Do you wish you could hear the audio and read the transcription of this session?

Advanced Core Java™ 2 Platform, Enterprise Edition (J2EE™) Patterns and Refactoring

John Crupi, Dan Malks, and Deepak Alur

Software Services
Sun Professional Services
Sun Microsystems, Inc.

Presentation Goal

Learn what's new in Core J2EE™ Patterns and how to use patterns and refactorings in building scalable multi-tier J2EE applications

Speaker's Qualifications

- John Crupi
  - Distinguished Engineer and Chief Java Architect
- Dan Malks
  - Principal Engineer
- Deepak Alur
  - Senior Java Architect
- Co-authors of *Core J2EE Patterns*

Agenda

- Introduction
- Patterns and Core J2EE Patterns
- New Core J2EE Patterns
  - Presentation Tier—Dan
  - Business Tier—Deepak
  - Integration Tier—John
- Summary
- Q&A

Introduction: Field Observations…

- Tremendous value in design "reuse"
- Patterns seem to be "sweet spot" of design
- Developers desire:
  - Pattern abstractions
  - Code realizations
- Bad Practices as valuable as good practices
- Refactorings provide clear way to "get to" a pattern
    - *Patterns help put all this together!*

Patterns Everywhere!

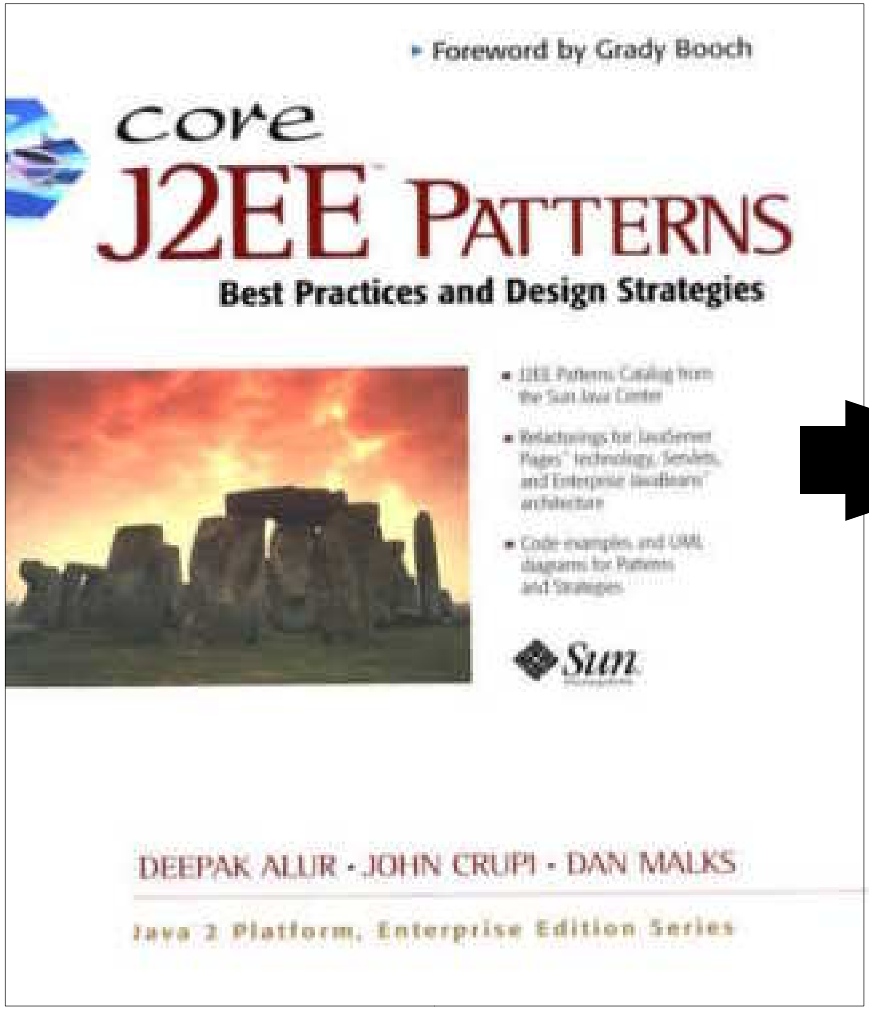
What Are Patterns?

- Patterns communicate:

  "Solution to a recurring Problem in a Context"

- A design which is used by others
- An abstraction which can be realized
- Discovered, proven expert solutions
- Creates a higher level vocabulary
- Combined to solver bigger problem

Core J2EE Patterns: 2nd Edition—Hot Off the Press!



▸ Foreword by Grady Booch

core J2EE™ Patterns

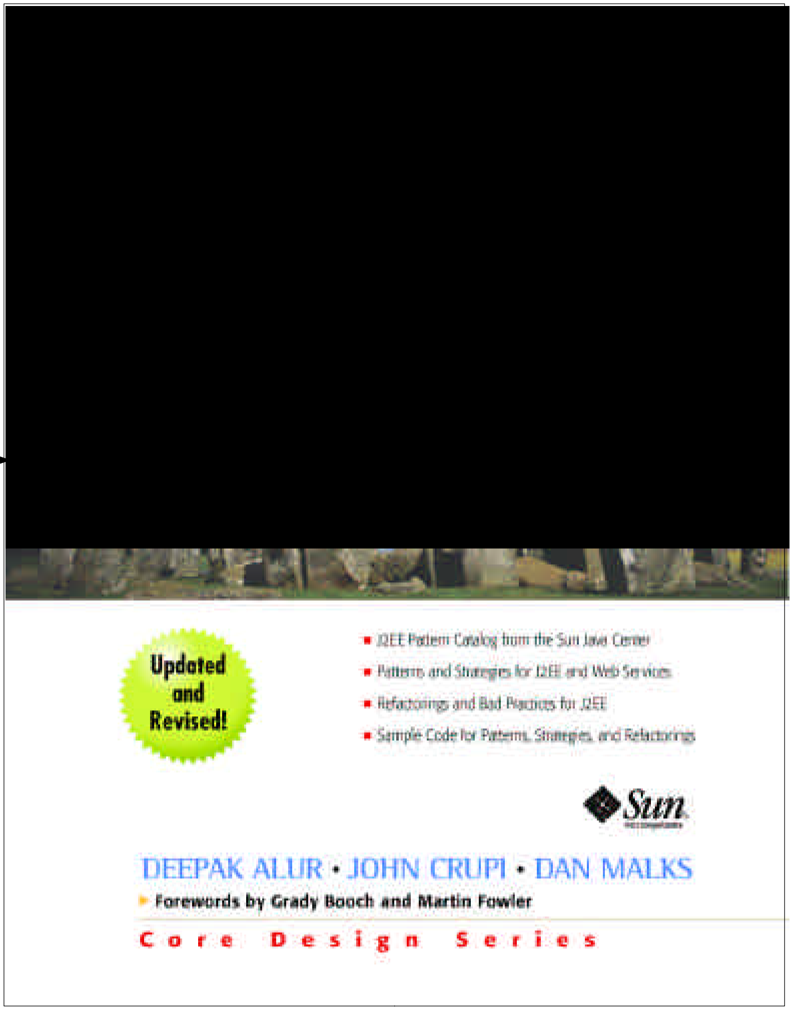
Best Practices and Design Strategies

- J2EE Patterns Catalog from the Sun Java Center
- Refactorings for JavaServer Pages™ technology, Servlets, and Enterprise JavaBeans™ architecture
- Code examples and UML diagrams for Patterns and Strategies

Sun

DEEPAK ALUR • JOHN CRUPI • DAN MALKS

Java 2 Platform, Enterprise Edition Series



Updated and Revised!

- J2EE Pattern Catalog from the Sun Java Center
- Patterns and Strategies for J2EE and Web Services
- Refactorings and Bad Practices for J2EE
- Sample Code for Patterns, Strategies, and Refactorings

Sun

DEEPAK ALUR • JOHN CRUPI • DAN MALKS

▸ Forewords by Grady Booch and Martin Fowler

Core Design Series

What's New?

- 15 original patterns fully revised
- 6 new patterns
- New coverage on Web Services
- New and Improved sample code
- Completely overhauled UML diagrams
- All patterns show inbound and outbound patterns
- Introducing: *Micro-architecture—Web Worker*

J2EE Pattern Catalog

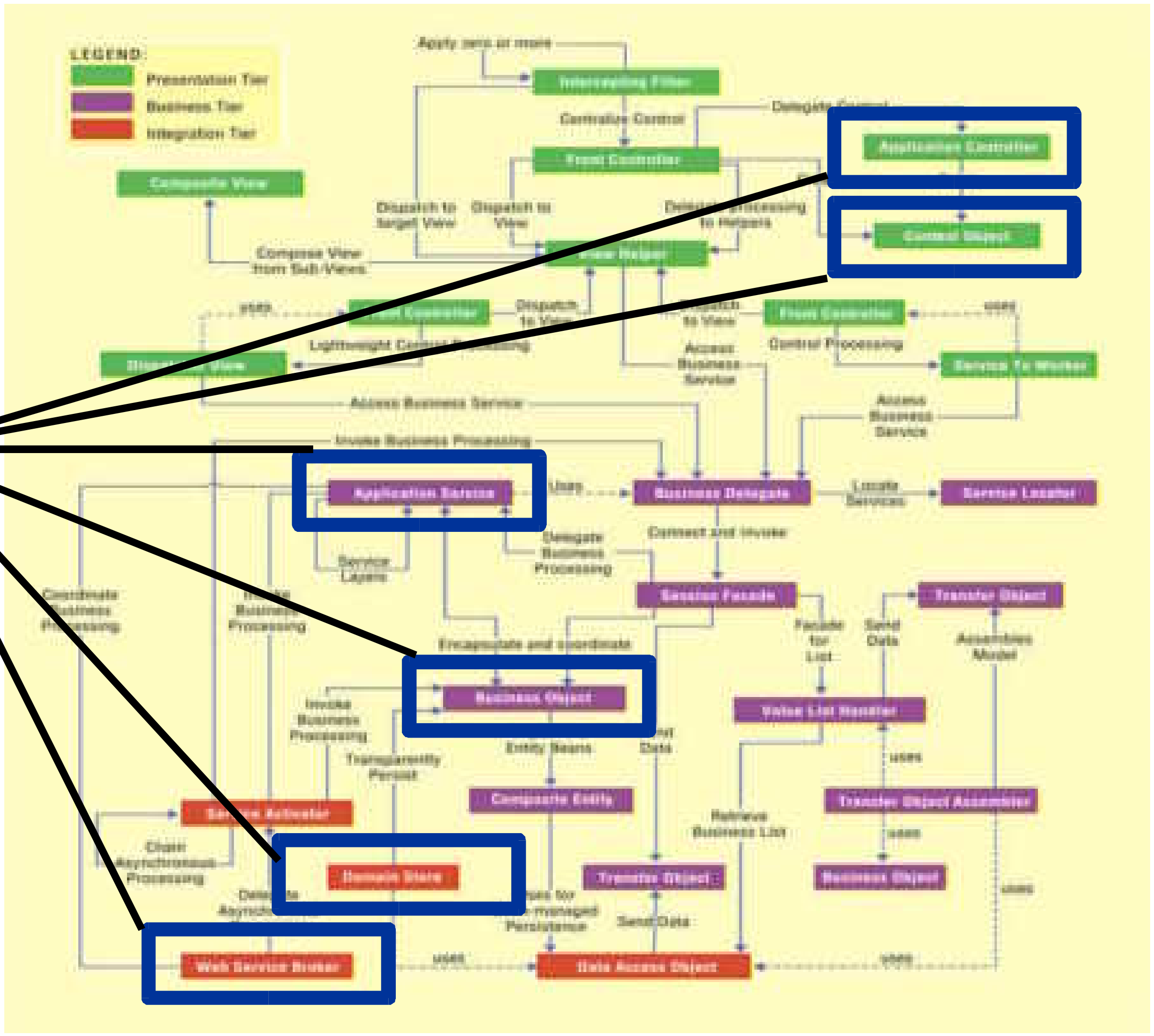
Presentation Tier
- Intercepting Filter
- Front Controller
- **Context Object**
- **Application Controller**
- View Helper
- Composite View
- Service To Worker
- Dispatcher View

Business Tier
- Business Delegate
- Service Locator
- Session Facade
- **Application Service**
- **Business Object**
- Composite Entity
- Transfer Object
- Transfer Object Assembler
- Value List Handler

Integration Tier
- Data Access Object
- Service Activator
- **Domain Store**
- **Web Service Broker**

Pattern Relationships

New Patterns



Agenda

- ✓ Introduction
- ✓ Patterns and Core J2EE Patterns
- New Stuff
  - → Presentation Tier—Dan
  - Business Tier—Deepak
  - Integration Tier—John
- Summary
- Q&A

Presentation Tier Patterns

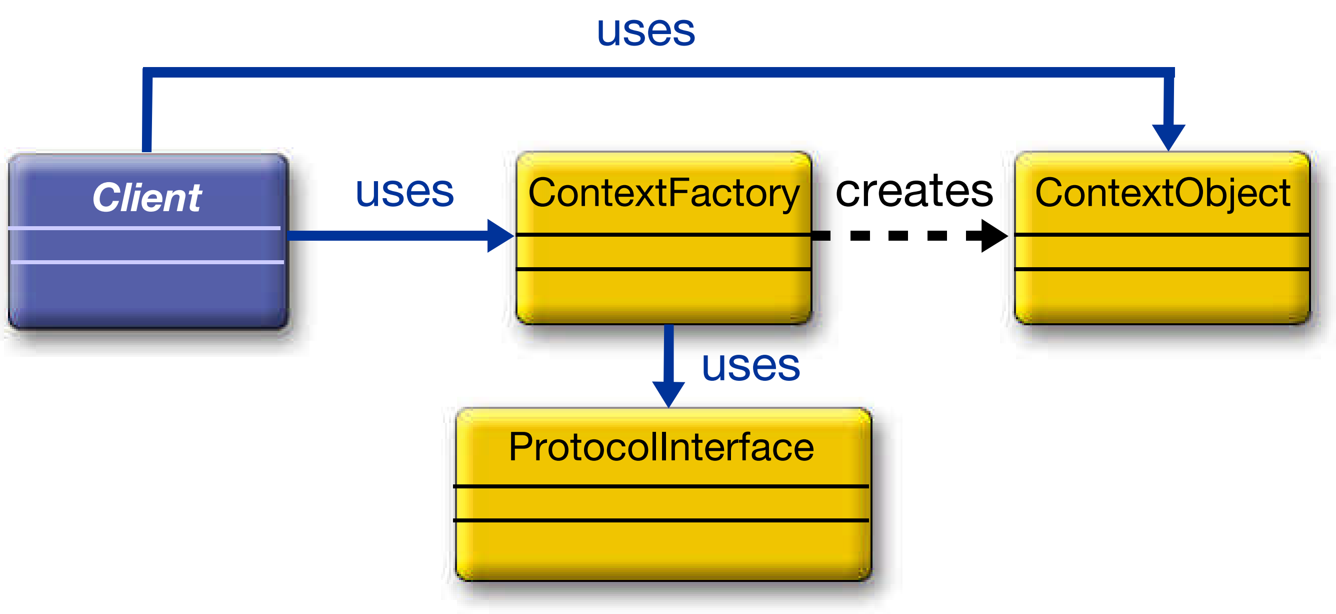
- Intercepting Filter
- Front Controller
- **Context Object**
- **Application Controller**
- View Helper
- Composite View
- Service To Worker
- Dispatcher View

Context Object

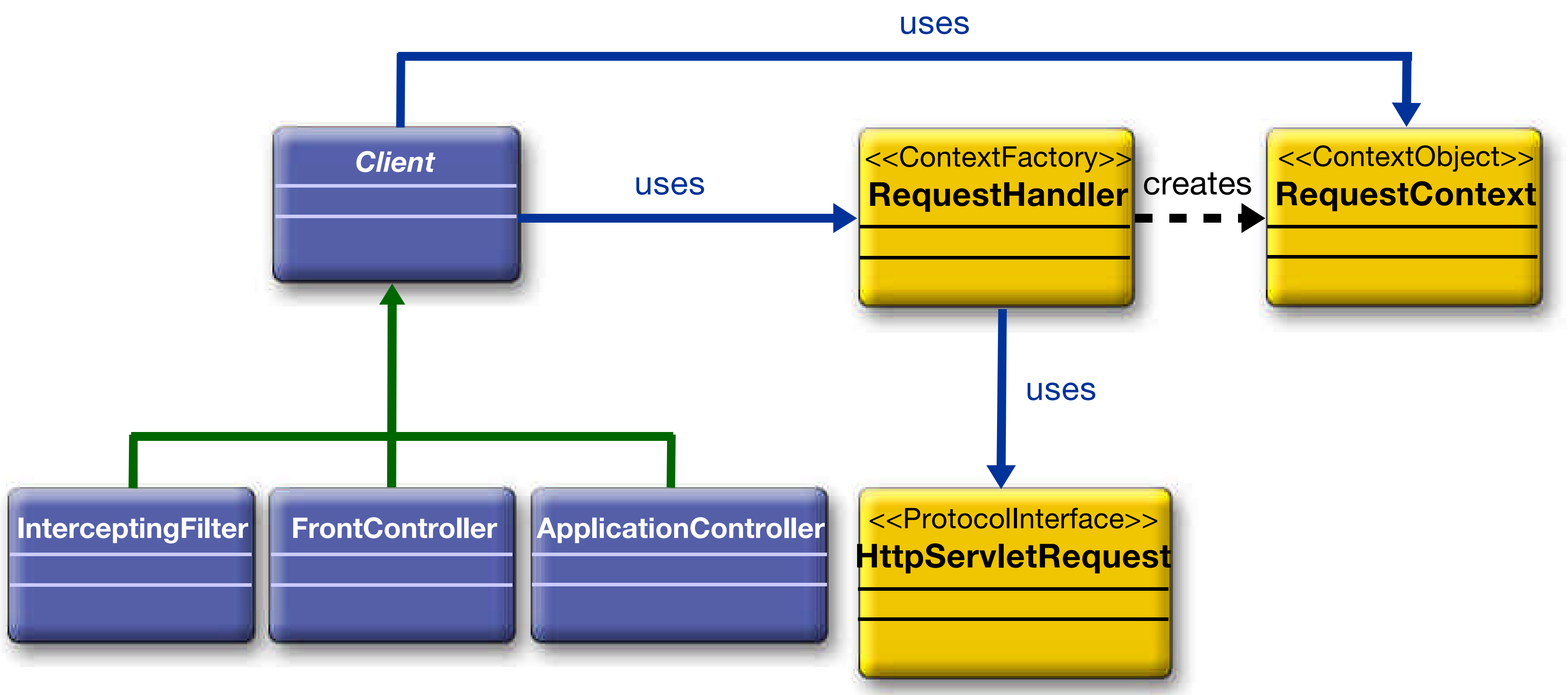
- Problem:
  - Avoid using protocol-specific system information outside of its relevant context
- Forces:
  - Components and services need access to system information
  - Decouple application components and services from the protocol specifics of system information
  - Expose only the relevant APIs within a context

Context Object

- Solution:
  - Use a Context Object to encapsulate state in a protocol-independent way to be shared throughout your application



Context Object: Request Context Strategy



Context Object: Strategies

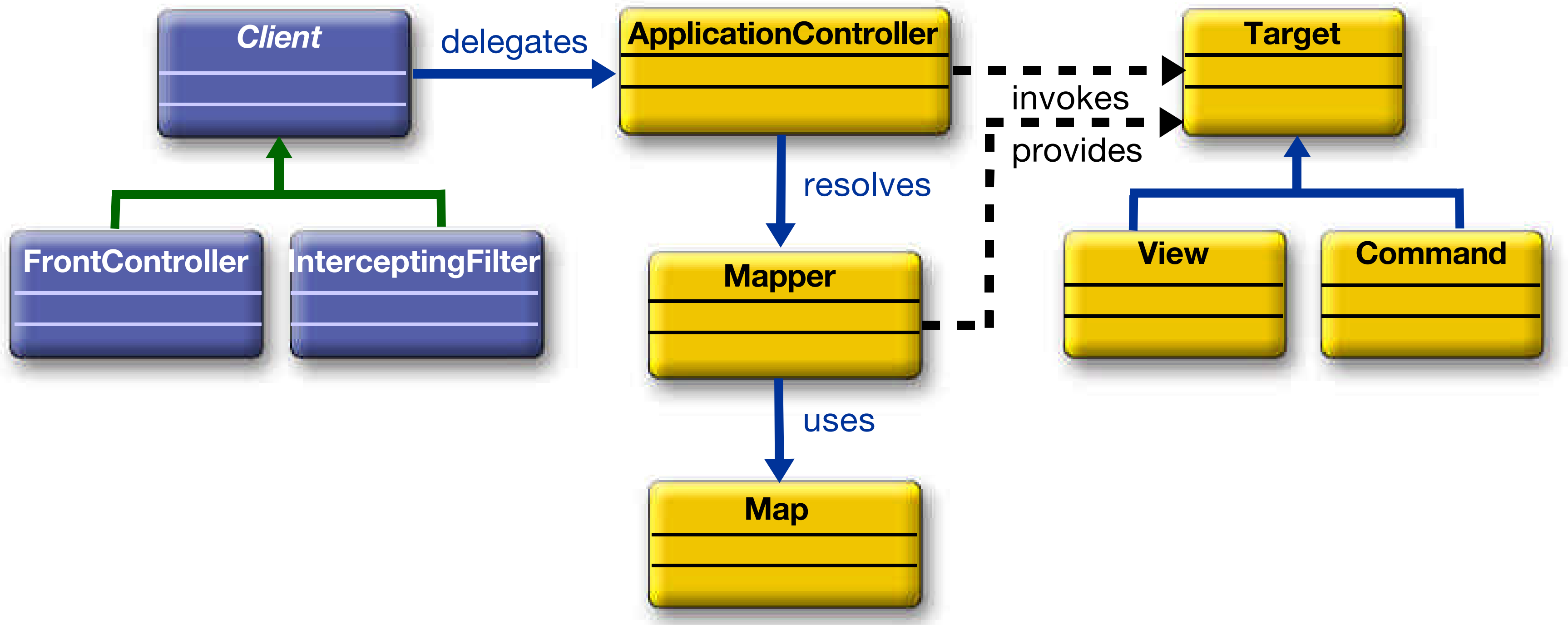
- Request Context Strategies
  - Request Context Map Strategy
  - Request Context POJO Strategy
  - Request Context Validation Strategy
- Configuration Context Strategies
  - JSTL Configuration Strategy
- Security Context Strategy
- General Context Object Strategies
  - Context Object Factory Strategy
  - Context Object Auto-population Strategy

Application Controller

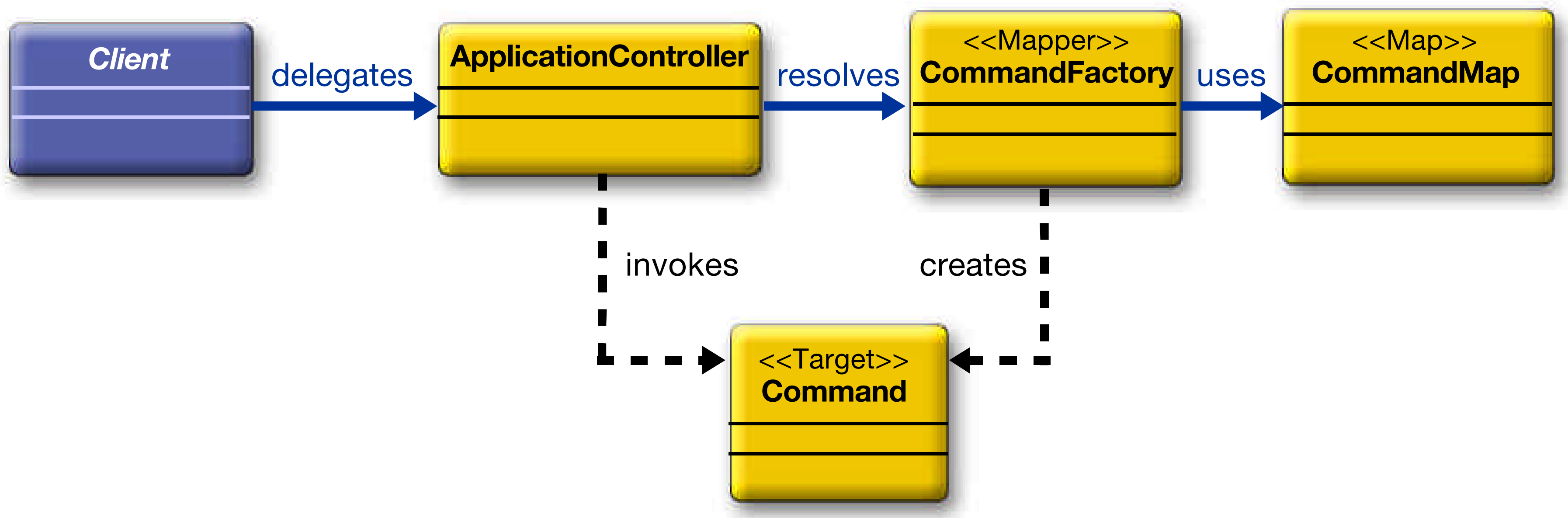
- Problem:
  - Centralize and modularize action and view management
- Forces:
  - Reuse action-management and view-management code
  - Improve code modularity and maintainability

Application Controller

- Solution:
  - Use an Application Controller to centralize retrieval and invocation of request-processing components



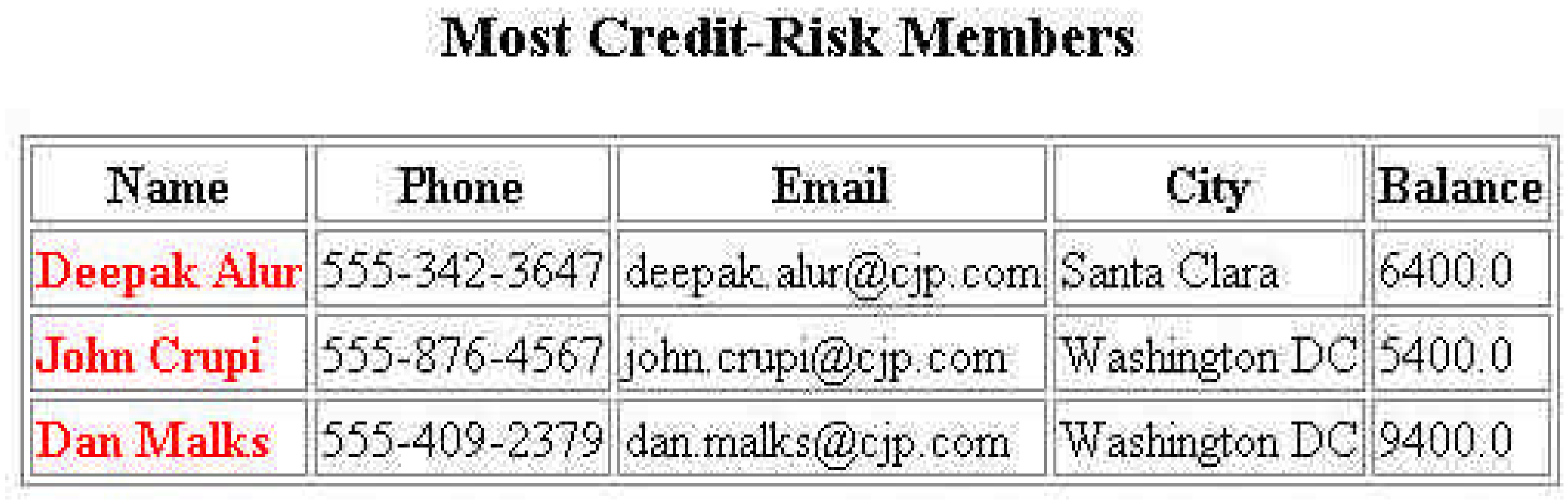
Application Controller: Command Handler Strategy



Bad Practice: Using Helpers as Scriptlets

- Problem Summary:
  - Helpers are introduced but continue to expose the same implementation details of the original scriptlet code
- Solution:
  - Use JSTL Helpers
  - Use Custom Tags
  - Use Tag Files

Bad Practice: Using Helpers as Scriptlets



Most Credit-Risk Members

Name	Phone	Email	City	Balance
Deepak Alur	555-342-3647	deepak.alur@cjp.com	Santa Clara	6400.0
John Crupi	555-876-4567	john.crupi@cjp.com	Washington DC	5400.0
Dan Malks	555-409-2379	dan.malks@cjp.com	Washington DC	9400.0

Problem: Scriptlets in View

```
<table border="1" >
<tr><th> Name </th> <th> Phone </th>
. . .
</tr>
<%
Iterator members = memberlist.listIterator();
int maxrows = 0;
while ( members.hasNext()) {
   MemberTO member = (MemberTO)members.next();
   // Biz Rule: Do not include Platinum members in this report
   if ( ( member.getPrivilege().equalsIgnoreCase("Gold") ||
           member.getPrivilege().equalsIgnoreCase("Silver")) &&
         (member.getCreditBalance() > 5000) && (maxrows < 10)) {
%>
   <tr>
   <td><%= member.getName()%></td>
   <td><%= member.getPhone()%></td>
. . .
   </tr>
<%
   maxrows++;
   }
}
%>
</table>
```

Solution: Use Helpers / JSTL

```
<table border="1" >
<tr>
<th> Name </th>
<th> Phone </th>
...
</tr>
<c:set value="0" var="RowCount"/>
<c:forEach var="member" items="${memberlist}">
<c:if test="${(member.privilege == 'Gold' or
                        member.privilege=='Silver')
  and (member.creditBalance > 5000) and (RowCount < 10 )}
">
<tr>
<td><B><c:out value="${member.name}"/></B></td>
<td><c:out value="${member.phone}"/></td>
...
</tr>
<c:set value="${RowCount+1}" var="RowCount"/>
</c:if>
```

Refactor: Use Custom Tag

```
<%@ taglib uri='/WEB-INF/corej2eetaglibrary.tld' prefix='cjp' %>

<html>
<head><title>Member Ratings</title></head>
<body>

<jsp:useBean id="memberlist" type="java.util.List"
scope="request"/>

<div align="center">
<h3>Most Credit-Risk Members</h3>
<cjp:buildMemberTable privilege="Gold,Silver" credit="5000"
rows="10"/>
</div>
</body>
</html>
```

Refactor: Use Tag File

CreditRiskMembers.jsp

```
. . .
<tags:bldmemberTable privilege="Gold" credit="5000" rows="10">
<jsp:attribute name="nameStyle">
<font color=”red”><B>${name}</B></font>
</jsp:attribute>
</tags:memberList>
```

CreditRiskMembers.tag

```
. . .
<%@ attribute name="nameStyle" fragment="true" %>
. . .
<c:forEach var="member" items="${memberlist}">
<c:if test="${(member.privilege == privilege)
and (member.creditBalance > credit) and (RowCount < rows )}">
<tr>
   <td>
      <c:set var="name" value="${member.name}"/>
      <jsp:invoke fragment="nameStyle"/>
   </td>
```

Agenda

- ✓ Introduction
- ✓ Patterns and Core J2EE Patterns
- New Stuff
  - ✓ Presentation Tier—Dan
  - → Business Tier—Deepak
  - Integration Tier—John
- Summary
- Q&A

Business Tier

- New Patterns:
  - Application Service
  - Business Object
- Design Considerations:
  - Service facade and Entity facade
- Refactoring Scenarios

Business Tier Patterns

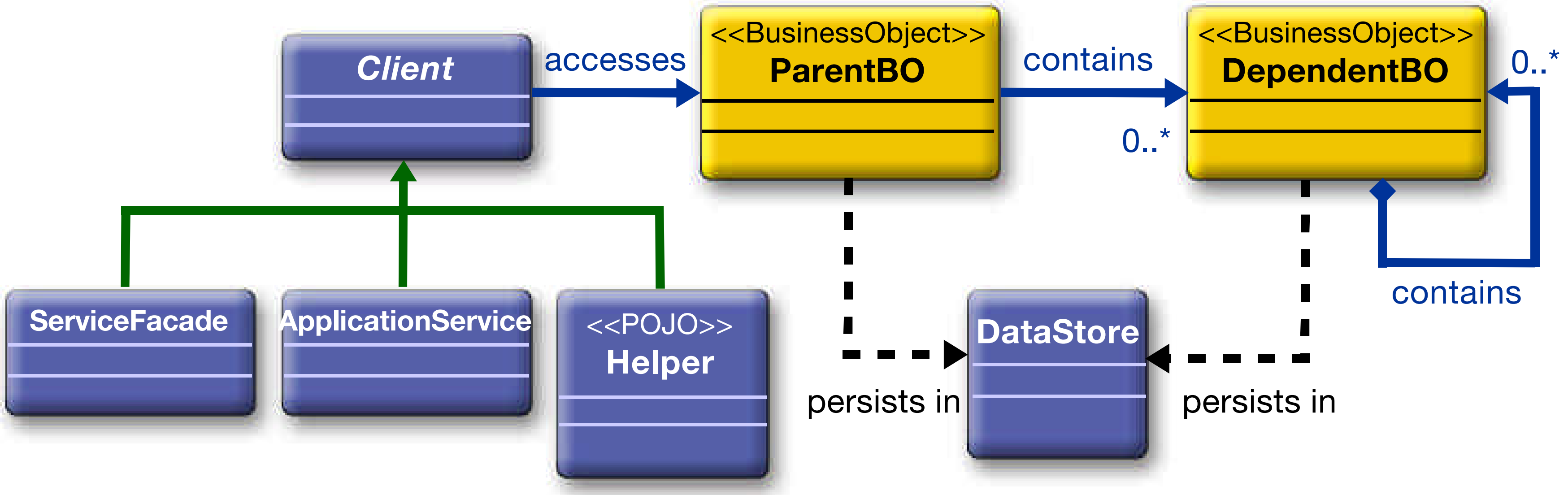
- Business Delegate
- Service Locator
- Session Facade
- **Business Object**
- **Application Service**
- Composite Entity
- Transfer Object
- Transfer Object Assembler
- Value List Handler

Business Object

- Problem:
  - Conceptual domain model with business logic and relationships
- Forces:
  - Structured, interrelated composite objects
  - Complex business logic, validation, and rules
  - Centralize business logic and state
  - Increase reusability of business logic and avoid duplication of code

Business Object

- Solution:
  - Use Business Objects to separate business data and logic using an object model

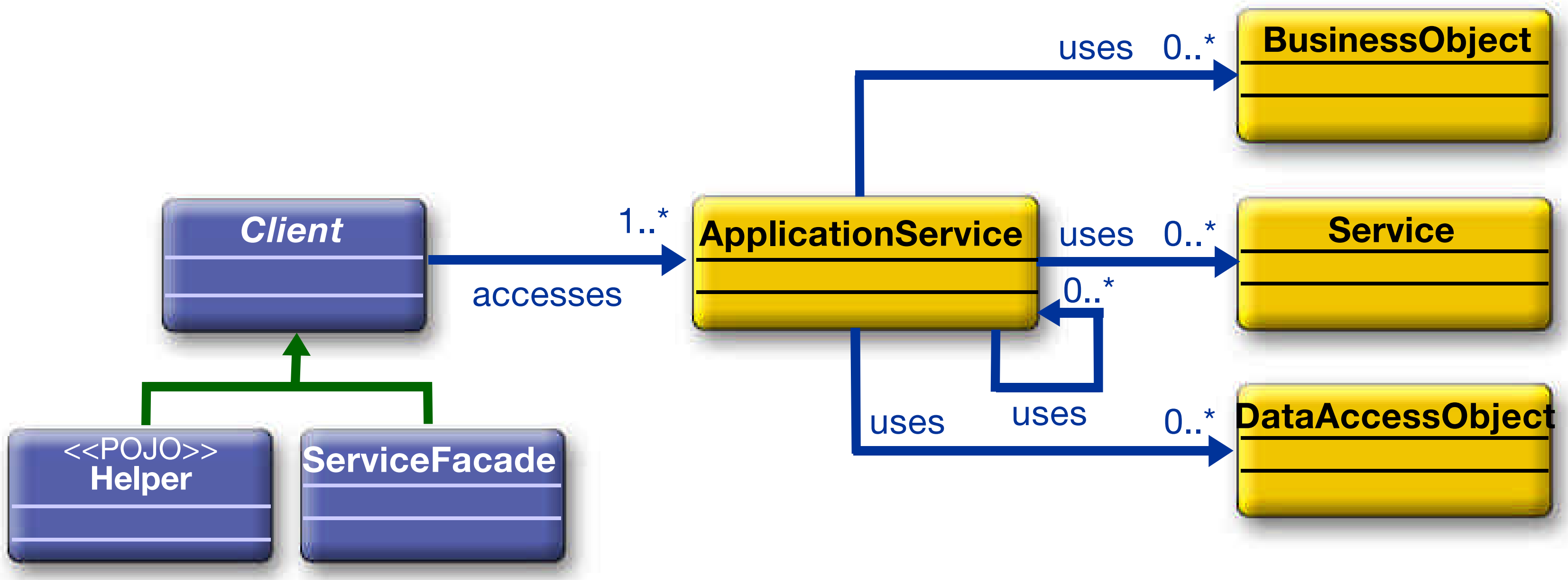


Application Service

- Problem:
  - Centralize business logic across several business-tier components and services
- Forces:
  - Minimize business logic in service facades
  - Business logic acting on multiple Business Objects or services
  - Provide a coarser-grained service API
  - Encapsulate use case-specific logic outside individual Business Objects

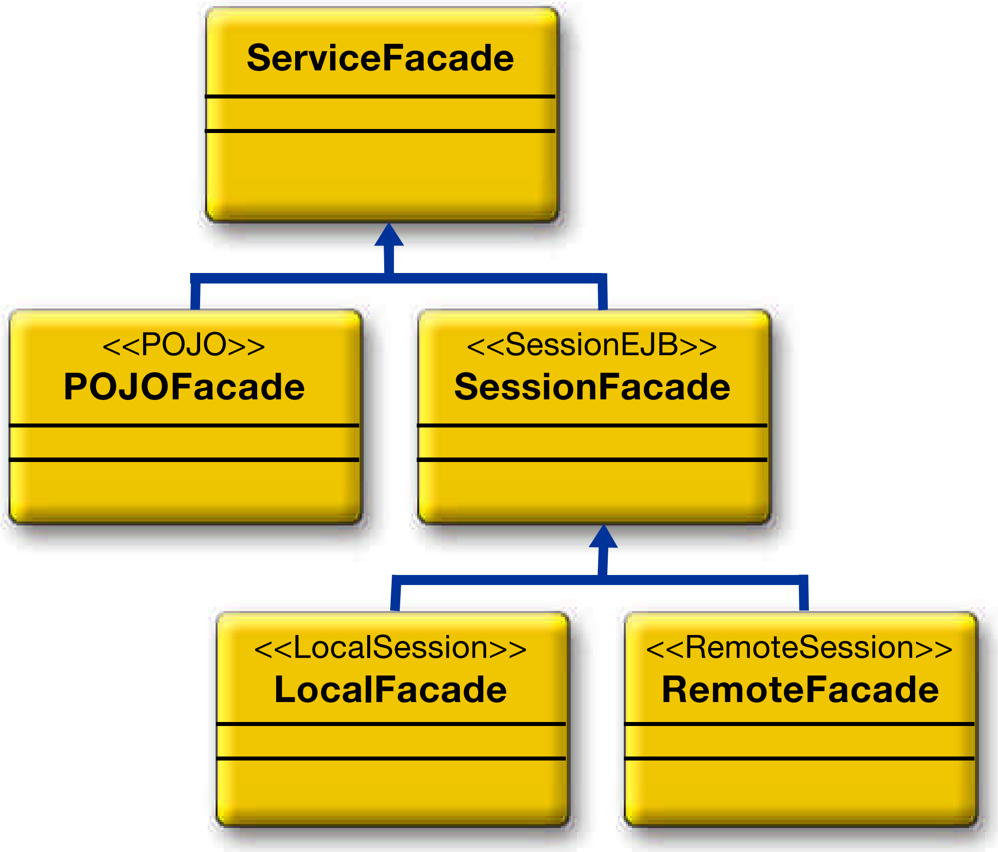
Application Service

- Solution:
  - Use an Application Service to centralize and aggregate behavior to provide a uniform service layer



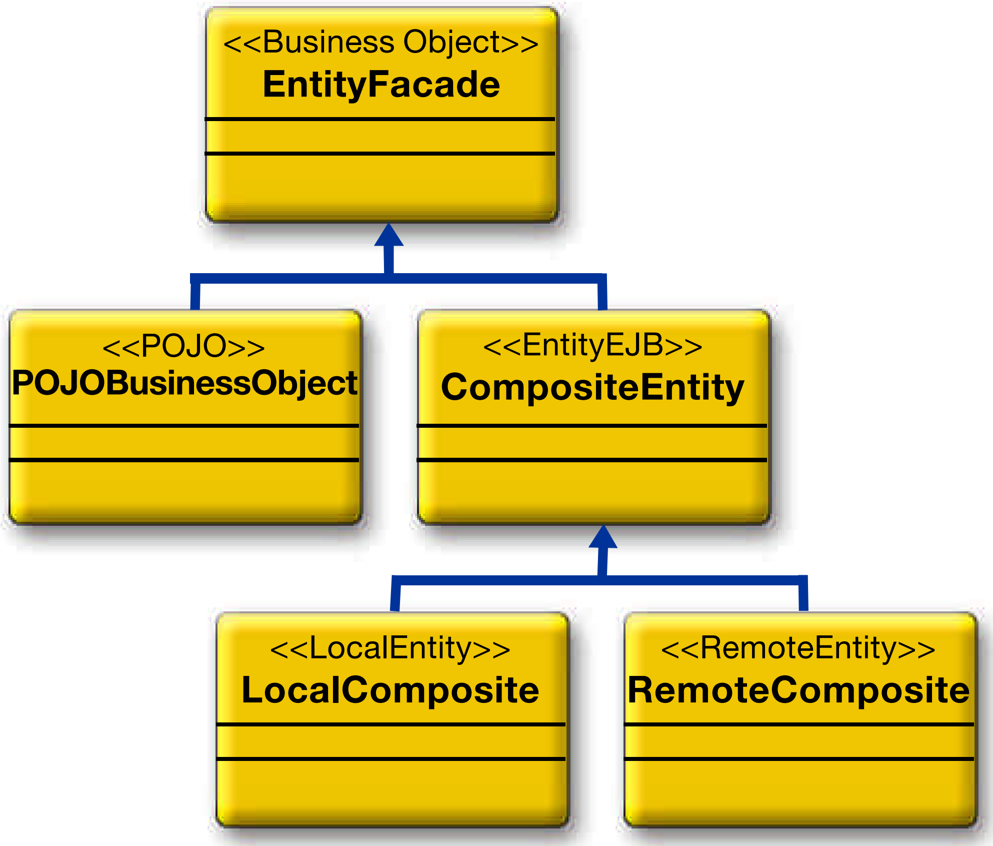
Design Idea: Service Facades

- Remote and non-Remote business tier



Design Idea: Entity Facades

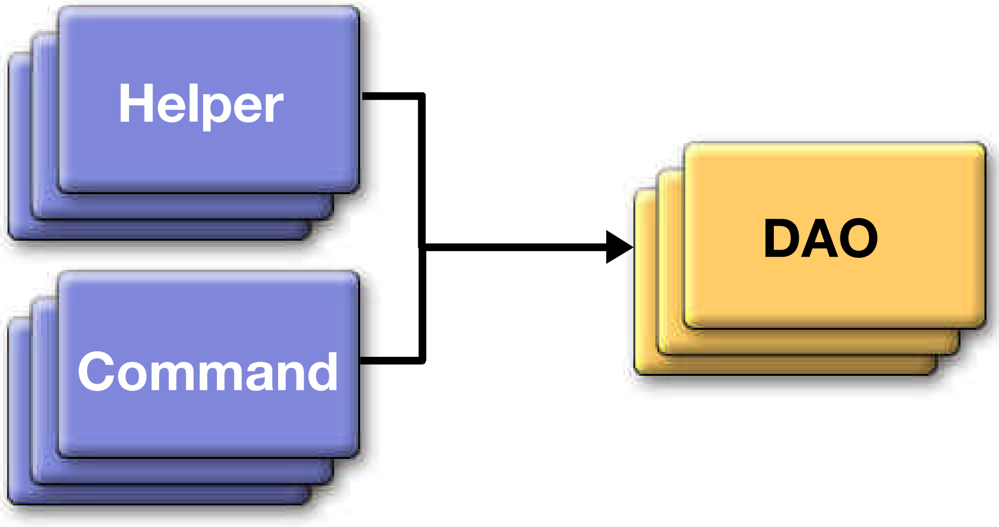
- Remote and non-Remote business tier



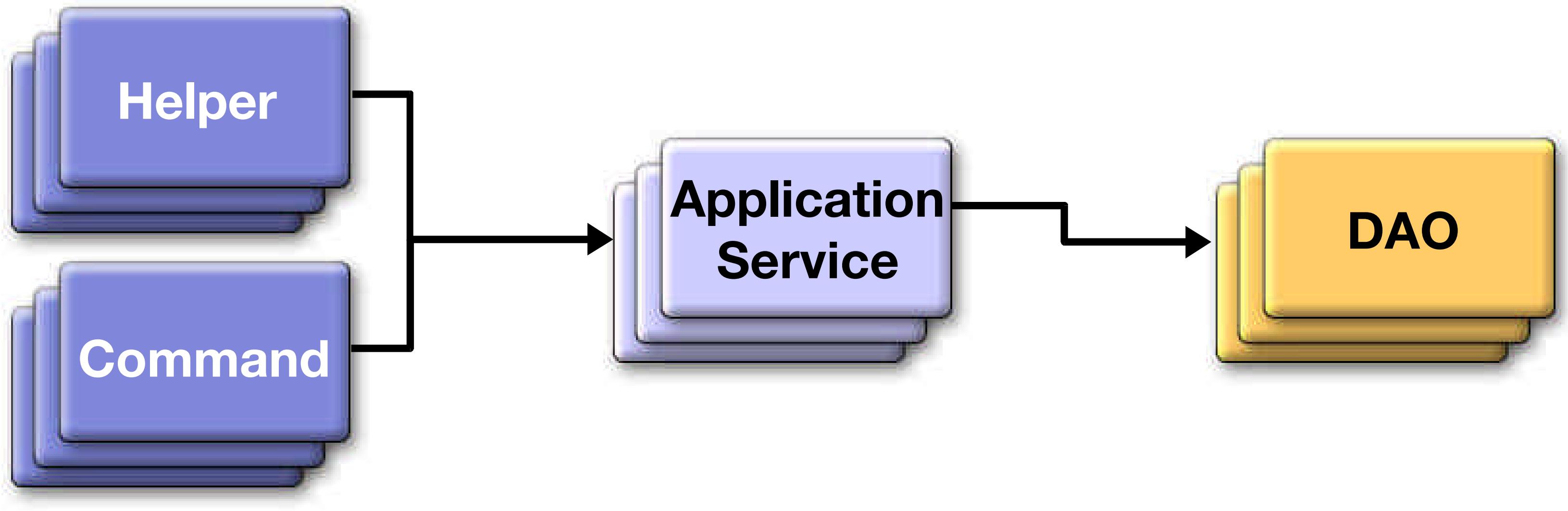
Refactoring Scenarios

- Direct Access
- Use Application Service
- Encapsulate With Session Facade
- Introduce Application Service
- Use Business Objects

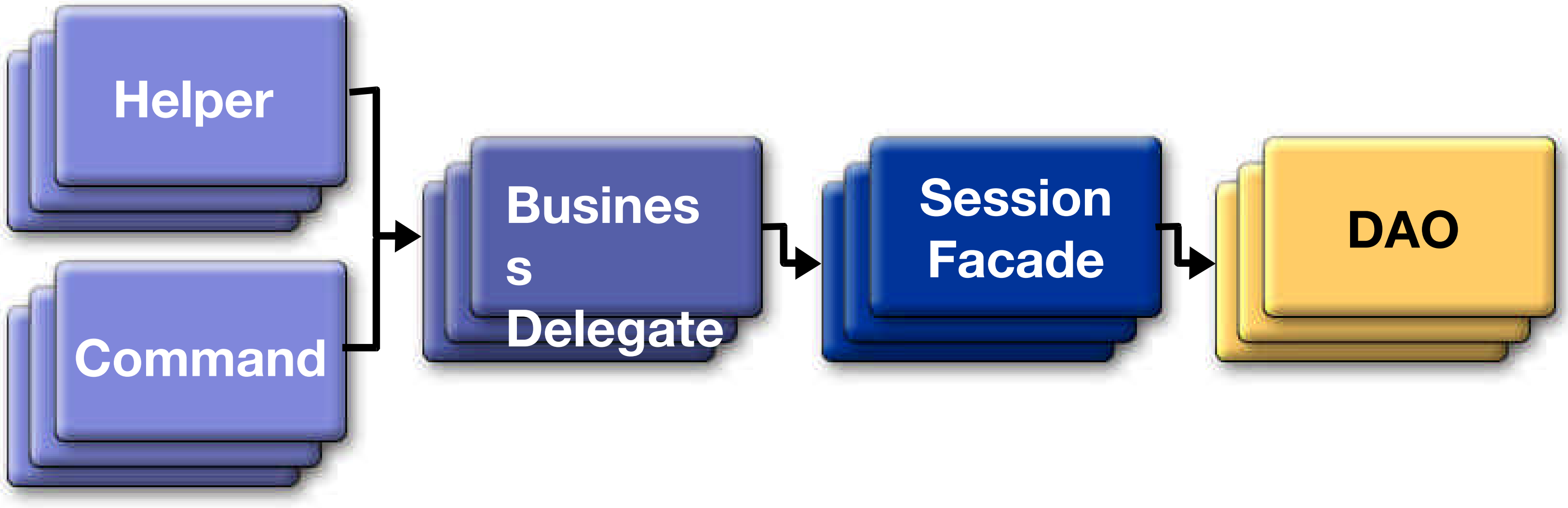
Direct Access



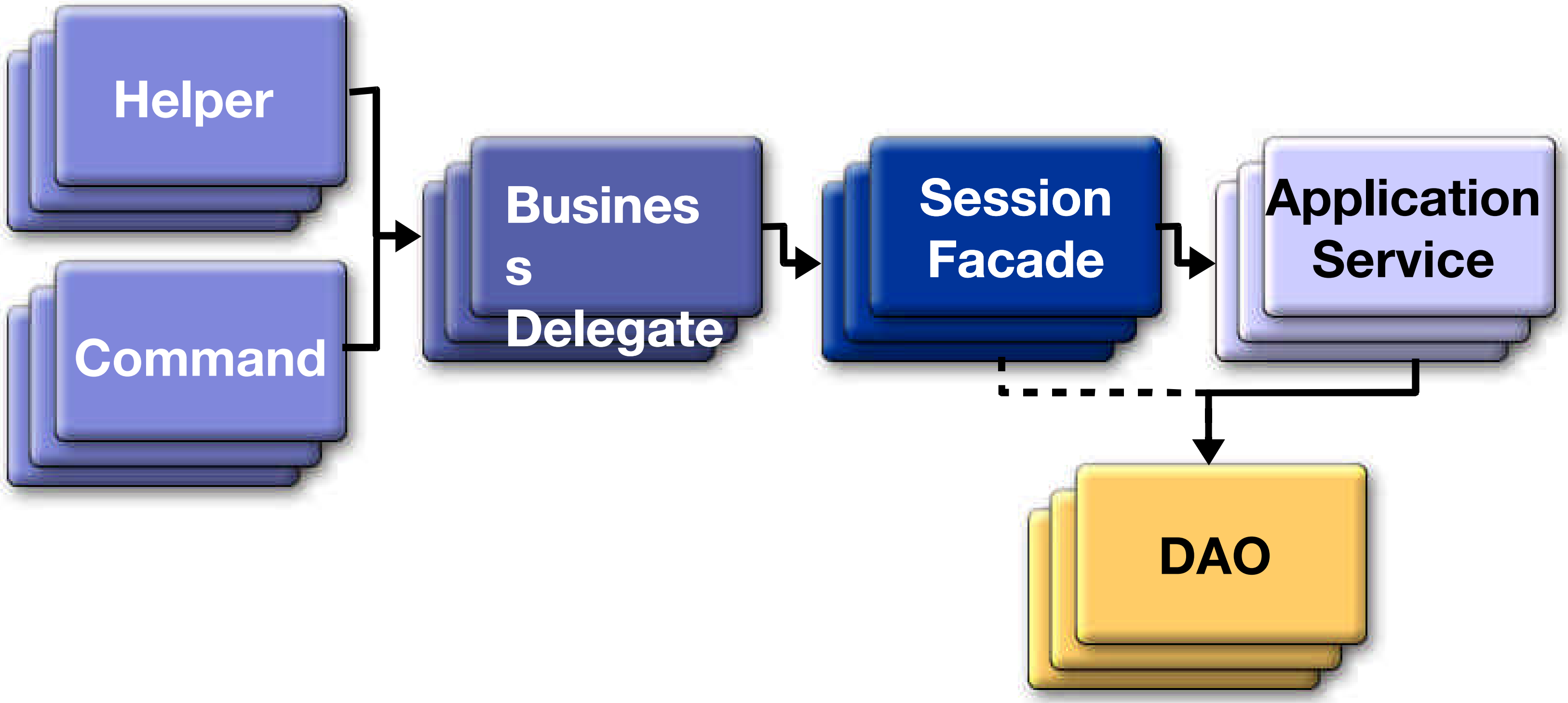
Use Application Service



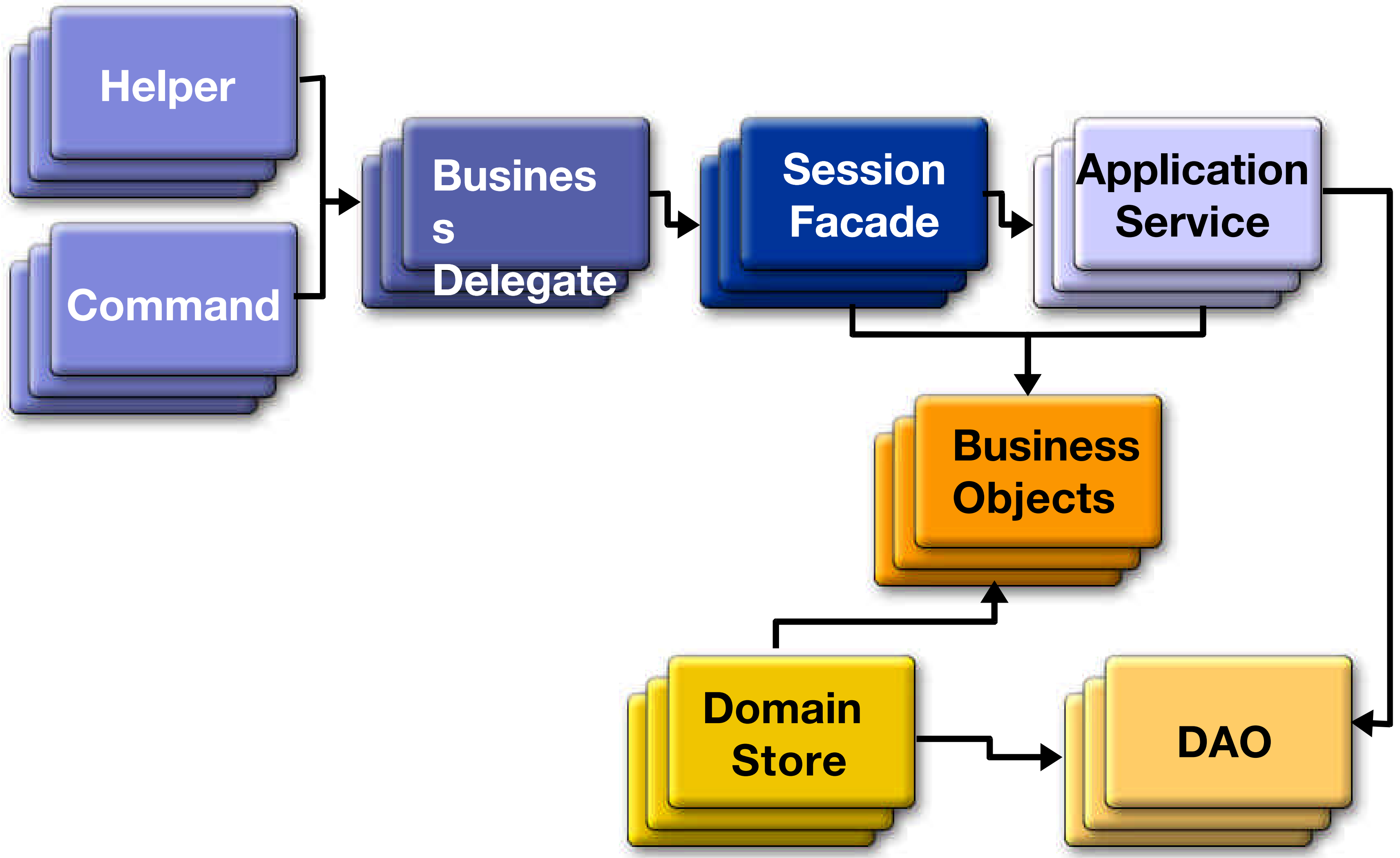
Encapsulate With Session Facade



Introduce Application Service



Use Business Objects



Agenda

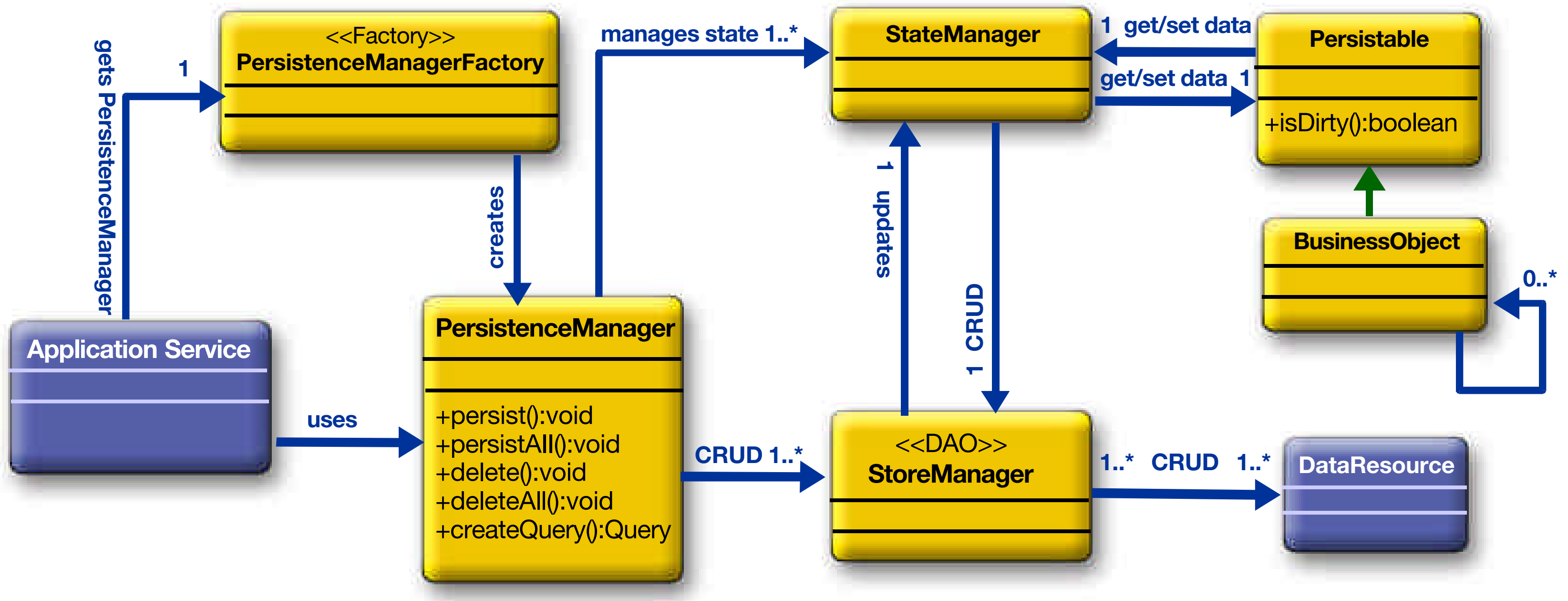
- ✓ Introduction
- ✓ Patterns and Core J2EE Patterns
- New Stuff
  - ✓ Presentation Tier—Dan
  - ✓ Business Tier—Deepak
  - → Integration Tier—John
- Summary
- Q&A

Integration Tier Patterns

- Data Access Object
- Service Activator
- **Domain Store**
- **Web Service Broker**

Domain Store

- Problem:
  - Separate persistence from your object model

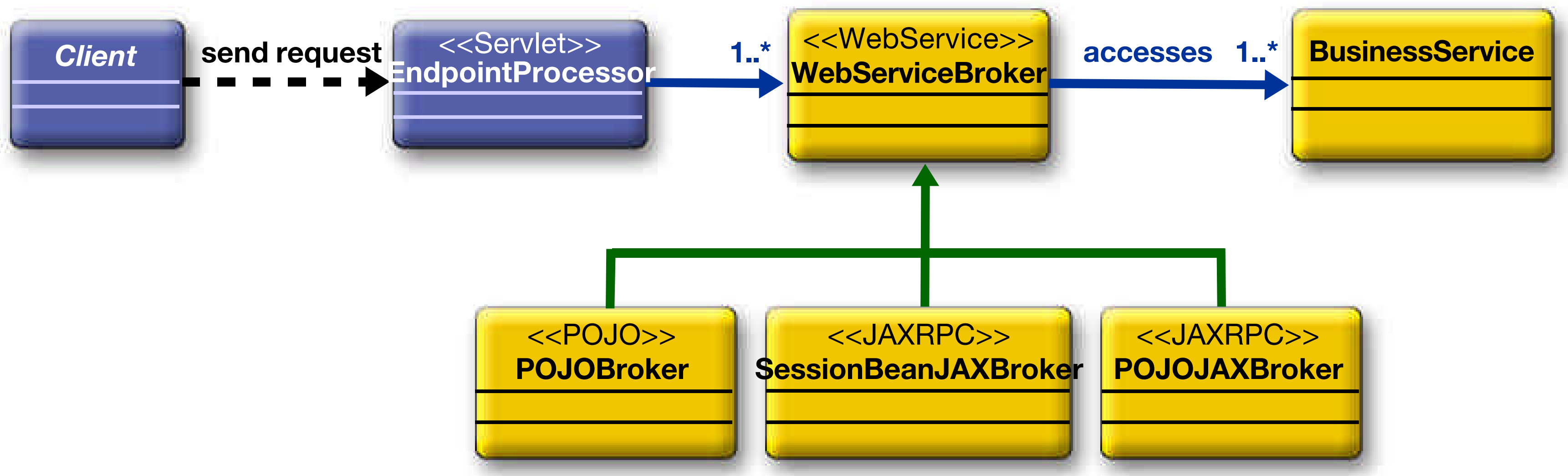


Web Service Broker

- Problem:
  - Provide access to one or more services using XML and web protocols
- Forces:
  - Reuse and expose existing services to clients
  - Monitor and potentially limit the usage of exposed services
  - Expose services using open standards

Web Service Broker

- Solution:
  - Use a Web Service Broker to expose and broker one or more services using XML and web protocols



Web Service Broker: Strategies

- Custom XML Messaging Strategy
- Java™ Technology Binding Strategy
- JAX-RPC Strategy

Support/Adoption

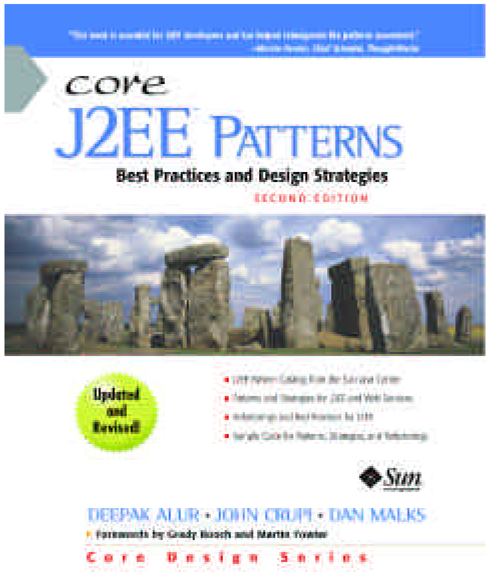
- Developer Community
- 3rd Party
  - Logic Library
  - Object Venture
  - TogetherSoft
  - Rational
  - Oakgrove Systems (Visit their Booth)
- Sun
  - Sun™ ONE Studio (5.x)
  - J2EE BluePrints/Java Pet Store/Adv Builder
  - Sun Education Courses (SL-500, SL-425)

Summary

- Patterns are great! Use them
- Reduce re-inventing the wheel
- Promote design re-use
- Increase developer productivity, communication
- Large community around patterns
- ISV support growing

Stay Connected

- Check out CJP:
  - Book signing—Bookstore at 3:00pm
  - http://java.sun.com/blueprints/corej2eepatterns
- Subscribe:
  - http://archives.java.sun/j2eepatterns-interest.html
- Write to us:
  - j2eepatterns-feedback@sun.com
- Java.Net
  - patterns.java.net


core
J2EE PATTERNS
Best Practices and Design Strategies
SECOND EDITION
Updated and Revised!
DEEPAK ALUR • JOHN CRUPI • DAN MALKS
Forewords by Grady Booch and Martin Fowler
Core Design Series

Q&A

JavaOne℠

Sun's 2003 Worldwide Java Developer Conference

Java™

java.sun.com/javaone/sf