as the memory cache aids the performance of a small number of users. At a certain load value, the cache is no longer able to provide performance benefits and the curve begins to climb.

If you profile several machines in the same way, you can compare these machine configurations to architectural goals, such as performance limits and scalability factors. For example, the solid line in shows a marginal system with no growth potential, the dotted line shows a system that exceeds requirement, and the dashed line shows an unacceptable system.

Cloud Computing

“Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”

- *NIST Definition of Cloud Computing v15.*



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

Cloud computing is a significant advancement in the delivery of information technology and services. By providing on demand access to a shared pool of computing resources in a self-service, dynamically scaled and metered manner, cloud computing offers compelling advantages in speed, agility and efficiency. Today, cloud computing is at an early stage in its lifecycle, but it is also the evolution and convergence of several trends that have been driving enterprise data centers and service providers over the last several years.

Cloud computing builds off a foundation of technologies such as grid computing, which includes clustering, server virtualization and dynamic provisioning, as well as SOA shared services and large- scale management automation.

Pros:

- Fast startup, simplicity
- Scalability
- Business agility
- Faster product development

Cons:

- Bandwidth could break the budget
- Application performance could suffer, when network latency is taken into account

- Concerns over data security. Data might not be “ready for the cloud”: type of data, who can see it.
- Enterprise might be too big to fail. Very large enterprises might not see benefit moving to a “public” cloud.
- In-house expertise may be lacking
- What happens to your cloud apps when you, or your cloud provider, have an outage or performance suffers? One of the misconceptions of cloud hosting is that it’s hosted “in the sky and not in a datacenter.” Cloud hosting resides in a single datacenter.

Public versus Private Clouds

While public clouds may seem attractive, they are subject to the issues described above. While companies want the benefits of cloud computing, they find the control and security issues of public clouds to be a major drawback. Large enterprises may find using a private cloud a better choice, “bursting” into a public cloud as necessary to handle demand.

This table, from

http://wikibon.org/wiki/v/Private_Cloud_is_more_Cost_Effective_than_Public_Cloud_for_Organizations_over_1B

compares private and public cloud costs (costs per seat, organizations \$1 billion and under):

| Area | Private Cloud | Public Cloud |
|----------------------|----------------|----------------|
| Management | \$167 | \$418 |
| Development Staff | \$334 | \$334 |
| Operational Staff | \$89 | \$76 |
| Help Desk | \$67 | \$67 |
| Outsourcing | \$22 | \$609 |
| Application Software | \$134 | \$134 |
| Network | \$45 | \$89 |
| Storage | \$45 | \$4 |
| Server | \$111 | \$11 |
| Total | \$1,114 | \$1,752 |

Additional Resources

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

- Cockcroft, Adrian and Bill Walker. *Capacity Planning for Internet Services*. Sun Microsystems, 2000.
- High Availability Fundamentals,
www.sun.com/blueprints/1100/HAFund.pdf
- Gunther, Neil. *Guerrilla Capacity Planning, A Tactical Approach to Planning for Highly Scalable Applications and Services*. Springer 2007.
- Gunther, Neil. *The Practical Performance Analyst*. iUniverse, 2000.



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

http://www.enterprise-architecture.info/EA_Services-Oriented-Enterprise.htm

Capacity Planning: An Essential Tool for Managing Web Services,
cs.gmu.edu/~menasce/papers/IEEE-ITProfessional-July2002.pdf

<http://highscalability.com/>

Quiz

In a situation where performance of an existing Enterprise Application is suffering when the total number of users on the application reaches its top point, what factor do you need to address in the new design for this application?

- a. Transaction load
- b. Request latency
- c. Bandwidth
- d. Chattiness



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

Answer: c

Quiz

Given that you have determined that network performance is suffering due to multiple round-trips between the user interface and the enterprise server, what steps can you take to improve performance? (Choose all that apply.)

- a. Reduce the number of database transactions
- b. Use load balancers
- c. Create coarse-grained services
- d. Aggregate data



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

Answer: c, d

Quiz

Which of the following situations would you require transactions?

- a. Two users reading data from the same database table
- b. Two users accessing an e-commerce site
- c. Two users updating an inventory record
- d. Two users recording their time cards



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

Answer: c

– the other situations either are reading, not writing, or in the case of d, have separate transaction spaces.

Quiz

When considering system capacity for a complex, multiuser, transaction intensive enterprise application of the following factors should you consider? (Choose all that apply.)

- a. Transaction rate and load
- b. Request response time
- c. Message size
- d. Database table size



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

Answer: a, b, c

Summary

In this lesson, you should have learned how to:

- Identify key risk factors in distributed enterprise systems
- Design a flexible object model
- Describe the guidelines of creating a network model
- Justify the use of transactions
- Plan system capacity

Practice 6 Overview: Consider Network, Transaction and Capacity Planning for the Architecture

This practice covers the following topics:

- Consider how to refine the architecture to minimize network load.
- Identify transaction boundaries and the effect of transactions on the architecture.
- Weigh options for capacity planning.



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

Java EE 6 Overview

ORACLE

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

Objectives

After completing this lesson, you should be able to:

- Describe the new features in Java EE 6
- Describe the impact of Java EE 6 features on J2EE and Java EE 5 architectures

Discussion Questions

- What version of the Java Enterprise edition platform do you currently use (J2EE, Java EE 5, Java EE 6?)
- If you are not using Java EE 6, do you have plans to adopt or migrate to it?



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

Java EE 6 Goals

- The most important goal of the Java EE 6 platform is to simplify development by providing a common foundation for the various kinds of components in the Java EE platform.
- Developers benefit from productivity improvements with more annotations and less XML configuration, more Plain Old Java Objects (POJOs), and simplified packaging.



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

More Flexible Technology Stack

Over time, the Java EE platform has gotten big, in some cases too big for certain types of applications. To remedy this, Java EE 6 introduces the concept of profiles, configurations of the Java EE platform that are designed for specific classes of applications. A profile may include a subset of Java EE platform technologies, additional technologies that have gone through the Java Community Process, but are not part of the Java EE platform, or both. Java EE 6 introduces the first of these profiles, the Web Profile, a subset of the Java EE platform designed for web application development. The Web Profile includes only those technologies needed by most web application developers, and does not include the enterprise technologies that these developers typically don't need.

In addition, the Java EE 6 platform has identified a number of technologies as candidates for pruning. These candidates include technologies that have been superseded by newer technologies or technologies that are not widely deployed. Pruning a technology means that it can become an optional component in the next release of the platform rather than a required component.

Enhanced Extensibility

Over time, new technologies become available that are of interest to web or enterprise application developers. Rather than adding these technologies to the platform—and growing the platform without bounds—Java EE 6 includes more extensibility points and more service provider interfaces than ever before. This allows you to plug in technologies—even frameworks—in your Java EE 6 implementations in a standard way. Once plugged in, these technologies are just as easy to use as the facilities that are built into the Java EE 6 platform. Particular emphasis on extensibility has been placed on the web tier. Web application developers often use third-party frameworks in their applications. However, registering these frameworks so that they can be used in Java EE web applications can be complicated, often requiring developers to add to or edit large and complex XML deployment descriptor files. Java EE 6 enables these frameworks to self-register, making it easy to incorporate and configure them in an application.

Further Ease of Development

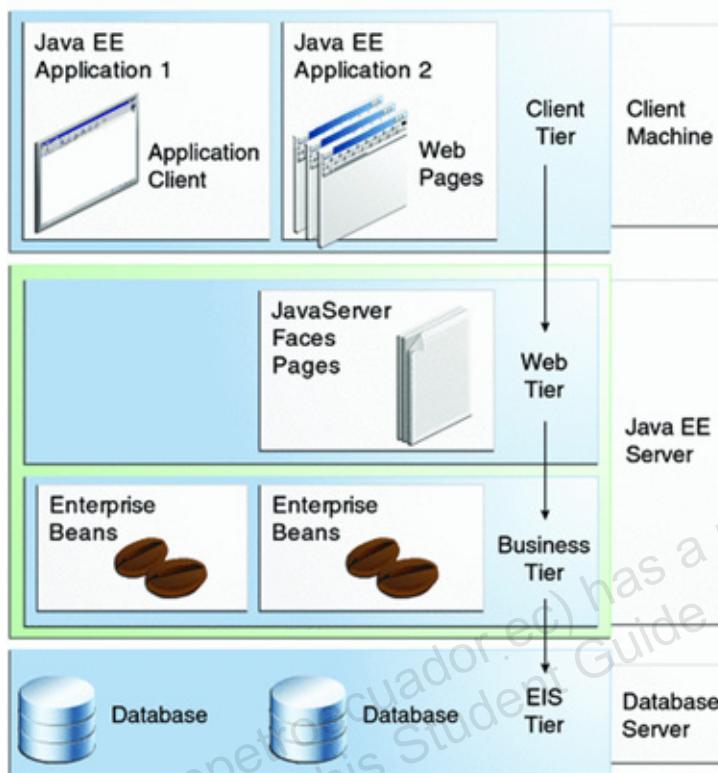
Java EE 5 made it significantly easier to develop web and enterprise applications. For instance, Java EE 5 introduced a simpler enterprise application programming model based on Plain Old Java Objects (POJOs) and annotations, and eliminated the need for XML deployment descriptors. In addition, Enterprise JavaBeans (EJB) technology was streamlined, requiring fewer classes and interfaces and offering a simpler approach to object-relational mapping by taking advantage of the Java Persistence API (informally referred to as JPA).

Java EE 6 makes it even easier to develop enterprise or web applications. Usability improvements have been made in many areas of the platform. For example, you can use annotations to define web components such as servlets and servlet filters. Furthermore, a set of annotations for dependency injection has been standardized, making injectable classes much more portable across frameworks. In addition, Java EE application packaging requirements have been simplified. For example, you can add an enterprise bean directly to a web archive (WAR) file. You no longer need to package an enterprise bean in a Java archive (JAR) file and then put the JAR file in an enterprise archive (EAR) file.

Powerful New Technologies, including the following:

- Java API for RESTful Web Services (JAX-RS)
- Managed Beans
- Contexts and Dependency Injection for the Java EE Platform (JSR 299), informally known as CDI
- Dependency Injection for Java (JSR 330)
- Bean Validation (JSR 303)
- Java Authentication Service Provider Interface for Containers (JASPI)
- New features for Enterprise JavaBeans (EJB) components
- New features for servlets
- New features for JavaServer Faces components
- New features for the Java Persistence API (JSR 317)

Distributed Multitiered Applications



ORACLE

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

The Java EE platform uses a distributed multitiered application model for enterprise applications. Application logic is divided into components according to function, and the application components that make up a Java EE application are installed on various machines, depending on the tier in the multitiered Java EE environment to which the application component belongs.

The Java EE application parts shown in above are presented in Java EE Components:

- Client-tier components run on the client machine.
- Web-tier components run on the Java EE server.
- Business-tier components run on the Java EE server.
- Enterprise information system (EIS)-tier software runs on the EIS server.

Security

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

Java EE Components

Java EE applications are made up of components. A Java EE component is a self-contained functional software unit that is assembled into a Java EE application with its related classes and files and that communicates with other components.

The Java EE specification defines the following Java EE components.

- Application clients and applets are components that run on the client.
- Java Servlet, JavaServer Faces, and JavaServer Pages (JSP) technology components are web components that run on the server.
- Enterprise JavaBeans (EJB) components (enterprise beans) are business components that run on the server.

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

Application Clients

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

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

The JavaBeans Component Architecture

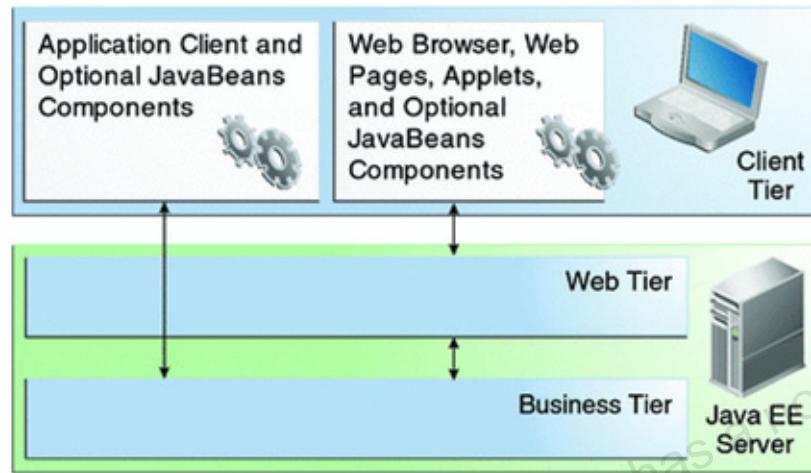
The server and client tiers might also include components based on the JavaBeans component architecture (JavaBeans components) to manage the data flow between the following:

- An application client or applet and components running on the Java EE server
- Server components and a database

JavaBeans components are not considered Java EE components by the Java EE specification.

JavaBeans components have properties and have get and set methods for accessing the properties. JavaBeans components used in this way are typically simple in design and implementation but should conform to the naming and design conventions outlined in the JavaBeans component architecture.

Java EE Server Communications

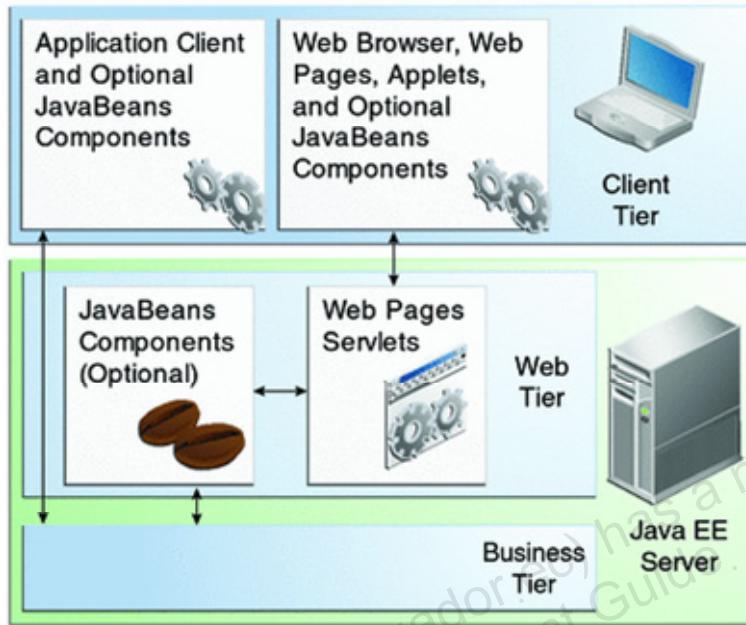


ORACLE

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

The slide shows the various elements that can make up the client tier. The client communicates with the business tier running on the Java EE server either directly or, as in the case of a client running in a browser, by going through web pages or servlets running in the web tier.

Web Tier and Java EE Applications



ORACLE

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

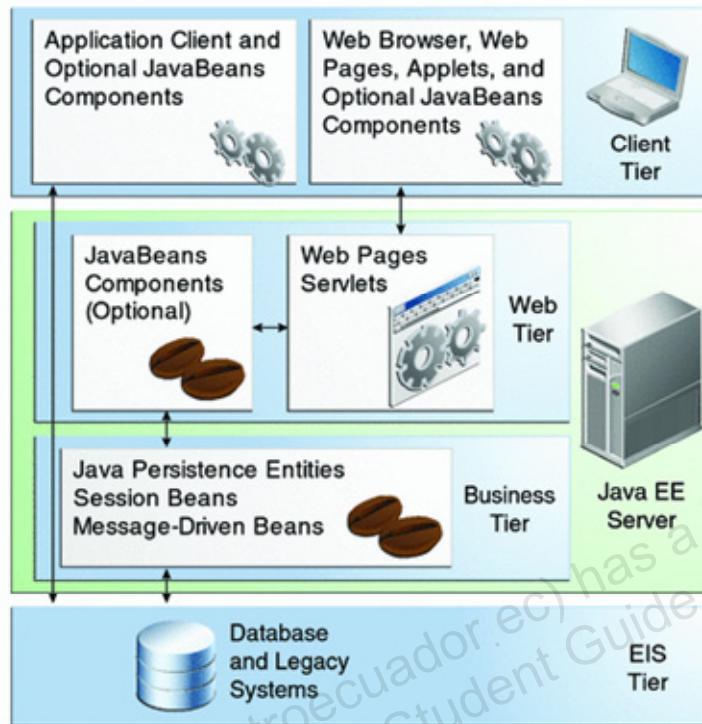
Web Components

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

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

As shown in the slide, the web tier, like the client tier, might include a JavaBeans component to manage the user input and send that input to enterprise beans running in the business tier for processing.

Business and EIS Tiers



ORACLE

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

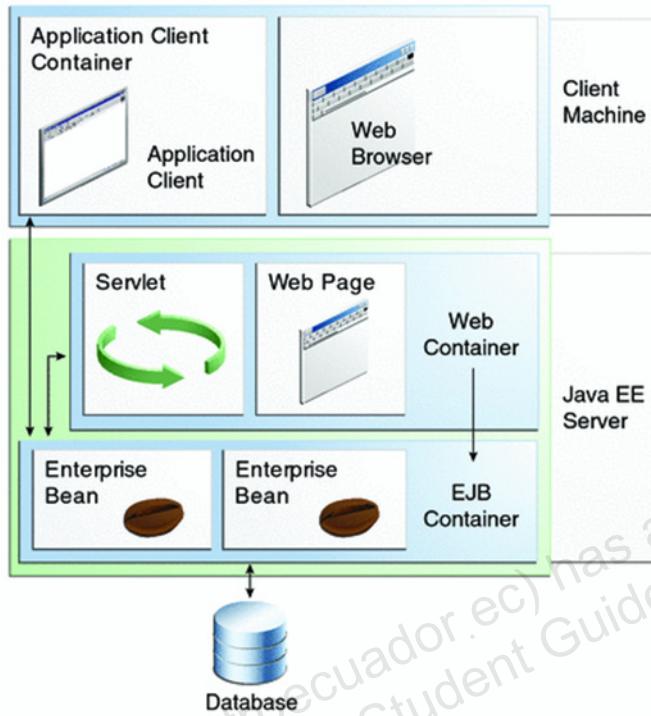
Business Components

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

Enterprise Information System Tier

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

Java EE Containers



ORACLE

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

Container Services

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

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

- The Java EE security model lets you configure a web component or enterprise bean so that system resources are accessed only by authorized users.
- The Java EE transaction model lets you specify relationships among methods that make up a single transaction so that all methods in one transaction are treated as a single unit.
- JNDI lookup services provide a unified interface to multiple naming and directory services in the enterprise so that application components can access these services.

- The Java EE remote connectivity model manages low-level communications between clients and enterprise beans. After an enterprise bean is created, a client invokes methods on it as if it were in the same virtual machine.

Because the Java EE architecture provides configurable services, application components within the same Java EE application can behave differently based on where they are deployed. For example, an enterprise bean can have security settings that allow it a certain level of access to database data in one production environment and another level of database access in another production environment.

The container also manages nonconfigurable services, such as enterprise bean and servlet lifecycles, database connection resource pooling, data persistence, and access to the Java EE platform APIs.

Container Types

The deployment process installs Java EE application components in the Java EE containers as illustrated in the slide.

- **Java EE server:** The runtime portion of a Java EE product. A Java EE server provides EJB and web containers.
- **Enterprise JavaBeans (EJB) container:** Manages the execution of enterprise beans for Java EE applications. Enterprise beans and their container run on the Java EE server.
- **Web container:** Manages the execution of web pages, servlets, and some EJB components for Java EE applications. Web components and their container run on the Java EE server.
- **Application client container:** Manages the execution of application client components. Application clients and their container run on the client.
- **Applet container:** Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.

New Features: EJBs

- No-interface view
- Singletons
- Asynchronous session bean invocation
- Simplified Packaging
- Schedule-based Timers
- EJB Lite



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

No-Interface View

The EJB 3.0 local client view is based on a plain old Java interface (POJI) called a local business interface. A local interface defines the business methods that are exposed to the client and that are implemented on the bean class. This separation of interface and implementation is sometimes unnecessarily cumbersome and adds little value—this is especially true for fine-grained components that are accessed locally from clients within the same module. EJB 3.1 simplifies this approach by making local business interfaces optional. A bean that does not have a local business interface exposes a no-interface view. Now you can get the same enterprise bean functionality without having to write a separate business interface.

The no-interface view has the same behavior as the EJB 3.0 local view, for example, it supports features such as pass-by-reference calling semantics and transaction and security propagation. However, a no-interface view does not require a separate interface, that is, all public methods of the bean class are automatically exposed to the caller.

Singletons

A singleton bean, also known as a singleton, is a new kind of session bean that is guaranteed to be instantiated once for an application in a particular Java Virtual Machine.

With singletons, you can easily share state between multiple instances of an enterprise bean component or between multiple enterprise bean components in the application. Consider, for example, a class that caches data for an application. You might define the class as a singleton and in doing so, ensure that only one instance of the cache and its state is shared in the application.

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.

Simplified Packaging

EJB 3.1, you can place enterprise bean classes in the .war file along with web tier components. You don't have to put EJB classes in the ejb-jar file.

Timer Service

Applications that model business work flows often rely on timed notifications. 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. For example, you could set timers to go off at 10:30 a.m. on May 23, in 30 days, or every 12 hours. Enterprise bean timers are either programmatic timers or automatic timers.

EJB Lite

For many applications, EJB technology offers a lot more functionality than those applications really need. The typical application that uses EJB only needs a subset of the features provided by the technology. EJB Lite meets the needs of these applications with a small subset of the features in EJB 3.1 centered around the session bean component model. EJB Lite offers the following features:

- Stateless, stateful, and singleton session beans
- Local EJB interfaces or no interfaces
- Interceptors
- Container-managed and bean-managed transactions
- Declarative and programmatic security
- Embeddable API

More New Features in EJB 3.1

EJB 3.1 delivers more features and enhancements than those covered in the previous sections. For example, it includes an embeddable API and container for use in the Java SE environment. These makes it easy to test EJB components outside a Java EE container, and more generally, in Java SE. For another example, the introduction of singletons in EJB 3.1 offers a convenient way for EJB applications to receive callbacks during application initialization or shutdown.

New Features: Java Servlet Technology

- Annotation support
- Asynchronous support
- Ease of configuration
- Enhancements to existing APIs
- Pluggability



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

Some of the most significant enhancements made in Java EE 6 appear in the web tier. As mentioned earlier, one of the goals of Java EE 6 is to make the platform more extensible, and two key improvements in the area of extensibility are web fragments and shared framework pluggability. These two new features are provided in Java EE 6 by Servlet 3.0 technology. Servlet 3.0, JSR 315, the latest version of Servlet technology, offers some other valuable enhancements such as support for asynchronous processing and support for annotations.

Support for Web Fragments in Servlet 3.0

A web fragment can be considered a logical segment of a web.xml file. There can be multiple web fragments, each representing a logical segment, and the set of web fragments can be viewed as constituting an entire web.xml file. This logical partitioning of the web.xml file enables web frameworks to self-register to the web container. Each web framework that you use in a web application can define in a web fragment all the artifacts that it needs, such as servlets and listeners, without requiring you to edit or add information in the web.xml file.

Web fragments enable web frameworks to self-register, eliminating the need for you to register them through deployment descriptors.

Annotations

The simpler annotation-based programming model that was introduced in Java EE 5 has been extended to other types of Java EE components, such as servlets and JSF components. For example, instead of using a deployment descriptor to define a servlet in a web application, all you need to do is mark a class with the `@WebServlet` annotation.

Asynchronous Processing in Servlet 3.0

Servlet 3.0 introduces support for asynchronous processing. With this support, a servlet no longer has to wait for a response from a resource such as a database before its thread can continue processing, that is, the thread is not blocked. This support enables long-lived client connections such as those in chat room applications. In these types of applications you don't want a server thread to be blocked for a long period of time serving a request from a single client. You want the servlet to process a request from the client and then free up the server thread as quickly as possible for other work. Among its benefits, support for asynchronous processing makes the use of servlets with Ajax more efficient.

Servlets and servlet filters that support asynchronous processing must be written with the goal of asynchrony in mind. In particular, several long-standing assumptions about the order in which some methods will be called do not apply for asynchronous processing.

Shared Framework Pluggability

Web fragments and annotations are not the only way that Servlet 3.0 allows you to extend a web application. You can also plug in shared copies of frameworks, such as Java API for XML-Based Web Services (JAX-WS), JAX-RS and JSF, that are built on top of the web container. Servlet 3.0 introduces a new interface called `ServletContainerInitializer` that can be used to plug in a framework.

More New Features in Servlet 3.0 and JSF 2.0

Another new feature of note in Servlet 3.0 enables you to use methods in the `ServletContext` class to programmatically add servlets and servlet filters to a web application during startup. You use the `addServlet()` method to add a servlet to the web application, and the `addFilter()` method to add a servlet filter. The ability to programmatically add servlets and servlet filters at startup is particularly useful to framework writers. In conjunction with the shared framework pluggability feature by which extension libraries can discover classes listed in the `@HandlesTypes` annotation, with this facility web frameworks can configure themselves with no developer intervention.

In addition, Servlet 3.0 works with a number of enhanced security features. For example, in addition to declarative security, Servlet 3.0 offers programmatic security through the `HttpServletRequest` interface. You can, for example, use the `authenticate()` method of `HttpServletRequest` in an application to perform username and password collection, or you can use the `login()` method to direct the container to authenticate the request caller from within an unconstrained request context. For more information about these and other features in Servlet 3.0, see [Servlet 3.0, JSR 315](#).

New Features: JSF 2.0

- Facelets
- Support for Ajax
- Annotations
- Composite Components



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

The latest release of the technology, JSF 2.0, JSR 314, makes UI development for Java EE applications even easier. One area of particular improvement is page authoring. Authoring a JSF page is much easier in JSF 2.0 through the use of the standard JavaServer Faces View Declaration Language, commonly called Facelets.

Facelets

Facelets is a powerful but lightweight declaration language that you can use to present JSF pages. In the Facelets approach, you use HTML-style templates to present a JSF page and to build component trees. Although JSF can be used with different display technologies, most JSF applications use JSP as the display technology. In other words, the UI in a JSF application is typically a JSP page that contains JSF components. However, Facelets offers several advantages over JSP.

Composite Components

Composite components is a new feature in JSF that makes it easy to create customized JSF components. You can create composite components by using JSF page markup, other JSF UI components, or both. And with the help of Facelets, any XHTML page can become a composite component. In addition, composite components can have validators, converters, and listeners attached to them just like the set of UI components provided by JSF. After you create a composite component, you can store it in a library and use it as needed.

Support for Ajax in JSF 2.0

In support of Ajax, JSF's request processing cycle has been expanded to allow partial page updates and partial view traversal. Partial view traversal allows one or more components in a view to be visited, potentially to have them pass through either or both the execute phase or render phase of the request processing lifecycle. This is a key feature in JSF and Ajax frameworks and it allows selected components in the view to be processed, rendered, or both. JSF 2.0 has built-in support for Ajax. With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page.

Some additional enhancements in JSF 2.0 relate to how resources are packaged and handled. JSF 2.0 standardizes where resources are packaged. All resources now go in a resources directory or a subdirectory. Resources are any artifacts that a component may need in order to be rendered properly, such as CSS files or JavaScript files.

New Features: Java API for RESTful Web Services

- The Java API for RESTful Web Services (JAX-RS) defines APIs for the development of web services built according to the Representational State Transfer (REST) architectural style.
- A JAX-RS application is a web application that consists of classes that are packaged as a servlet in a WAR file along with required libraries.



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

Java API for RESTful Web Services (JAX-RS), JSR 311 enables you to rapidly build lightweight web services that conform to the 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. To manipulate these resources, components of the network, clients and servers, communicate through a standardized interface such as HTTP and a small, fixed set of verbs—GET, PUT, POST, and DELETE—and exchange representations of these resources.

RESTful web services are web services built according to the REST architectural style. Building web services with the RESTful approach has emerged as a popular alternative to using SOAP-based technologies thanks to REST's lightweight nature and the ability to transmit data directly over HTTP.

JAX-RS makes it simple to create RESTful web services in Java.

JAX-RS furnishes a standardized API for building RESTful web services in Java. The API contributes a set of annotations and associated classes and interfaces. Applying the annotations to POJOs enables you to expose web resources. This approach makes it simple to create RESTful web services in Java.

New Features: Managed Beans

- Managed Beans, lightweight container-managed objects (POJOs) with minimal requirements, support a small set of basic services, such as resource injection, lifecycle callbacks, and interceptors.
- Managed Beans represent a generalization of the managed beans specified by JavaServer Faces technology and can be used anywhere in a Java EE application, not just in web modules.



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

A managed bean is implemented by a Java class, which is called its bean class. A top-level Java class is a managed bean if it is defined to be a managed bean by any other Java EE technology specification, such as the JavaServer Faces technology specification, or if it meets all the following conditions:

- It is not a nonstatic inner class.
- It is a concrete class or is annotated @Decorator.

It is not annotated with an EJB component-defining annotation or declared as an EJB bean class in ejb-jar.xml.

It has an appropriate constructor. That is, one of the following is the case:

- The class has a constructor with no parameters.
- The class declares a constructor annotated @Inject.

No special declaration, such as an annotation, is required to define a managed bean.

New Features: Context and Dependency Injection

- Contexts and Dependency Injection (CDI help to knit together the web tier and the transactional tier of the Java EE platform.
- CDI is a set of services that, used together, make it easy for developers to use enterprise beans along with JavaServer Faces technology in web applications.
- CDI unifies and simplifies the EJB and JSF programming models.



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

Contexts and Dependency Injection for the Java EE Platform (CDI), JSR 299 is a technology that supplies a powerful set of services to Java EE components. These services allow Java EE components, including EJB session beans and JavaServer Faces (JSF) managed beans, to be bound to lifecycle contexts, to be injected, and to interact in a loosely coupled way by firing and observing events. It allows enterprise beans to replace JSF managed beans in a JSF application.

In essence, CDI helps bridge what was a major gap between the web tier of the Java EE platform and the enterprise tier. The enterprise tier, through technologies such as EJB and JPA, has strong support for transactional resources. For example, using EJB and JPA you can easily build an application that interacts with a database, commits or rolls back transactions on the data, and persists the data. The web tier, by comparison, is focused on presentation. Web tier technologies such as JSF and JavaServer Pages (JSP pages) render the user interface and display its content, but have no integrated facilities for handling transactional resources. Through its services, CDI brings transactional support to the web tier. This can make it a lot easier to access transactional resources in web applications. For example, CDI makes it a lot easier to build a Java EE web application that accesses a database with persistence provided by JPA.

Designed for use with stateful objects, CDI also has many broader uses, allowing developers a great deal of flexibility to integrate various kinds of components in a loosely coupled but typesafe way.

Overview of CDI

The most fundamental services provided by CDI are:

- **Contexts:** The ability to bind the lifecycle and interactions of stateful components to well-defined but extensible lifecycle contexts
- **Dependency injection:** The ability to inject components into an application in a typesafe way, including the ability to choose at deployment time which implementation of a particular interface to inject



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

In addition, CDI provides the following services:

- Integration with the Expression Language (EL), which allows any component to be used directly within a JavaServer Faces page or a JavaServer Pages page
- The ability to decorate injected components
- The ability to associate interceptors with components using typesafe interceptor bindings
- An event-notification model
- A web conversation scope in addition to the three standard scopes (request, session, and application) defined by the Java Servlet specification
- A complete Service Provider Interface (SPI) that allows third-party frameworks to integrate cleanly in the Java EE 6 environment

A major theme of CDI is loose coupling. CDI does the following:

- Decouples the server and the client by means of well-defined types and qualifiers, so that the server implementation may vary
- Decouples the lifecycles of collaborating components by doing the following:
 - Making components contextual, with automatic lifecycle management
 - Allowing stateful components to interact like services, purely by message passing
- Completely decouples message producers from consumers, by means of events
- Decouples orthogonal concerns by means of Java EE interceptors

Along with loose coupling, CDI provides strong typing by

- Eliminating lookup using string-based names for wiring and correlations, so that the compiler will detect typing errors
- Allowing the use of declarative Java annotations to specify everything, largely eliminating the need for XML deployment descriptors, and making it easy to provide tools that introspect the code and understand the dependency structure at development time

Dependency Injection

Dependency injection (DI) is a programming design pattern that enables you to declare component dependencies. It is based on the principle of Inversion of Control (IOC) that is used to reduce dependencies between systems (loose coupling). DI enables one component to call another component or resources through interfaces. The components are glued together by using configuration, instead of code. The complexities of the service or resource instantiation, initialization, sequencing, and lookup is handled by the EJB container.

DI enables you to declare the component dependencies at design time. The EJB container reads the target EJB configuration, figures out what beans and resources the EJB requires, and injects them into the target EJB at run time. For example, you can use the @EJB annotation in a session bean to inject EJBs. Similarly, the @Resources annotation can be used to inject non-EJB resources, such as a database connection.

The implementation of DI is totally opposite to that of implementing Java Naming and Directory Interface (JNDI). In case of JNDI, it is the responsibility of the EJB to do a lookup and obtain a reference of the resources. As a result, the component and resource name are hard-coded in the EJB.

Note: Dependency injection is not limited to EJBs. You can use it in the web tier (such as in JSPs and servlets).

Beans as Injectable Objects

- The concept of injection has been part of Java technology for some time.
- Since the Java EE 5 platform was introduced, annotations have made it possible to inject resources and some other kinds of objects into container-managed objects.
- CDI makes it possible to inject more kinds of objects and to inject them into objects that are not container-managed.



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

About Beans

CDI redefines the concept of a bean beyond its use in other Java technologies, such as the JavaBeans and Enterprise JavaBeans (EJB) technologies. In CDI, a bean is a source of contextual objects that define application state and/or logic. A Java EE component is a bean if the lifecycle of its instances may be managed by the container according to the lifecycle context model defined in the CDI specification.

The following kinds of objects can be injected:

- (Almost) any Java class
- Session beans
- Java EE resources: data sources, Java Message Service topics, queues, connection factories, and the like
- Persistence contexts (JPA EntityManager objects)
- Producer fields
- Objects returned by producer methods
- Web service references
- Remote enterprise bean references

New Features: Bean Validation

- The Bean Validation specification defines a metadata model and API for validating data in JavaBeans components.
- Instead of distributing validation of data over several layers, such as the browser and the server side, you can define the validation constraints in one place and share them across the different layers.



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

Validating data is a common task that occurs throughout an application. For example, in the presentation layer of an application, you might want to validate that the number of characters a user enters in a text field is at most 20 characters or that the number a user enters in a numeric field is positive. If you set those constraints, you probably want the same data to be validated before it's used in the business logic of the application and when the data is stored in a database.

Developers often code the same validation logic in multiple layers of an application, something that's time consuming and error-prone. Or they put the validation logic in their data model, cluttering it with what is essentially metadata.

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, JSR 303 makes validation simpler and reduces the duplication, errors, and clutter that characterizes the way validation is often handled in enterprise applications. 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.

Specifically, 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.

Here, for example, is part of a class that declares some constraints through Bean Validation annotations:

```
public class Address {  
    @NotNull @Size(max=30)  
    private String addressline1;  
  
    @Size(max=30)  
    private String addressline2;  
    public String getAddressline1() {  
        return addressline1;  
    }  
    public void setAddressline1(String addressline1)  
    {this.addressline1 = addressline1;  
    }
```

New Features: Java Persistence API

- Object/Relational mapping enhancements, such as the ability to model collections of objects
- Additions to the Java Persistence query language
- A new criteria-based query API
- A new metamodel API for examining persistence units
- Support for pessimistic locking



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

Object/Relational Mapping Enhancements

JPA 1.0 supported the mapping of collections. However, those collections could only contain entities. JPA 2.0 adds the mapping of collections of basic data types, such as Strings or Integers, as well as collections of embeddable objects. Recall that an embeddable object in JPA is an object that cannot exist on its own, but exists as part of a parent object—that is, its data does not exist in its own table, but is embedded in the parent object's table.

Additions to the Java Persistence Query Language

JPA 1.0 defined an extensive Java Persistence query language (informally referred to as JPQL) with which you can query entities and their persistent state. JPA 2.0 makes a number of enhancements to JPQL. For example, you can now use case expressions in queries. JPA 2.0 also adds a number of new operators to JPQL.

Criteria API

You can use the new Criteria API to dynamically construct object-based queries. One of the significant new features introduced in JPA 2.0 is the Criteria API, an API for dynamically constructing object-based queries. In essence, the Criteria API is the object-oriented equivalent of JPQL. With it, you can take an object-based approach to creating queries, rather than using the string manipulation required by JPQL.

Support for Pessimistic Locking

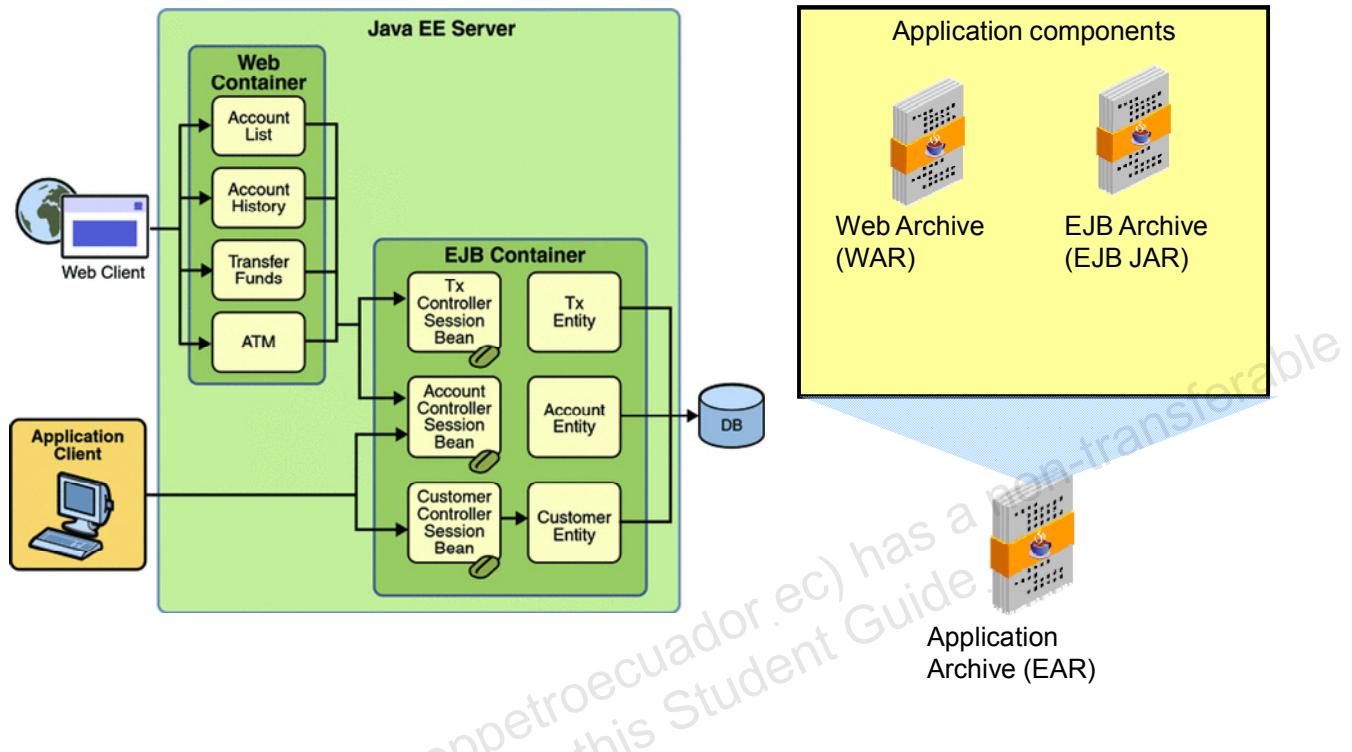
Locking is a technique for handling database transaction concurrency. When two or more database transactions concurrently access the same data, locking is used to ensure that only one transaction at a time can change the data.

JPA 1.0 only supported optimistic locking. You could indicate what type of locking was in effect by specifying a lock mode value of READ or WRITE in the lock() method of the EntityManager class.

More New Features in JPA 2.0

In addition to the enhancements and new features described in the previous sections, JPA 2.0 can use Bean Validation to automatically validate an entity when it is persisted, updated, or removed. What this means is that you can specify a constraint on an entity, for example, that the maximum length of a field in the entity is 15, and have the field automatically validated when the entity is persisted, updated, or removed. You use the <validation-mode> element in the persistence.xml configuration file to specify whether automatic lifecycle event validation is in effect.

“Classical” J2EE / Java EE 5 Architecture



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

ORACLE

As shown in the slide, “classical” J2EE and Java EE 5 architecture drew attention to the separation of components to their containers. This was due in part to the packaging requirements: ejb-jars, WARs, EAR files. Regardless of complexity or simplicity of one’s application, one had to package the components in this fashion.

In addition, extra classes (session facades, data value objects, helper beans, managed beans) were frequently needed to enable the web, business and integration tiers to interact. This has been resolved to a great degree by Java EE 6.

Impact of Java EE 6 on Architecture

- EJBs and servlets allow asynchronous calls
- Packaging requirements have been lifted, allowing components to be packed in the same WAR file (“collocated”). This simplifies the packaging, managing and deployment of Java EE 6 applications.
- CDI and the new definition of beans helps to blur or eliminate the supporting classes that made previous Java EE architectures complex.
 - For example, a JSF can now interact directly with its EJB, without needing a managed bean.
 - An EJB can use a JPA entity directly and save its state to the database without needing extra classes.
- Services can be exposed as JAX-WS or JAX-RS and implemented with POJOs or EJBs easily



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

Java EE 6 places more emphasis on the functions and use of components (servlets, JSFs, EJBs, JPA entities) and less on where they physically reside and how they are packaged. Using annotations, the application server can easily determine when a particular component must execute; what services and context must be provided for it.

EJBs can now be packed in the WAR file (“collocated”) with the servlets and JSFs, so deployment is simpler. In a real sense, Java EE 6 is more an example of taking away complexity, than adding onto it.

Quiz

To use the no-interface view feature you must use a Java EE6 implementation.

- a. True
- b. False



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

Answer: a

True—the no-interface view feature is not available before Java EE 6.

Quiz

Context Dependency Injection (CDI) provides:

- a. Integration with Expression Language
- b. Support for transactions to Web services
- c. A component model for UIs
- d. A simpler way to do lookups



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

Answer: a, b, d

Quiz

Some of the key reasons to use of the Java EE6 platform over previous versions for Enterprise Architectures includes:
(choose all that apply)

- a. Better support for the Struts framework
- b. Asynchronous calls
- c. Context Dependency Injections
- d. Reduced packaging requirements



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

Answer: b, c, d

Summary

In this lesson, you should have learned how to:

- Describe the new features in Java EE 6
- Describe new architecture choices available in Java EE 6
- Describe the impact of Java EE 6 features on J2EE and Java EE 5 architectures



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

Practice 7 Overview

This practice covers the following topics:

- How does applying the new features of Java EE 6 change your existing architecture?



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

In this practice you will consider the impact of Java EE 6 new features on your existing architecture.