



Core J2EE Patterns, Frameworks and Micro Architectures

Deepak.Alur@sun.com

**Patterns & Design Expertise Center
Sun Software Services**

January 2004



Agenda

- Patterns
- Core J2EE Pattern Catalog Background
- J2EE Progressive Refactoring
- Pattern Frameworks
- Micro Architecture
 - Web Worker Micro Architecture Example
 - Messaging Micro Architecture Example
- Q&A

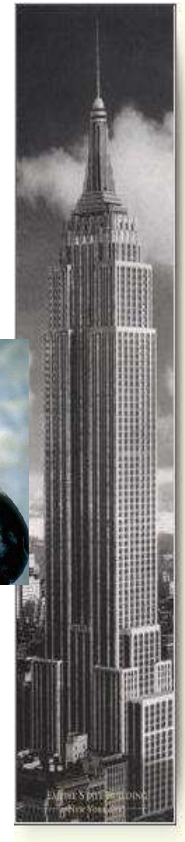
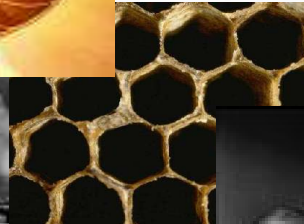
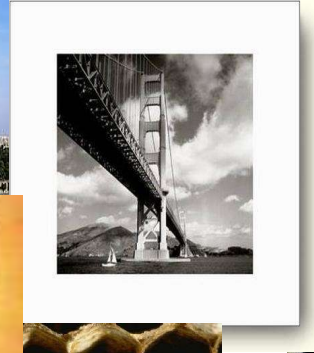
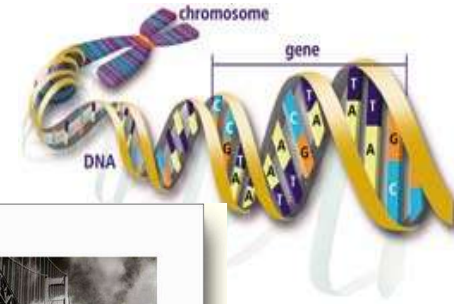
Architectural Decisions Produce Varying Results



.Net



J2EE



- [illegible]

Patterns are...

- Abstractions
- Discovered, not created
- Difficult to see the appropriate granularity
- Mined from good designs
- Refactoring targets

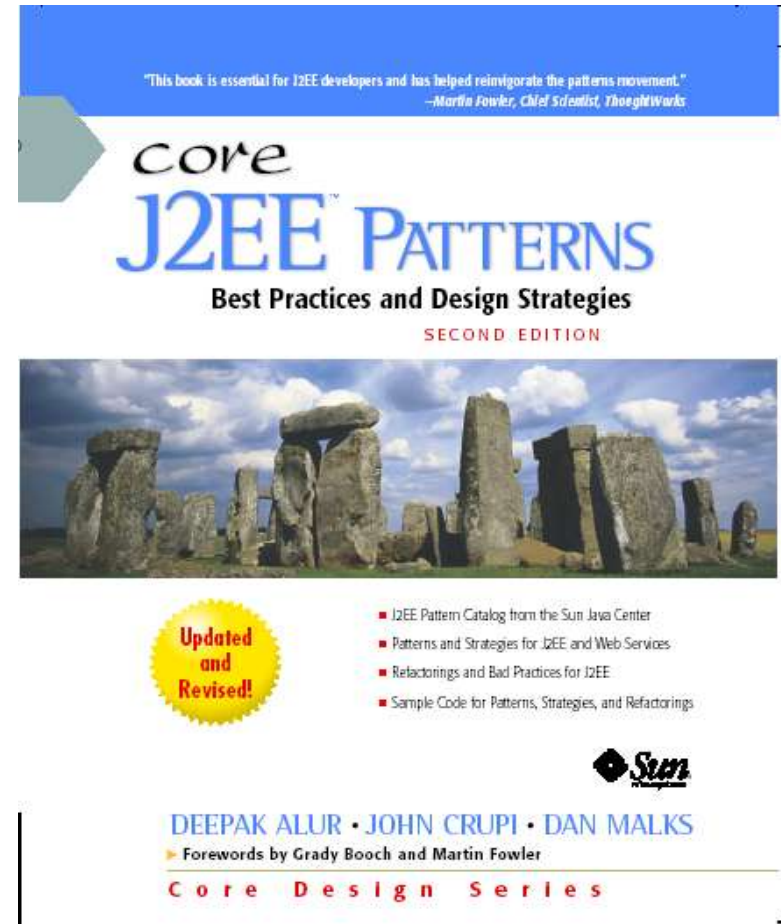
Core J2EE Patterns

- Core J2EE Patterns are platform patterns.
 - The context is bounded by the J2EE platform
 - Built upon non-platform patterns – GoF



Core J2EE Patterns Book

- 1st Edition June 2001
 - 15 Patterns categorized by tiers:
 - Presentation
 - Business
 - Integration
 - Lots of Code Samples
 - Design Considerations
 - Bad Practices
 - Refactorings
-
- 2nd Edition JavaOne, June 2003
 - 21 patterns
 - Micro-architecture



Core J2EE Patterns Book



J2EE Pattern Catalog
Addresses 3 Tiers

Pattern Format

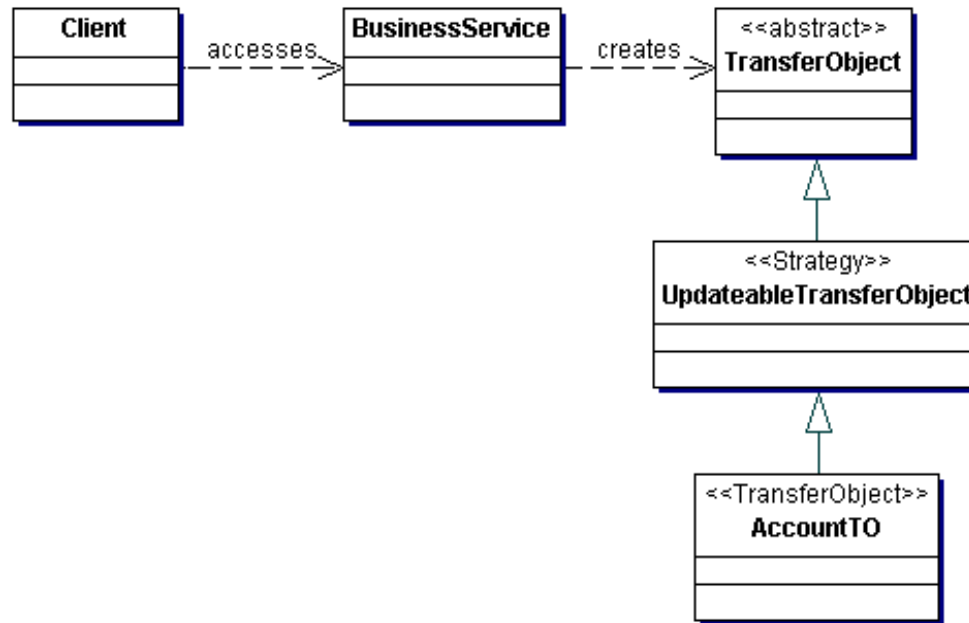
- Problem
- Forces
- Solution
 - Structure
 - Interaction
- Consequences
- Strategies



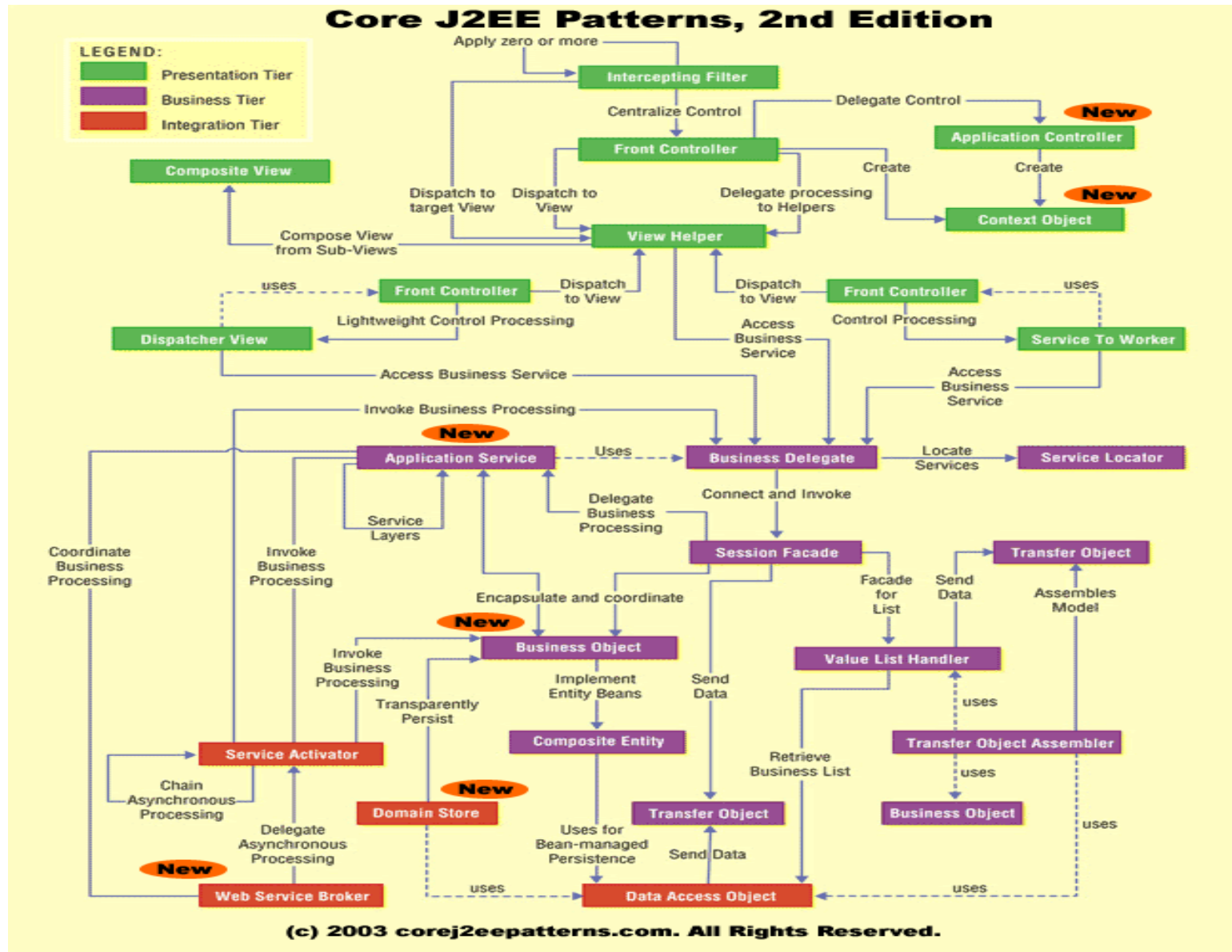
Extensibility

Pattern Strategies

- Pattern is abstract and a strategy is (more) concrete



Pattern Relationships



Presentation-Tier Patterns

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

Business Tier Patterns

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

Integration Patterns

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

New Patterns Facts

- Patterns represent abstractions emerging from using existing patterns in complex applications and flesh out pattern language (Context Object, Application Controller, Business Object, etc.).
- New patterns rely on POJO stereotype
- New patterns identify a “web container only” scenario
- Domain Store addresses Transparent Persistence (JDO and the like)
- Updated for J2EE 1.4 and Web Services

Presentation Tier Patterns

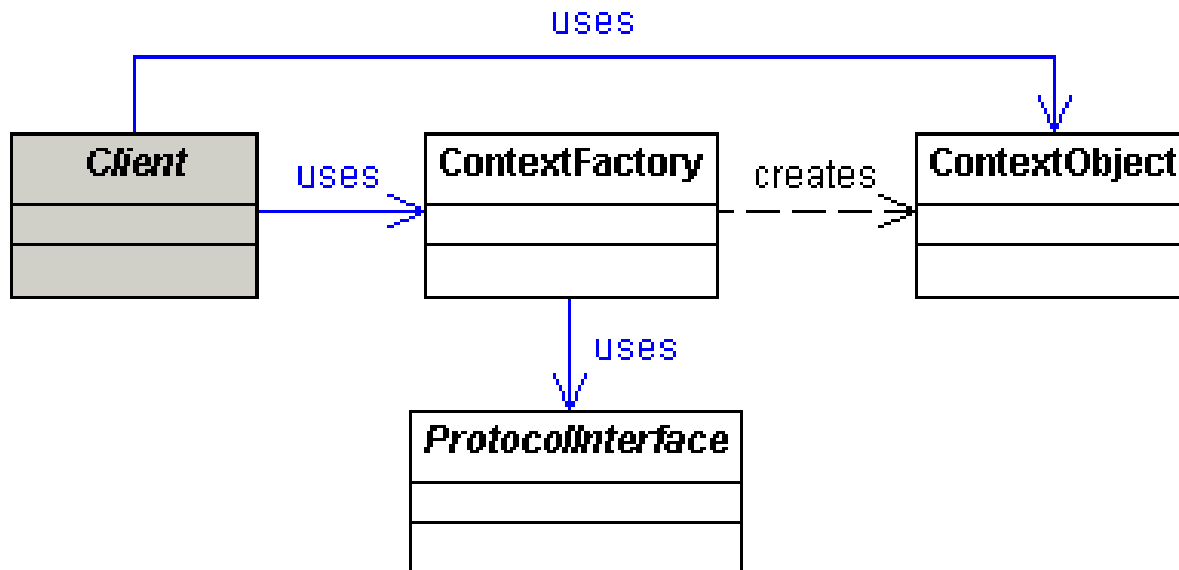
- Intercepting Filter
- Front Controller
- Context Object
- Application Controller
- View Helper
- Composite View
- Service To Worker
- Dispatcher View

Context Object

- Problem:
 - You want to avoid using protocol-specific system information outside of its relevant context
- Forces:
 - You have components and services that need access to system information
 - You want to decouple application components and services from the protocol specifics of system information
 - You want to 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 Strategies

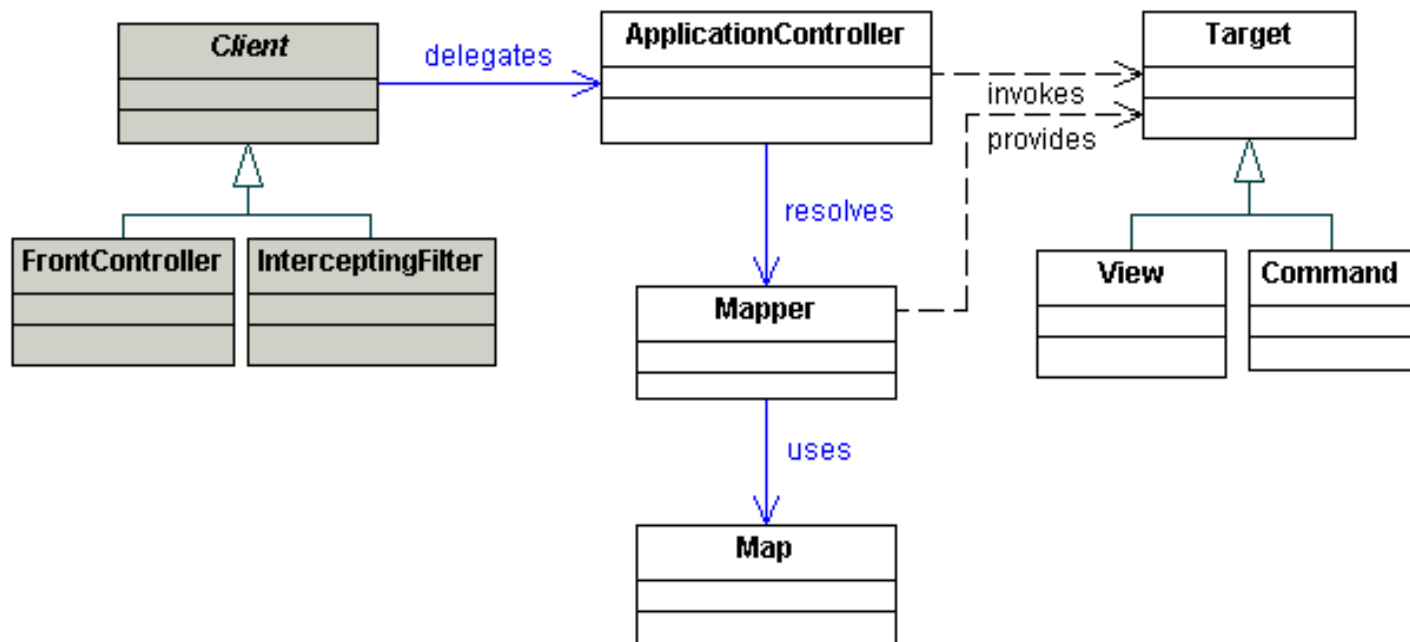
- Request Context Strategies
 - Request Context Map Strategy
 - Request Context POJO Strategy
 - Request Context Validation Strategy
- Configuration Context Strategies
 - JSTL Configuration Strategy
- Security Context Strategies
- General Context Object Strategies
 - Context Object Factory Strategy
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

Application Controller

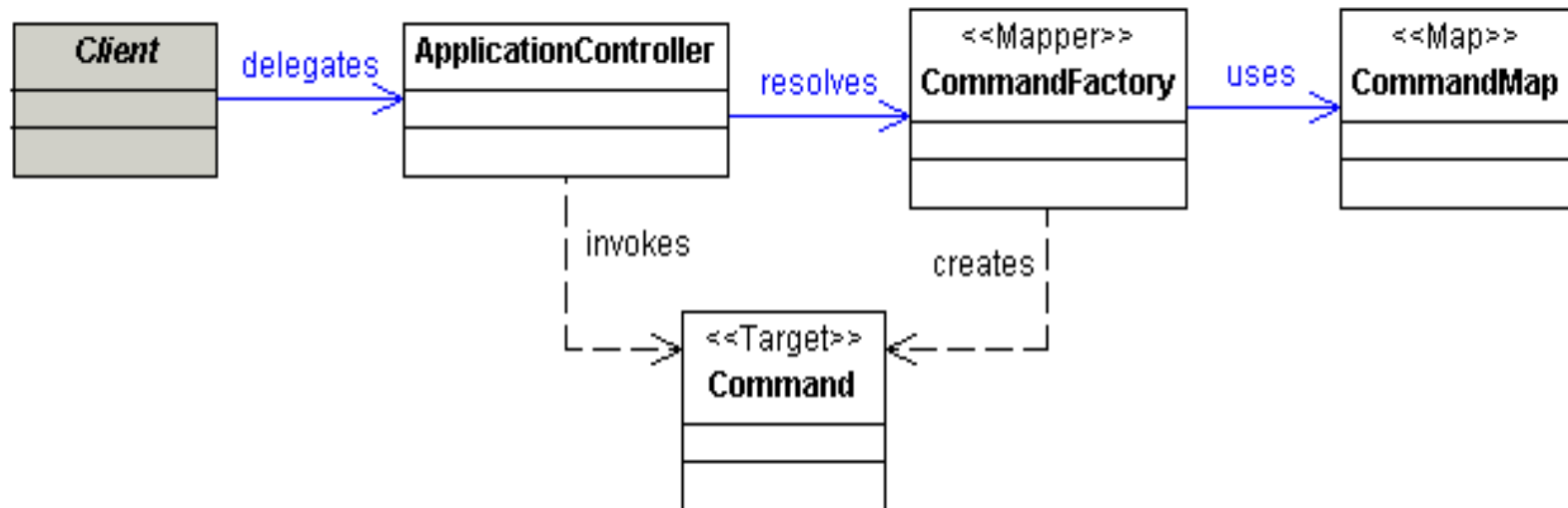
- Problem:
 - You want to centralize and modularize action and view management
- Forces:
 - You want to reuse action-management and view-management code
 - You want to improve code modularity and maintainability
 - You want dynamic lookup and dispatch to target

Application Controller

- Solution:
 - Use an Application Controller to centralize retrieval and invocation of request-processing components, such as commands and views.



Application Controller: Command Handler Strategy



Business Tier Patterns

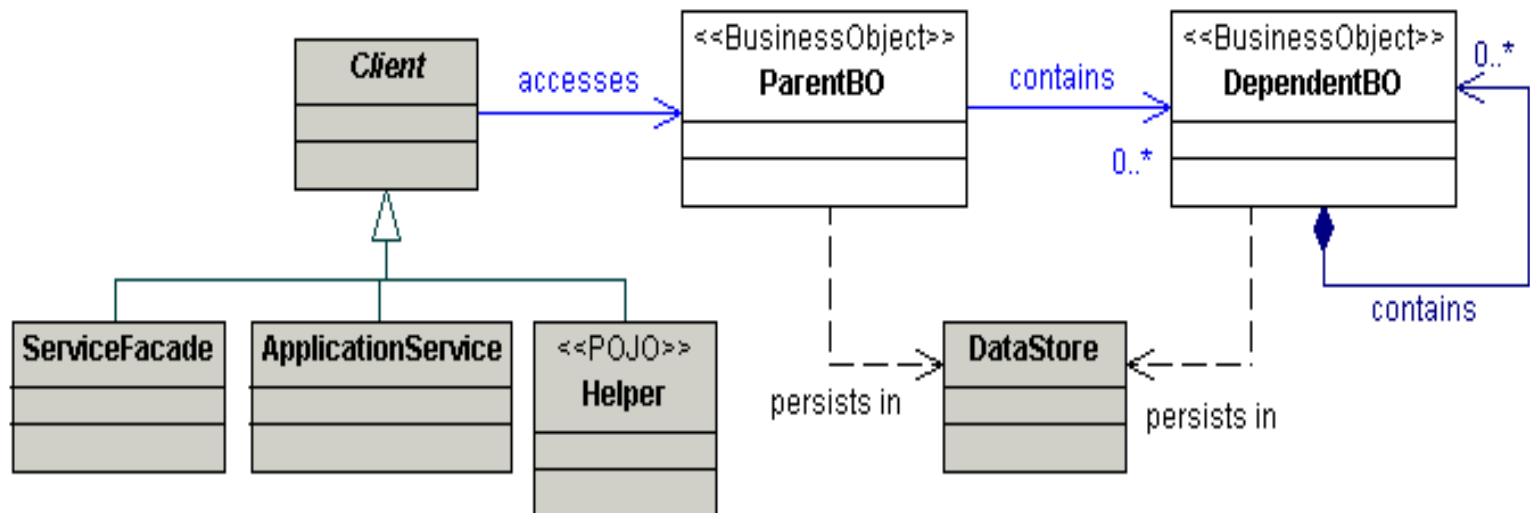
- Business Delegate
- Service Locator
- Session Facade
- Business Object
- Application Service
- Composite Entity
- Transfer Object
- Transfer Object Assembler
- Value List Handler

Business Object

- Problem:
 - You have a conceptual domain model with business logic and relationships
- Forces:
 - You have a conceptual model containing structured, interrelated composite objects, complex business logic, validation, rules
 - You want to centralize business logic and state in an application
 - You want to 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

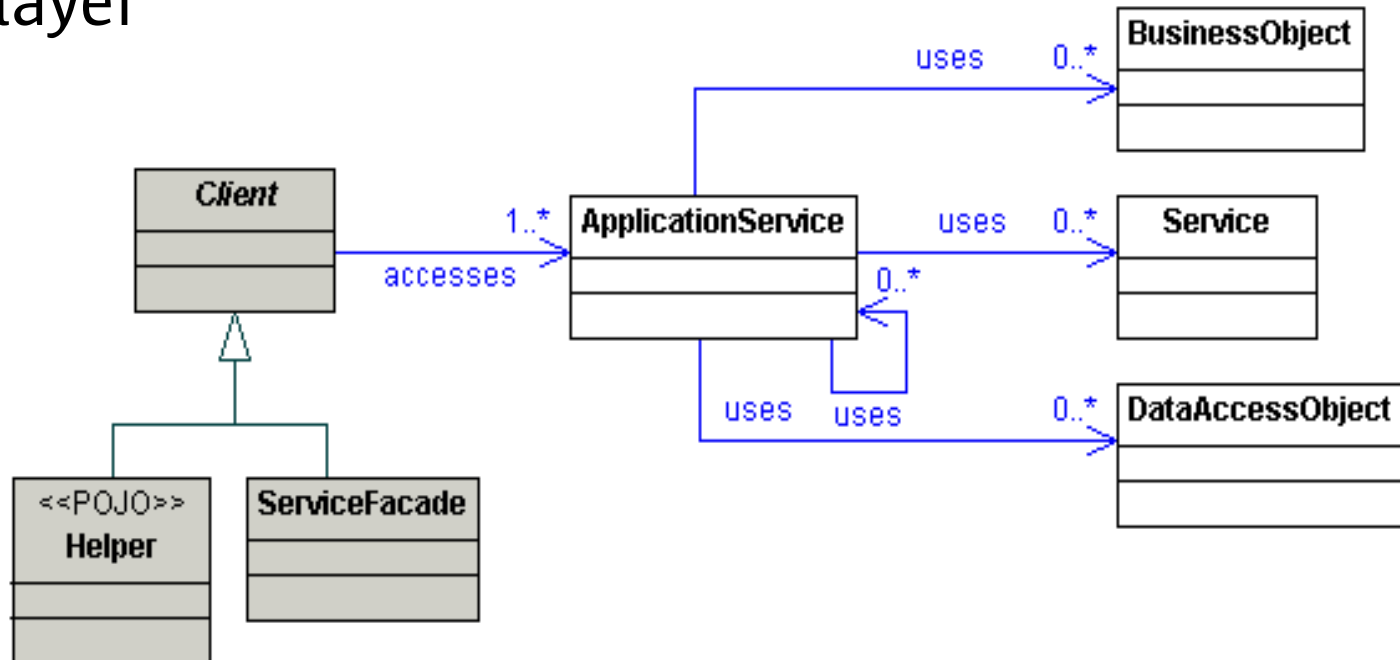


Application Service

- Problem:
 - You want to centralize business logic across several business-tier components and services
- Forces:
 - You want to minimize business logic in service facades
 - You have business logic acting on multiple Business Objects or services
 - You want to encapsulate use case-specific logic outside of individual Business Objects

Application Service

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



Integration Tier Patterns

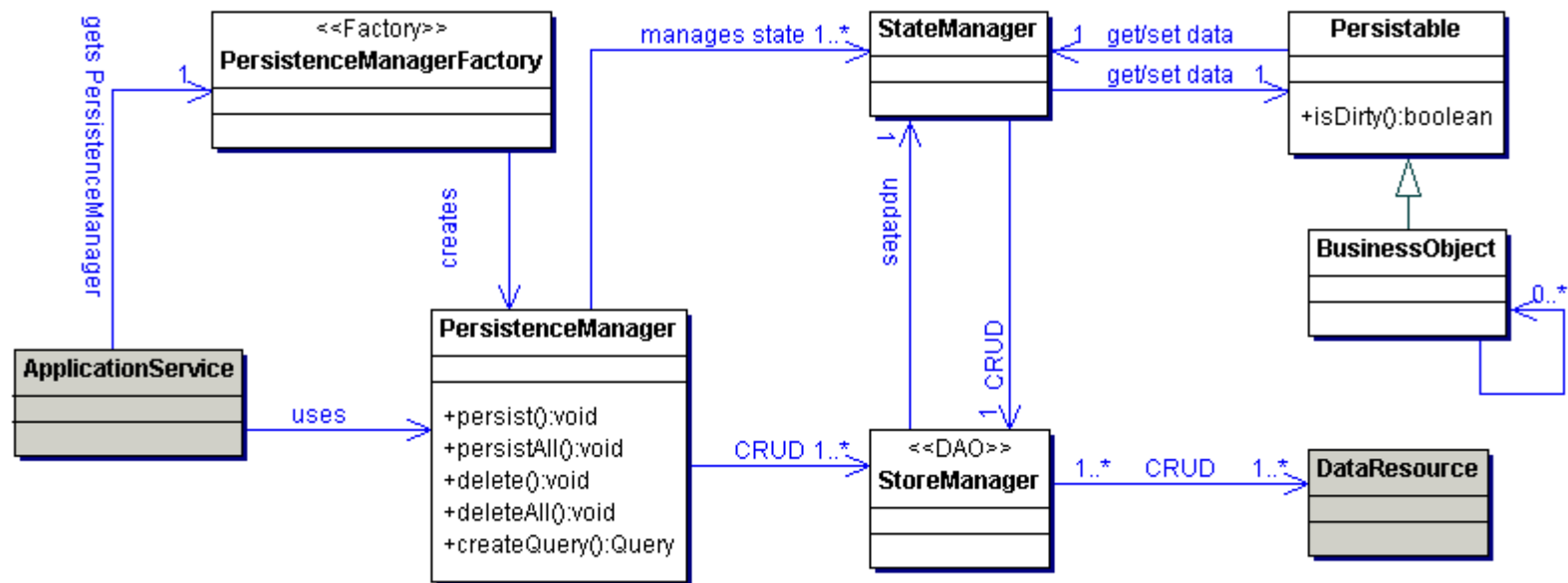
- Data Access Object
- Service Activator
- Domain Store
- Web Service Broker

Domain Store

- Problem:
 - You want to separate persistence from your object model
- Forces:
 - You want to avoid putting persistence details in your Business Objects
 - You do not want to use entity beans
 - Your application might be running in a web container
 - Your object model uses inheritance and complex relationships

Domain Store

- Solution:
 - Use Domain Store to separate persistence from the object model

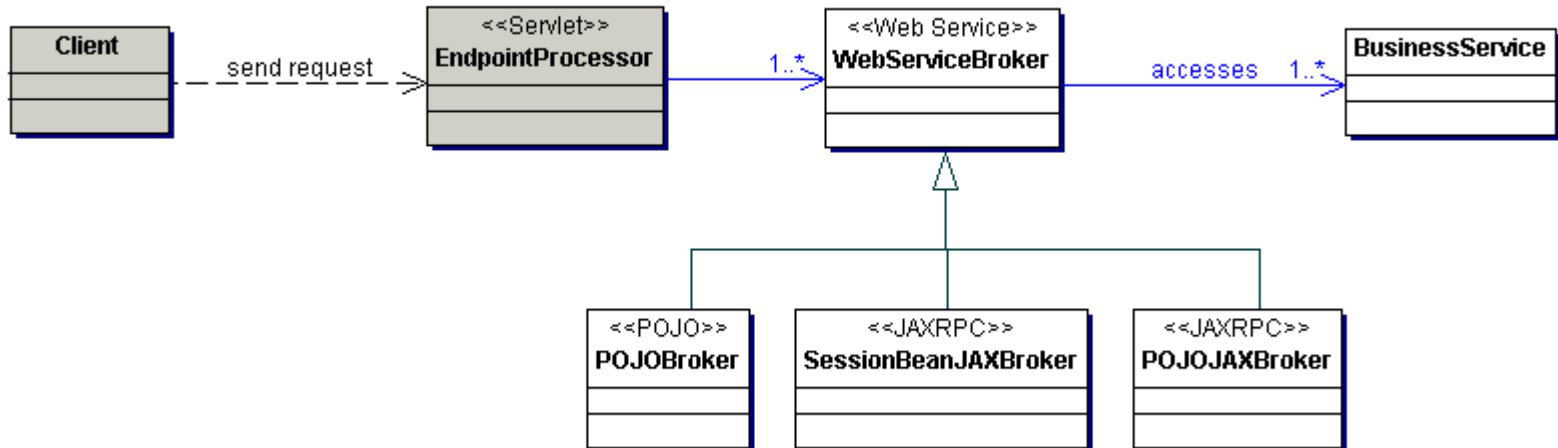


Web Service Broker

- Problem:
 - You want to provide access to one or more services using XML and web protocols
- Forces:
 - You want to reuse and expose existing services to clients
 - You want to monitor and potentially limit the usage of exposed services
 - Your services must be exposed using open standards

Web Service Broker

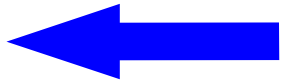
- Solution:
 - Use a Web Service Broker to expose and broker one or more services using XML and web protocols



Web Service Broker: Strategies

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

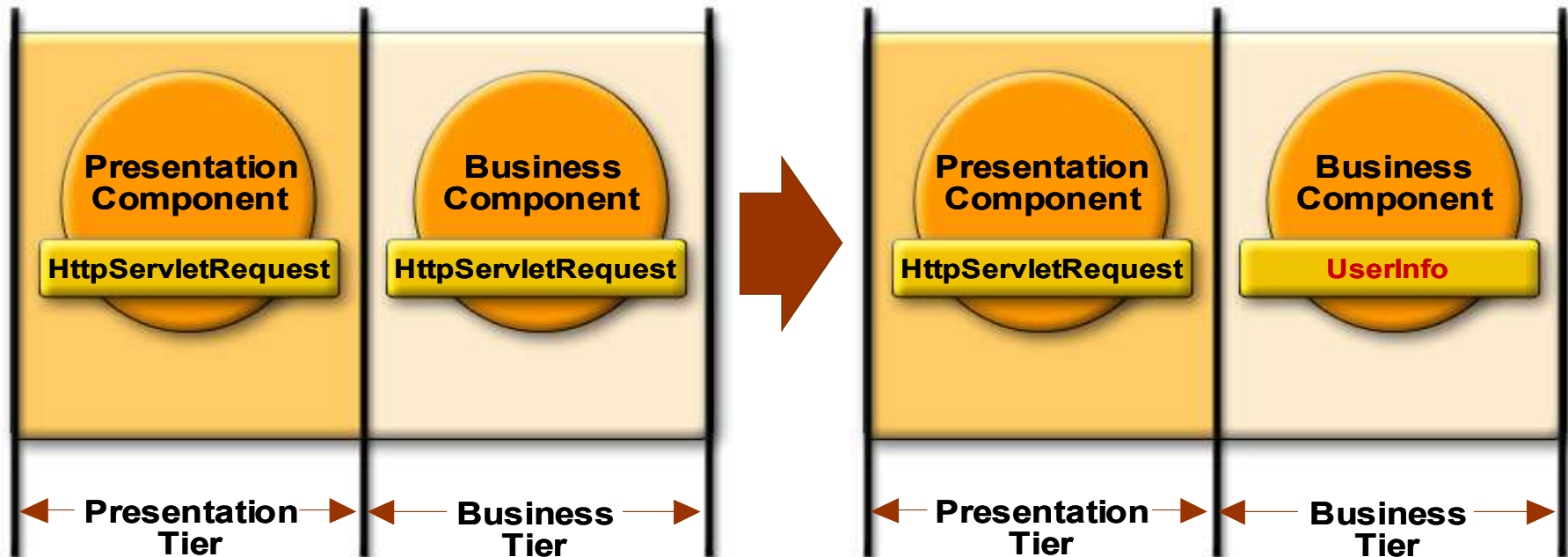
Agenda

- Patterns
- Core J2EE Pattern Catalog Background
- J2EE Progressive Refactoring Scenarios 
- Pattern Frameworks
- Micro Architecture
 - Web Worker Micro Architecture Example
 - Messaging Micro Architecture Example
- Q&A

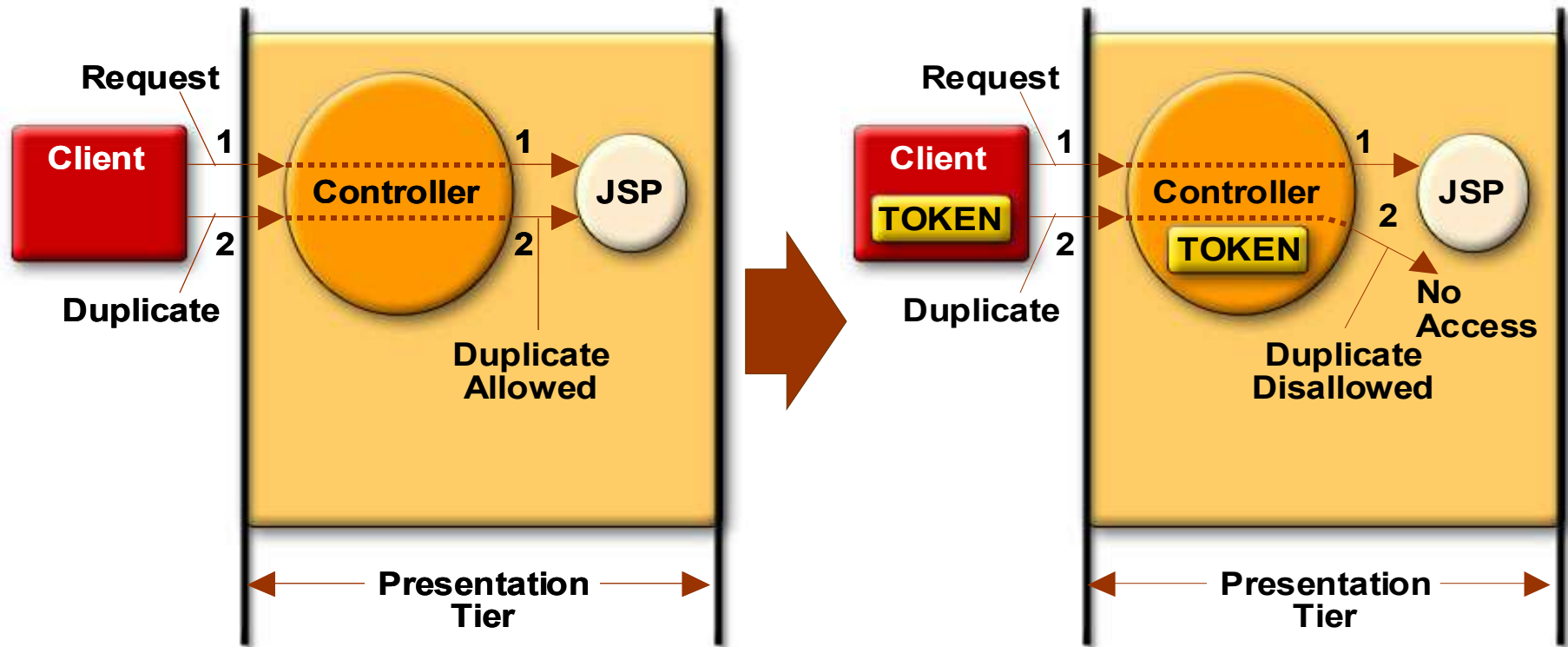
J2EE Refactoring

- 14 Refactorings in the book
- Presentation Tier:
 - Hide Presentation Tier specifics from Business Tier
 - Introduce Synchronizer Token
- Business Tier:
 - Wrap Entities with Session
 - Merge Session Beans

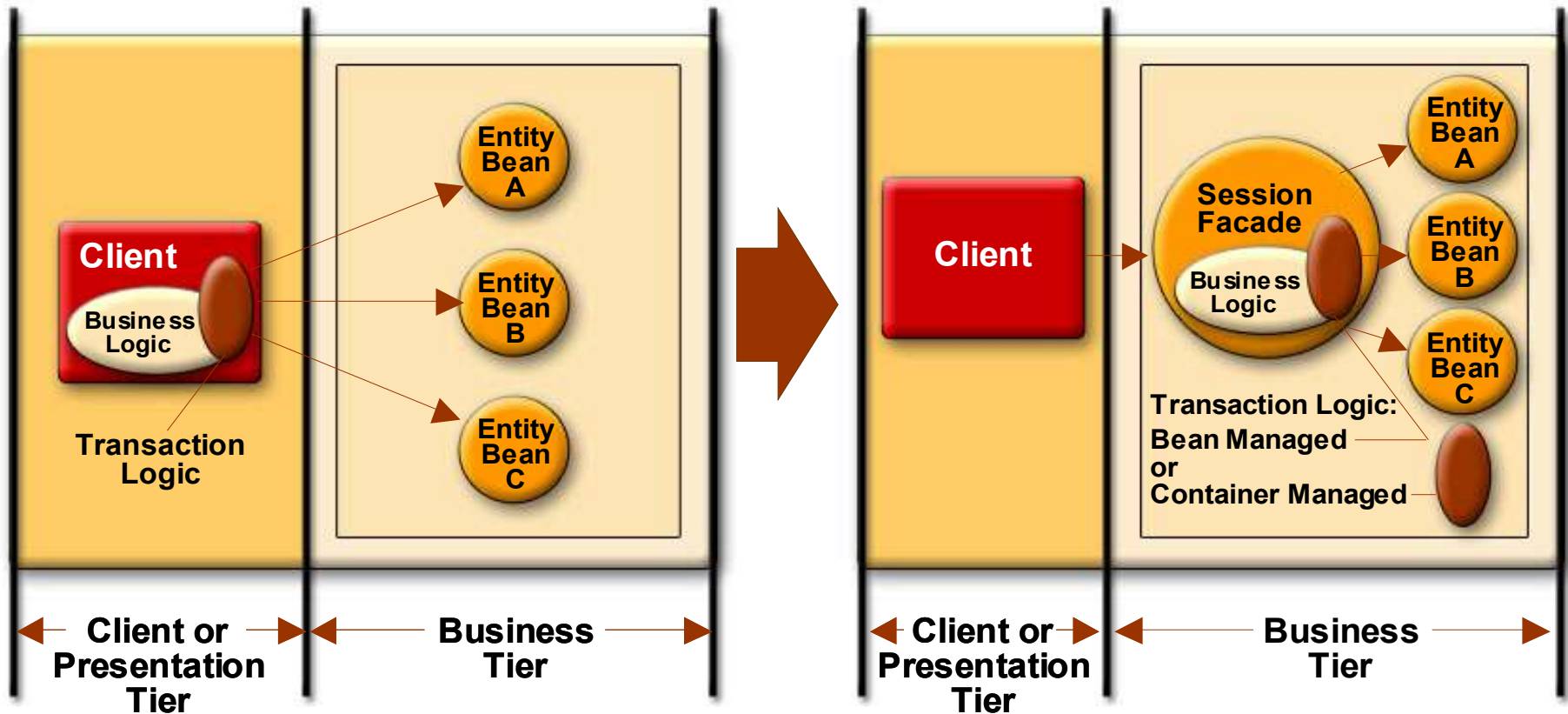
Hide Presentation Tier specifics...



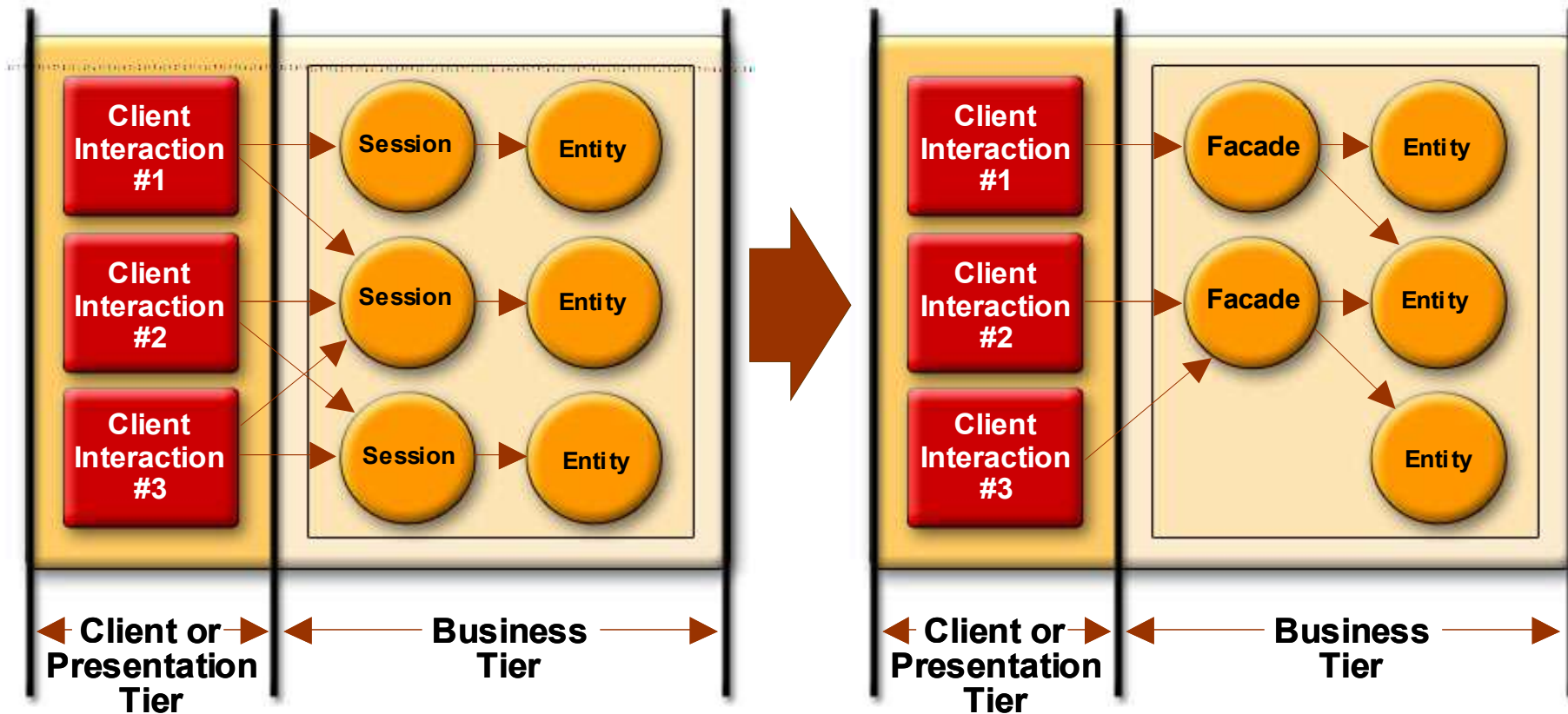
Introduce Synchronizer Token



Wrap Entities With Session



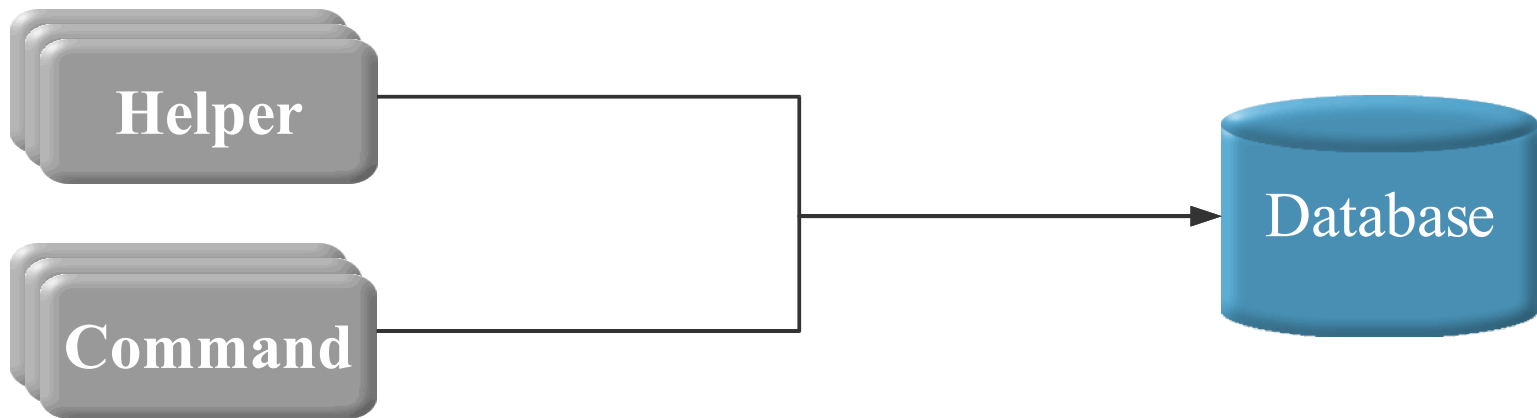
Merge Session Beans



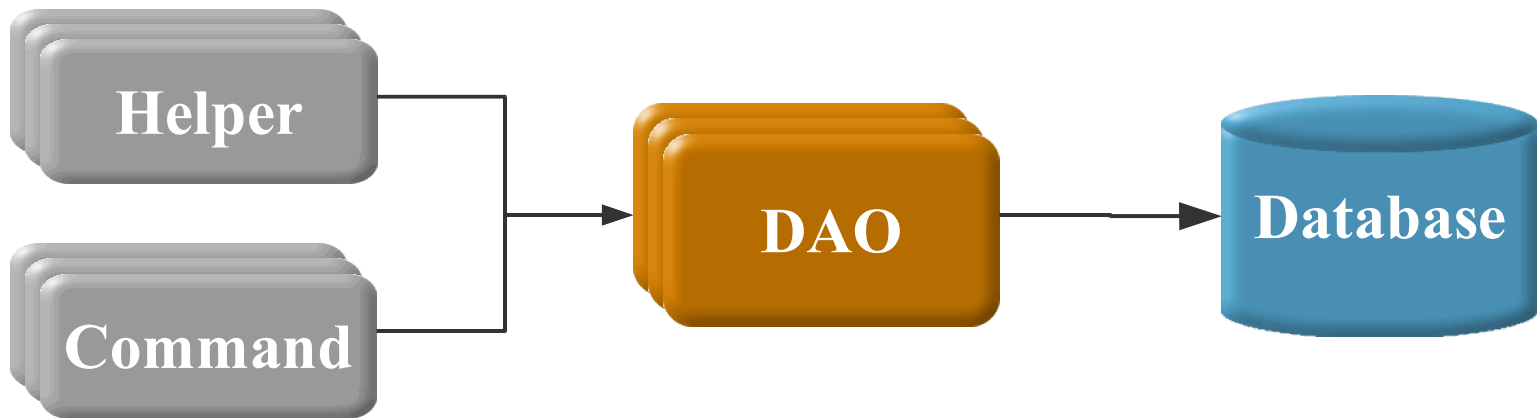
Progressive Refactoring Scenarios

- Direct Access
- Introduce DAO
- Introduce Application Service
- Introduce Service Facade
- Introduce Business Objects

Direct Access

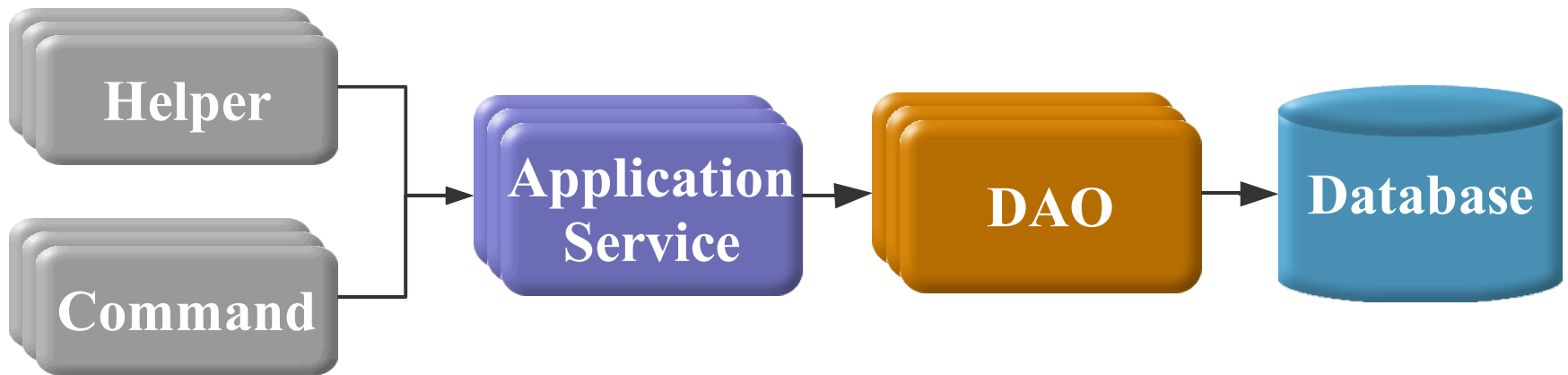


Introduce DAO



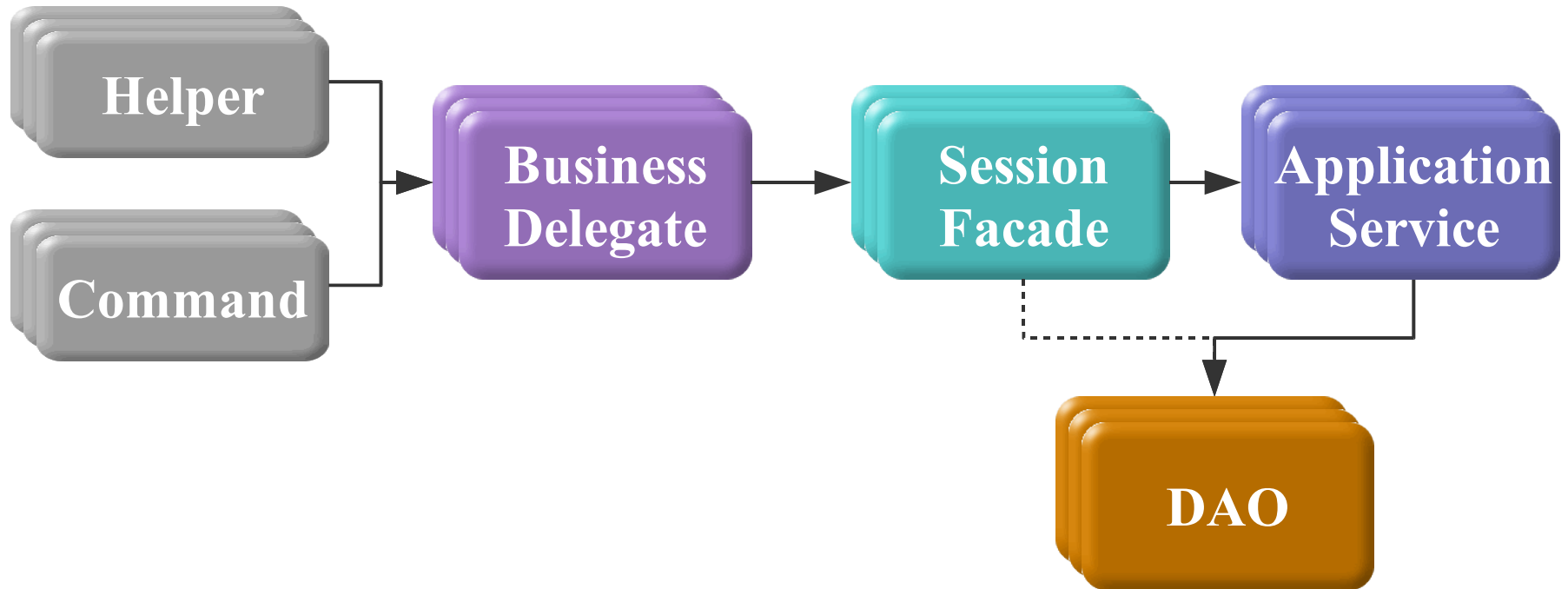
Introduce Application Service

POJO Architecture



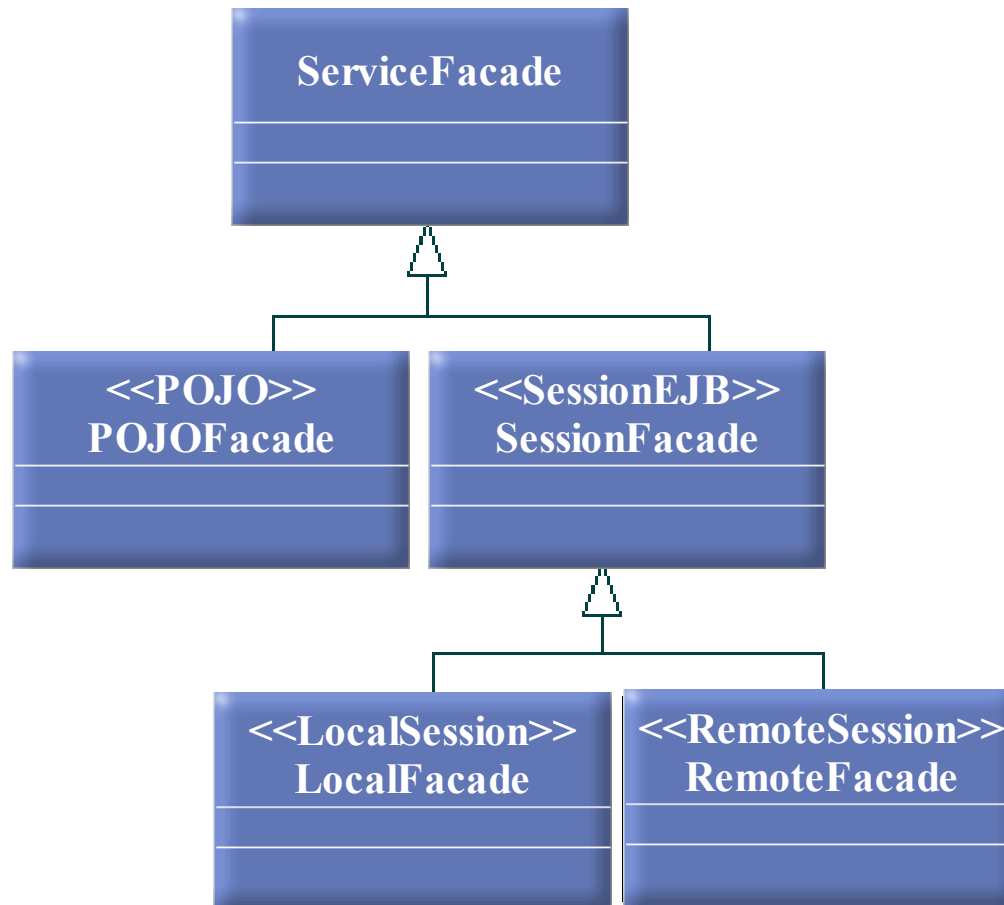
Introduce Application Service

EJB Architecture



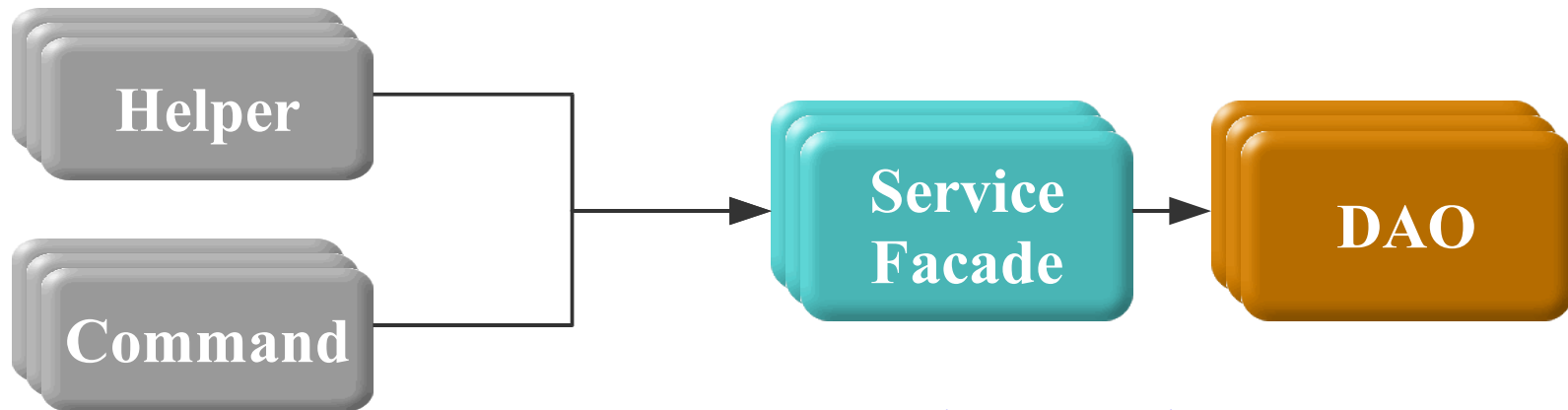
Design Note: Service Facades

- Remote and non-Remote business tier



Introduce Service Facade

Non-Remote Business Tier



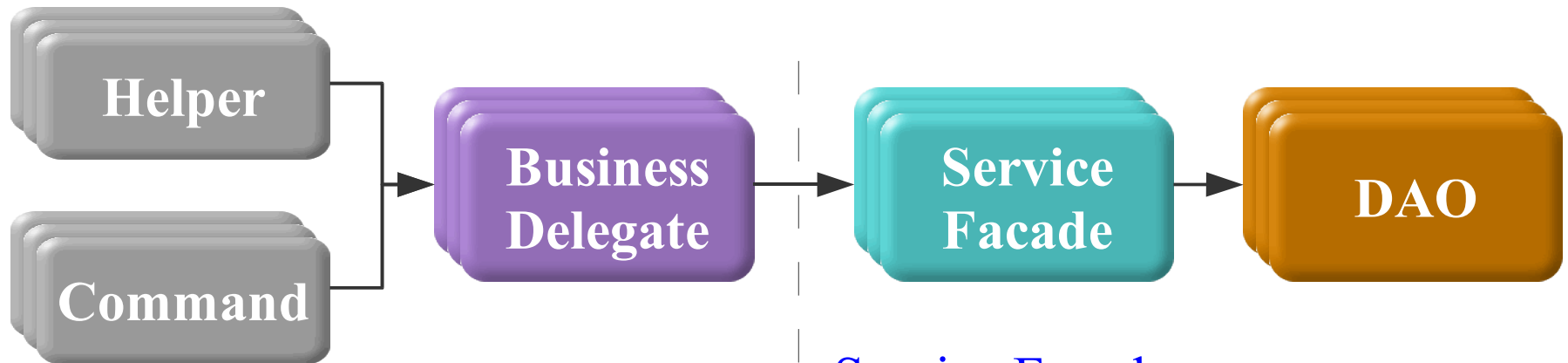
Service Facade >>

Local Facade >>

Local Session Bean | POJO

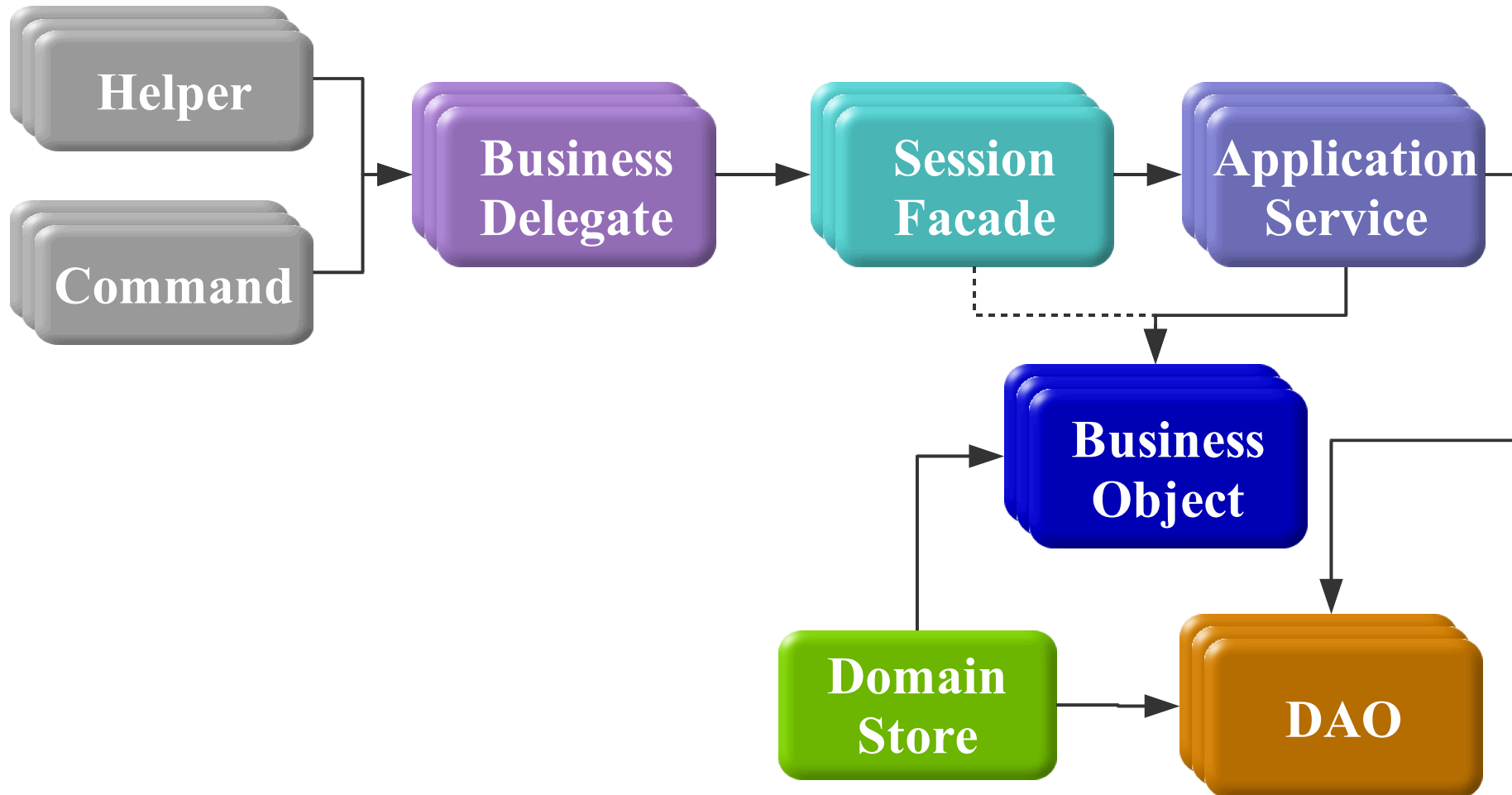
Introduce Service Facade

Remote Business Tier



Service Facade >>
Remote Facade >>
Remote Session Bean

Introduce Business Objects

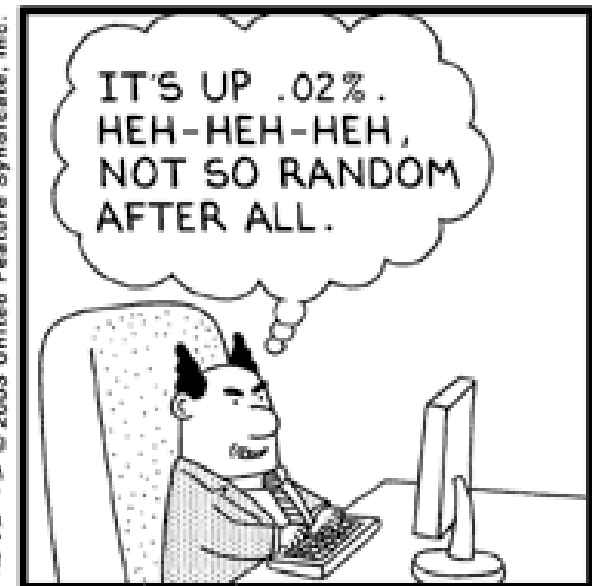




www.dilbert.com
scottadam@aol.com

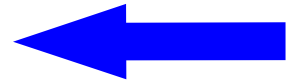


14-13-03 © 2003 United Feature Syndicate, Inc.



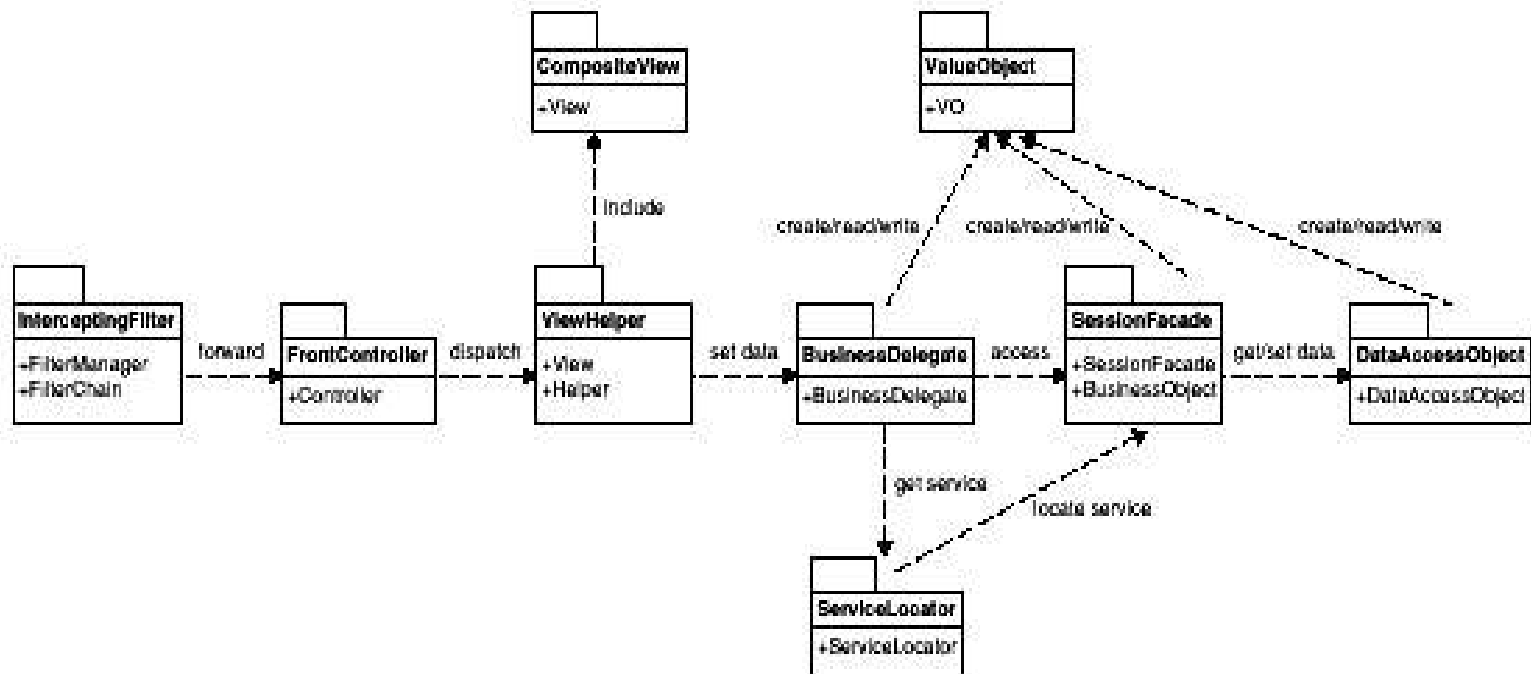
Agenda

- Patterns
- Core J2EE Pattern Catalog Background
- J2EE Progressive Refactoring Scenarios
- Pattern Frameworks
- Micro Architecture
 - Web Worker Micro Architecture Example
 - Messaging Micro Architecture Example
- Q&A



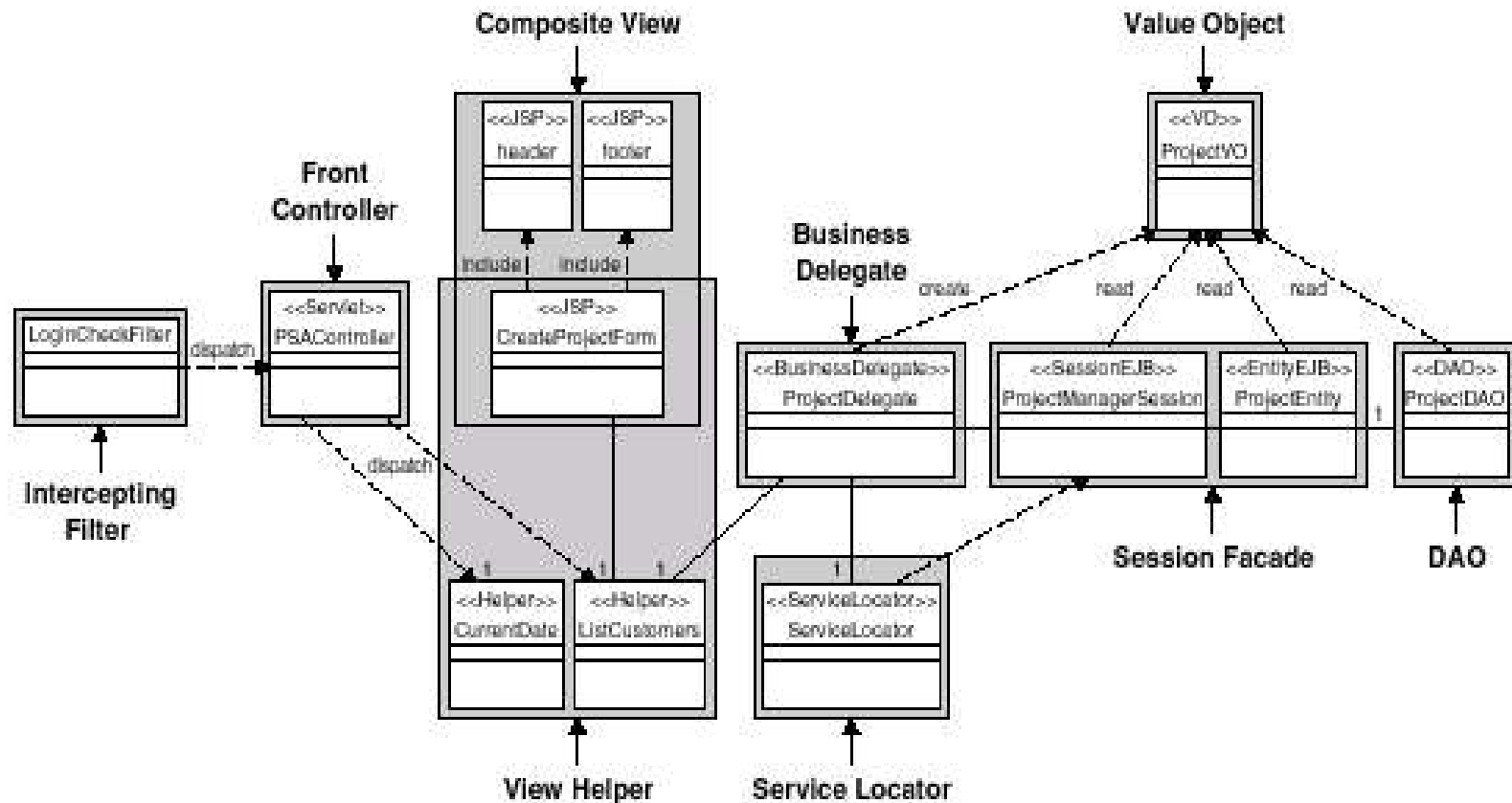
Pattern Framework

- Set of cooperating patterns
- Targeting macro problem
- Basis for pattern driven design



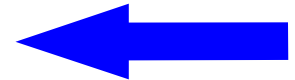
Pattern Realization

- Realizing patterns to code



Agenda

- Patterns
- Core J2EE Pattern Catalog Background
- J2EE Progressive Refactoring Scenarios
- Pattern Frameworks
- Micro Architecture
 - Web Worker Micro Architecture Example
 - Messaging Micro Architecture Example
- Q&A

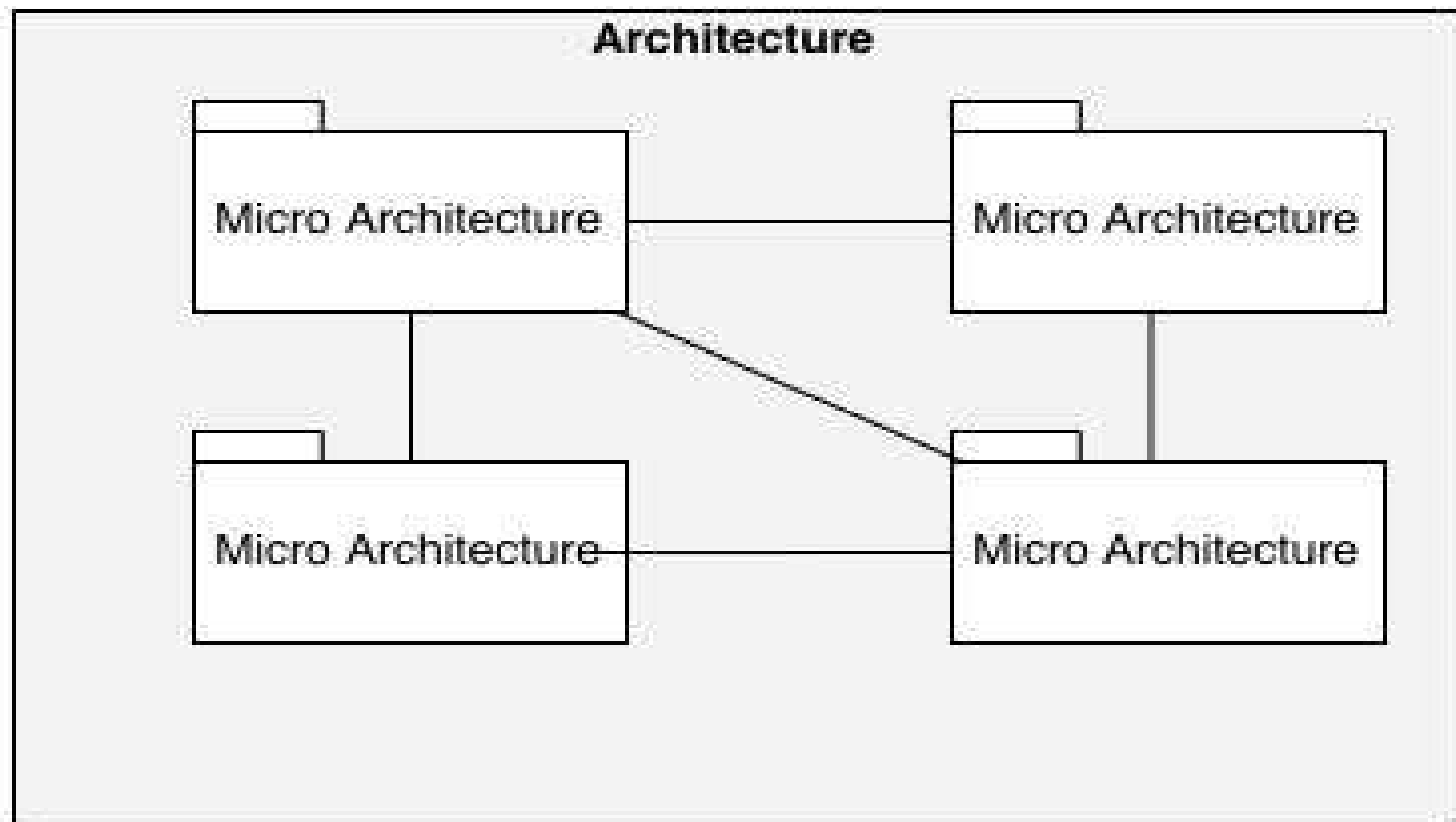


Micro Architectures

- Micro-architectures are building blocks for designing applications
- They represent a higher level of abstraction than the individual patterns described in the catalog, and are expressed by a combination of patterns to solve a problem
- Micro-architecture is a prescriptive design leveraging patterns to solve a larger problem, such as designing a subsystem
- Micro-Architectures:
 - WebWorker Micro Architecture
 - Messaging Micro Architecture

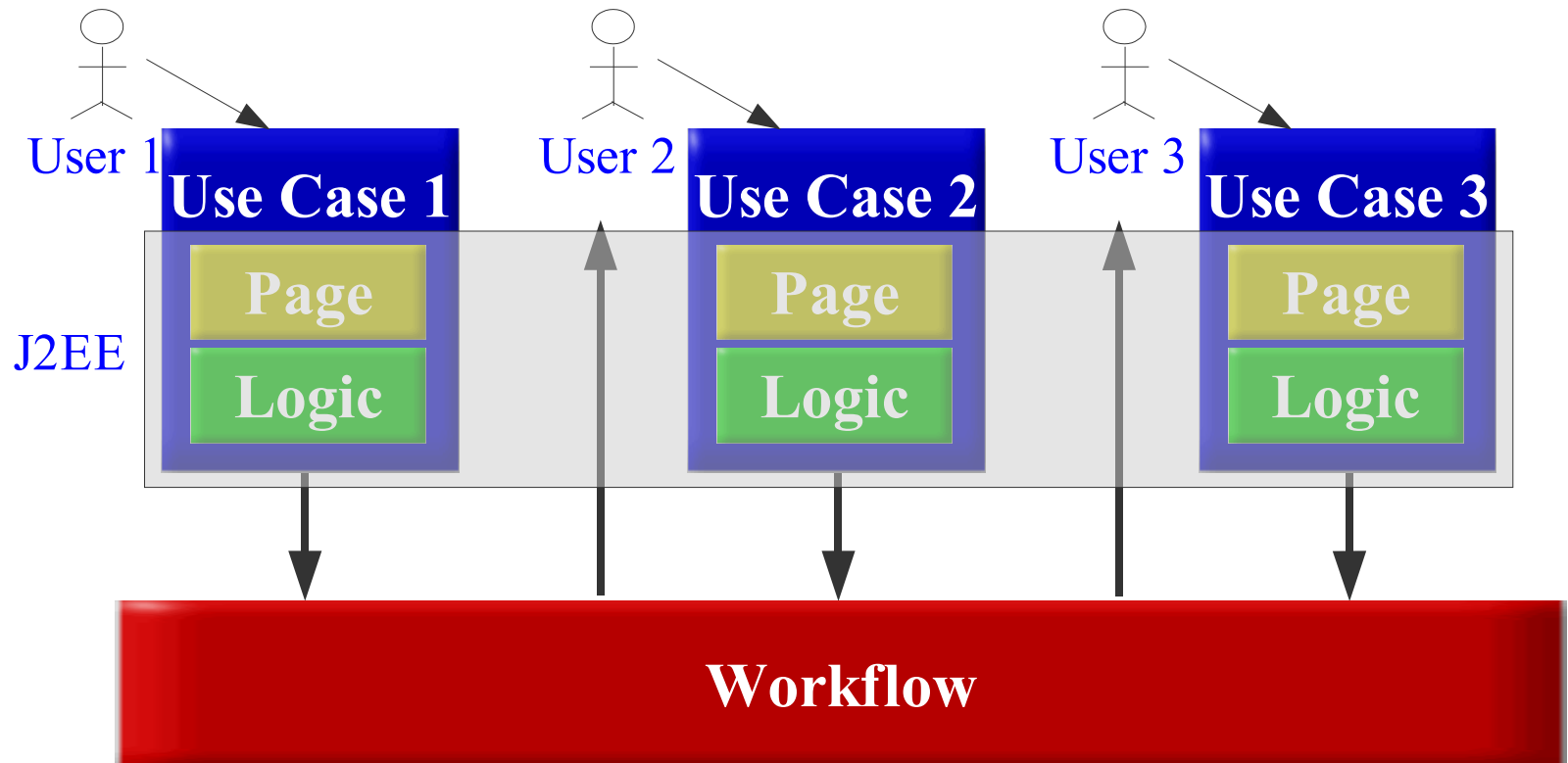
Micro Architectures

An Architecture is composed
of several Micro Architectures

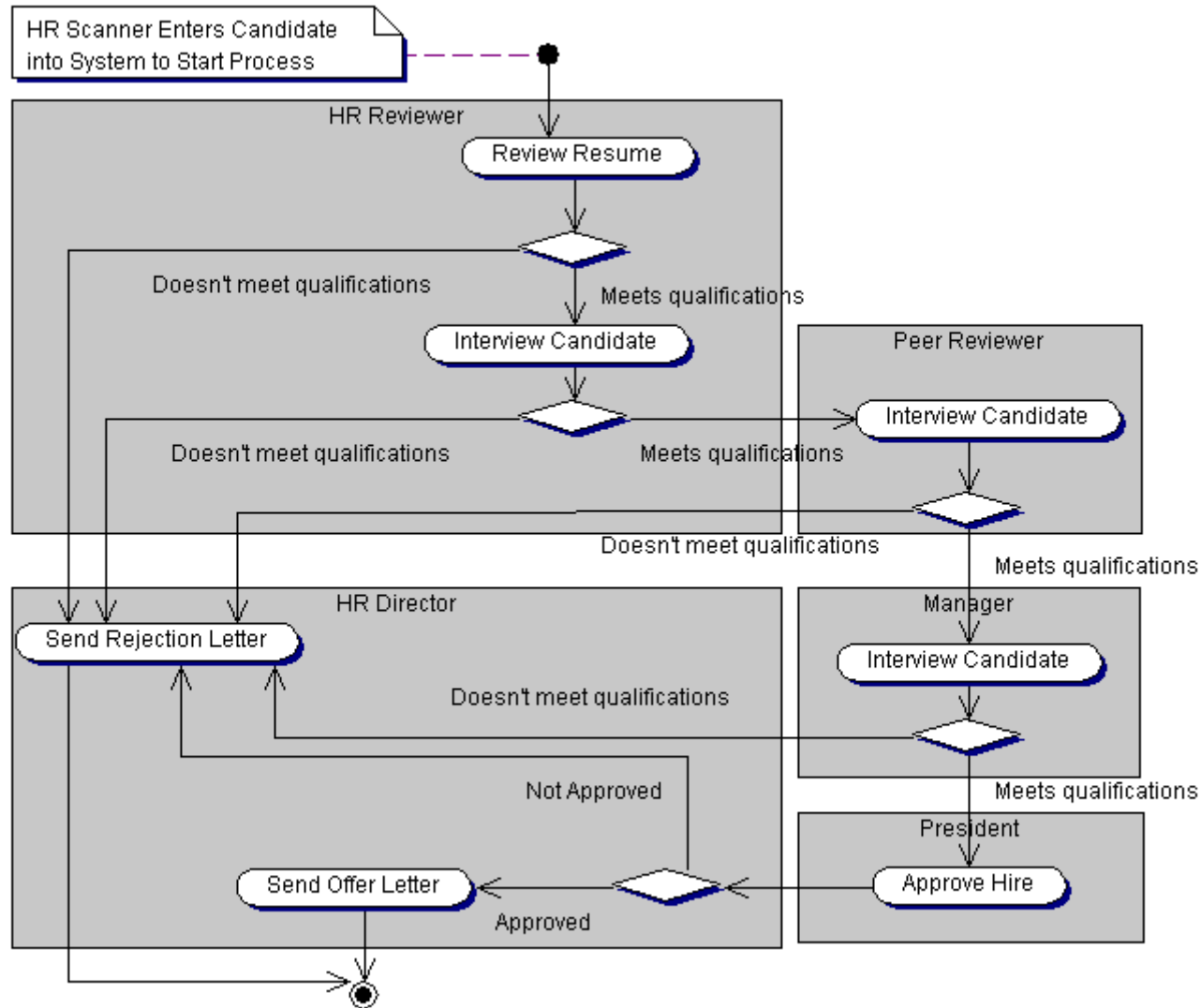


Web Worker Micro Architecture

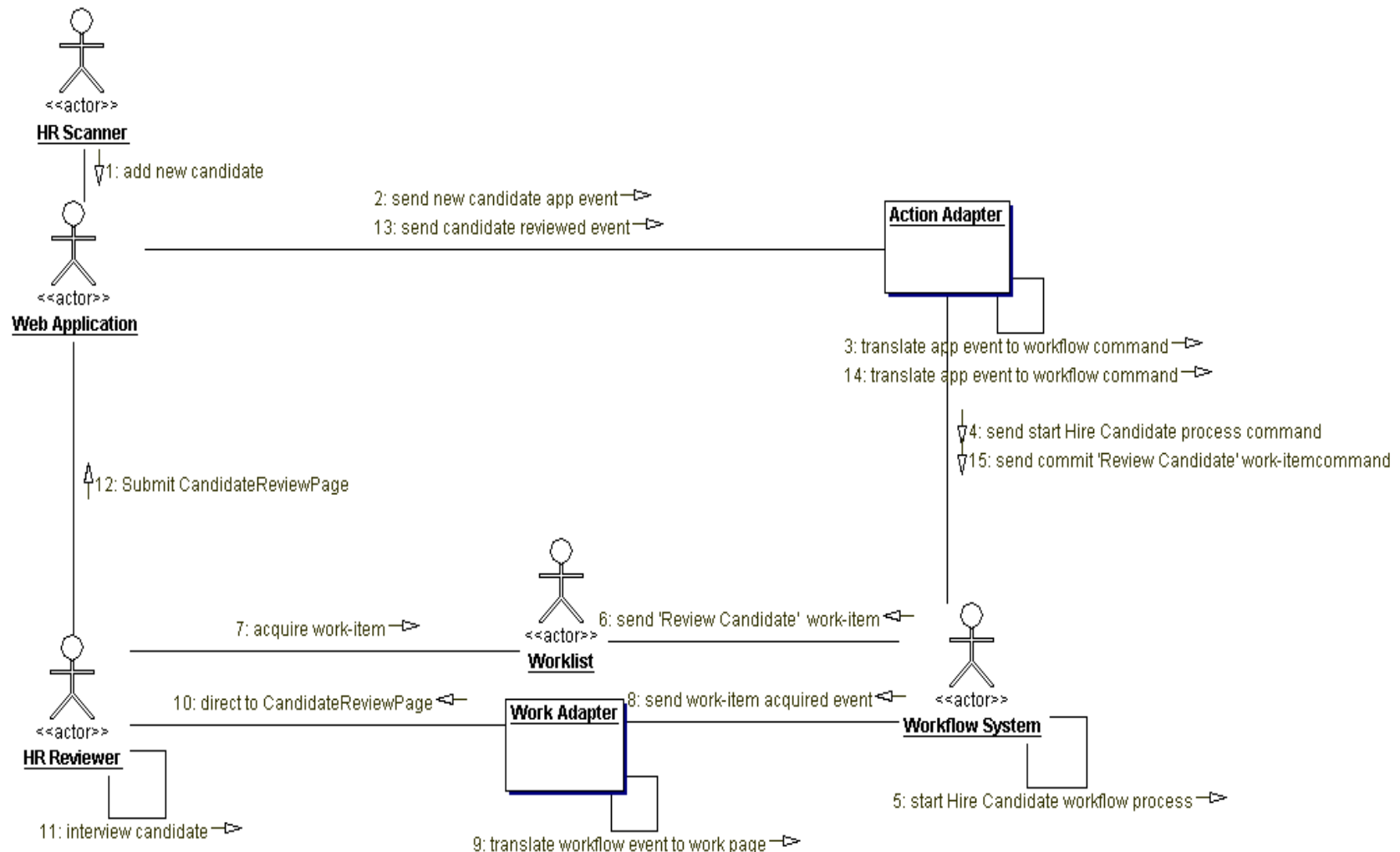
- Problem:
 - How do you integrate a J2EE application and a workflow system and have the workflow system direct users to the appropriate web page



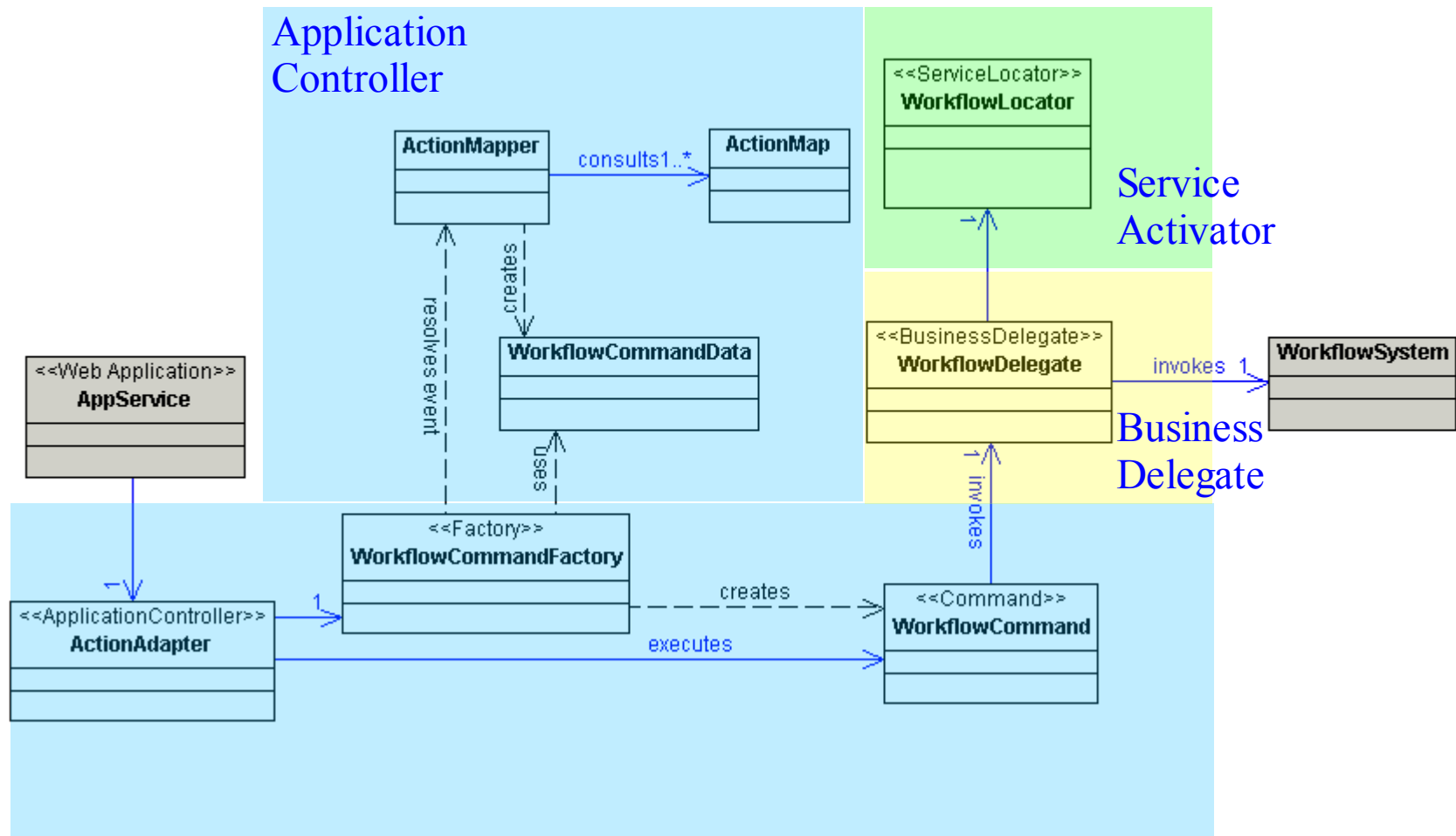
Hire Employee Workflow



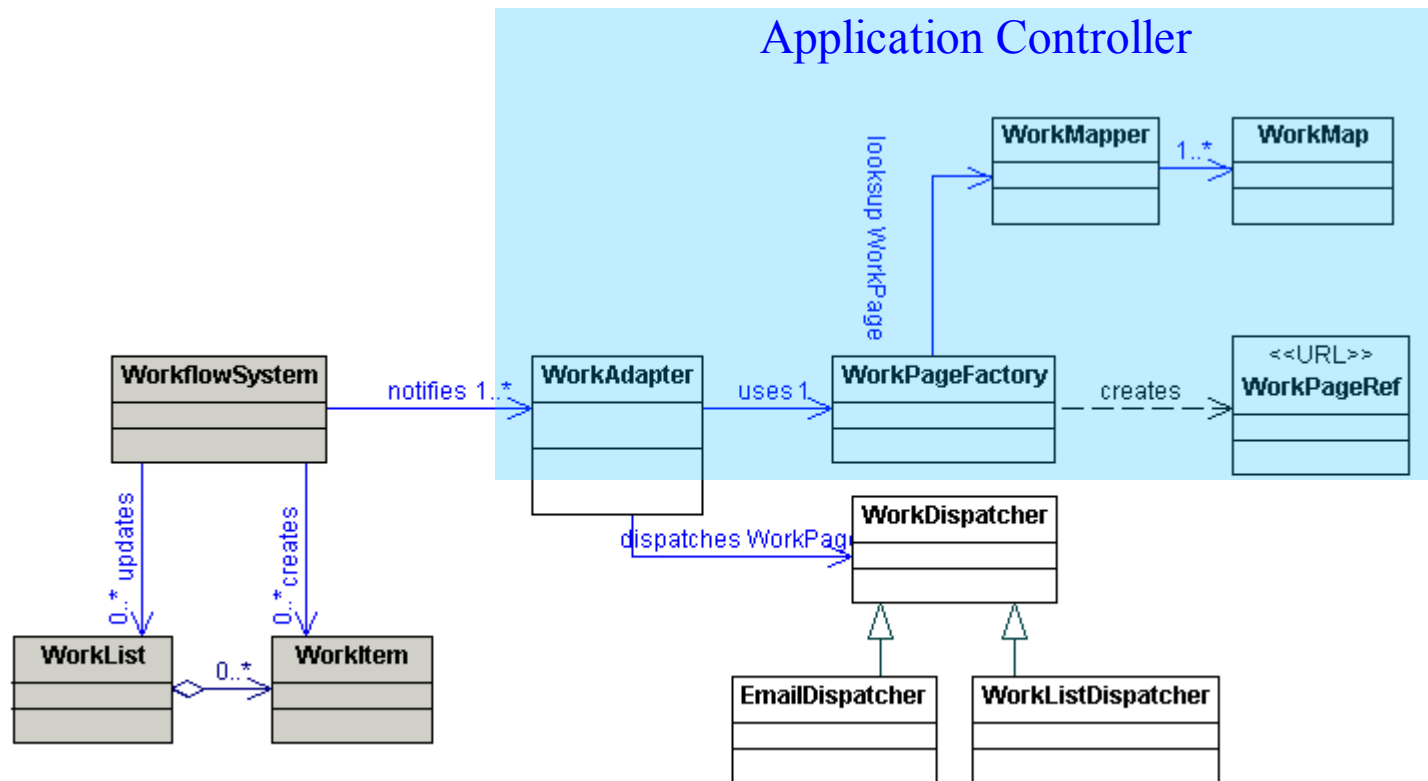
Hire Employee Collaboration with Adapters



Action Adapter Class Diagram



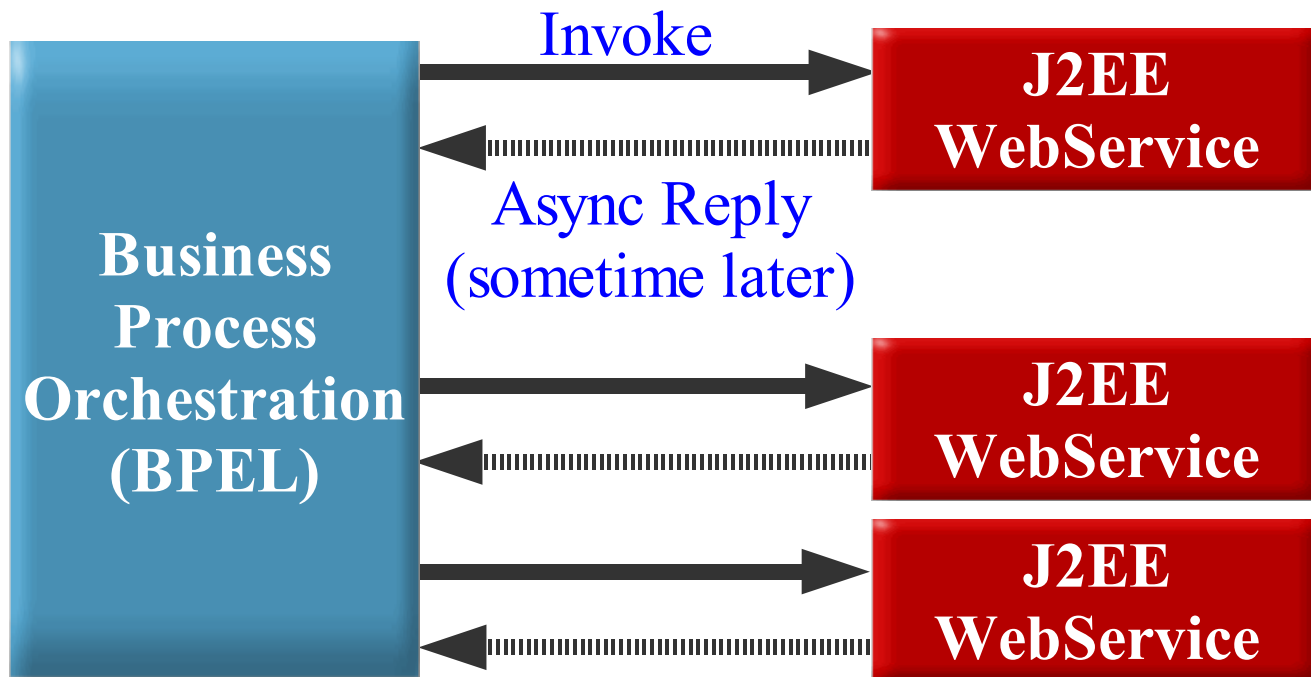
Work Adapter Class Diagram



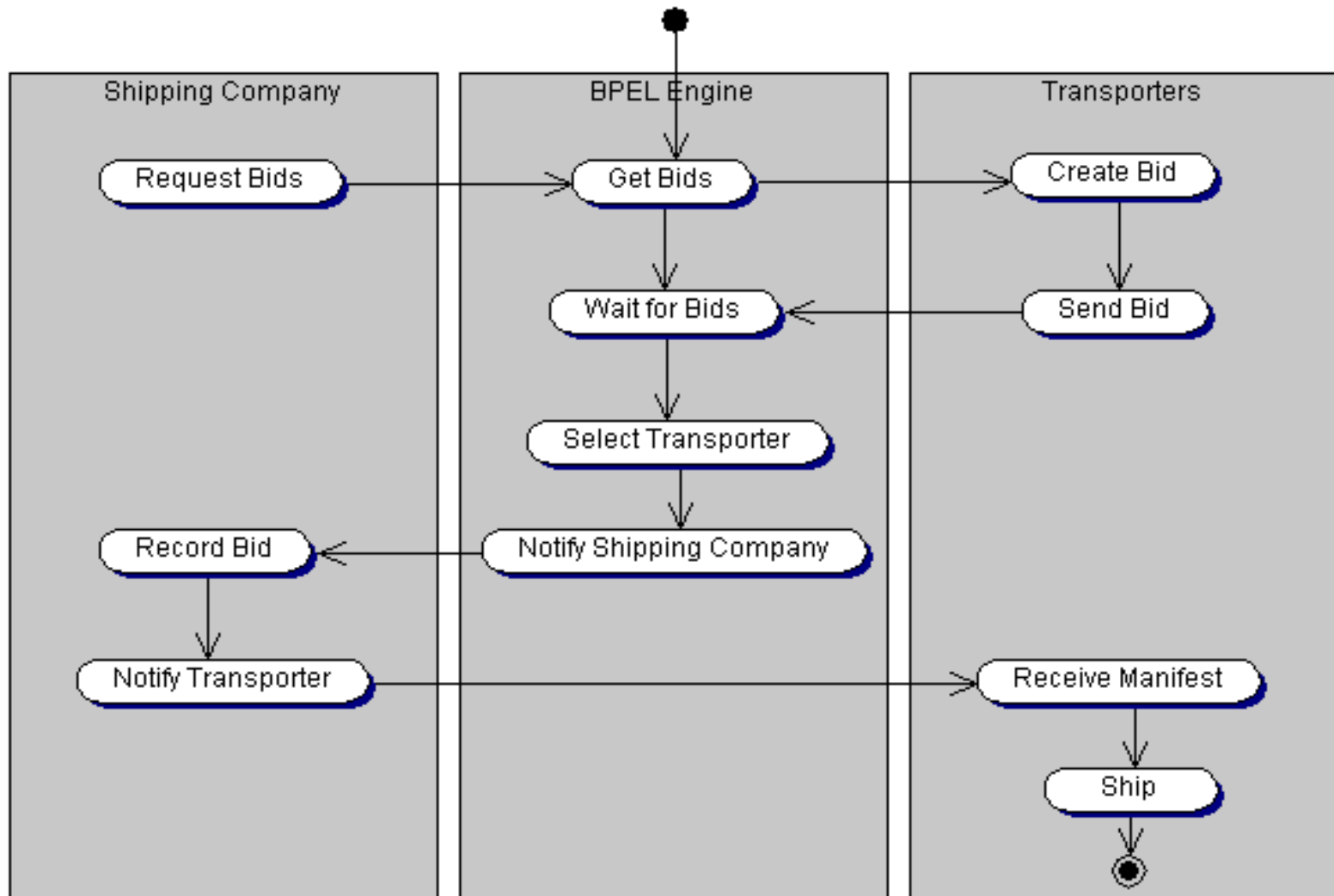
Messaging Micro Architecture

- Messaging >> Async, Web Services
- Problem:
 - How do you provide async, doc-based web services in J2EE
 - How do you orchestrate these web services

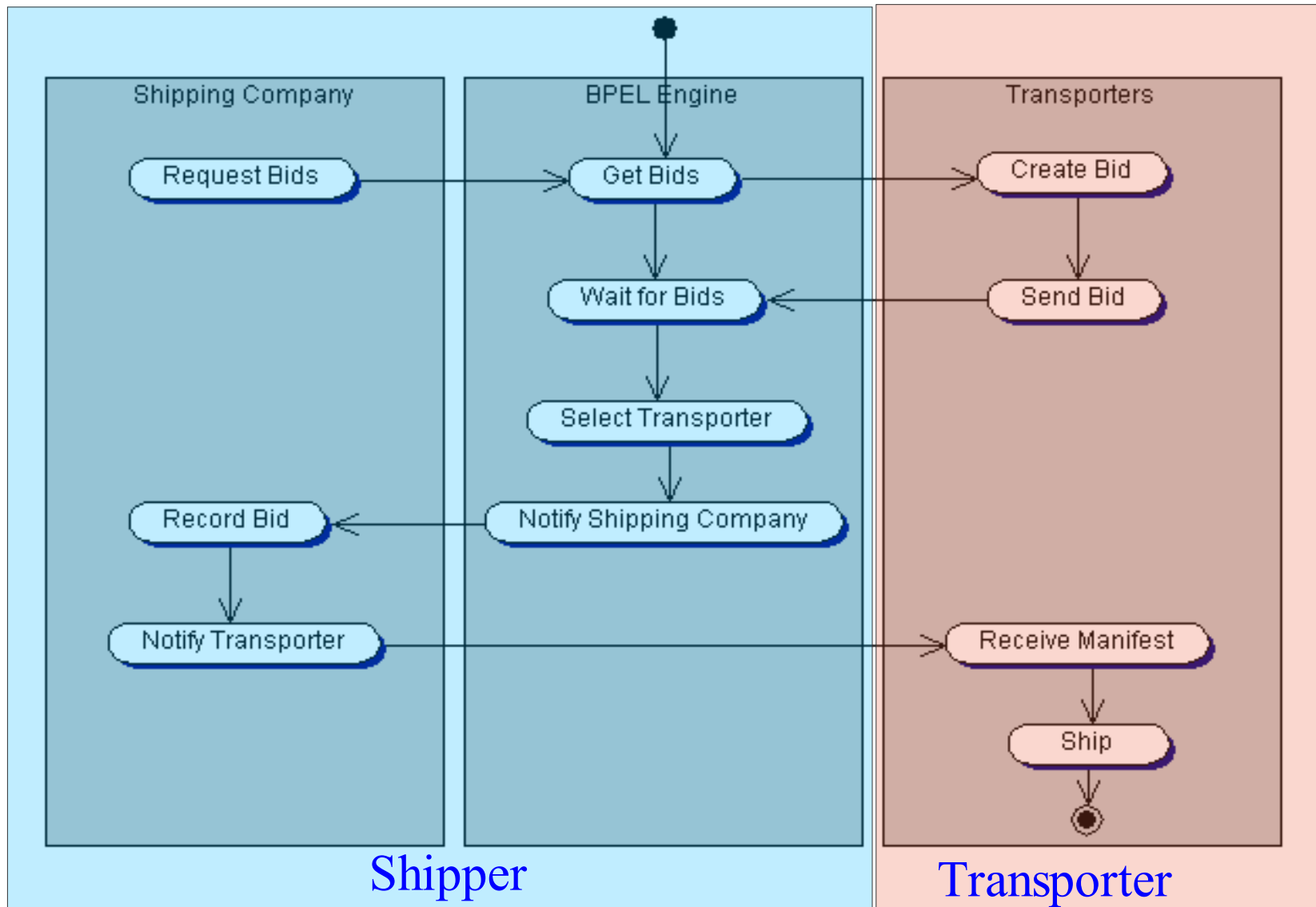
Async WS Orchestration With J2EE



Shipping Example

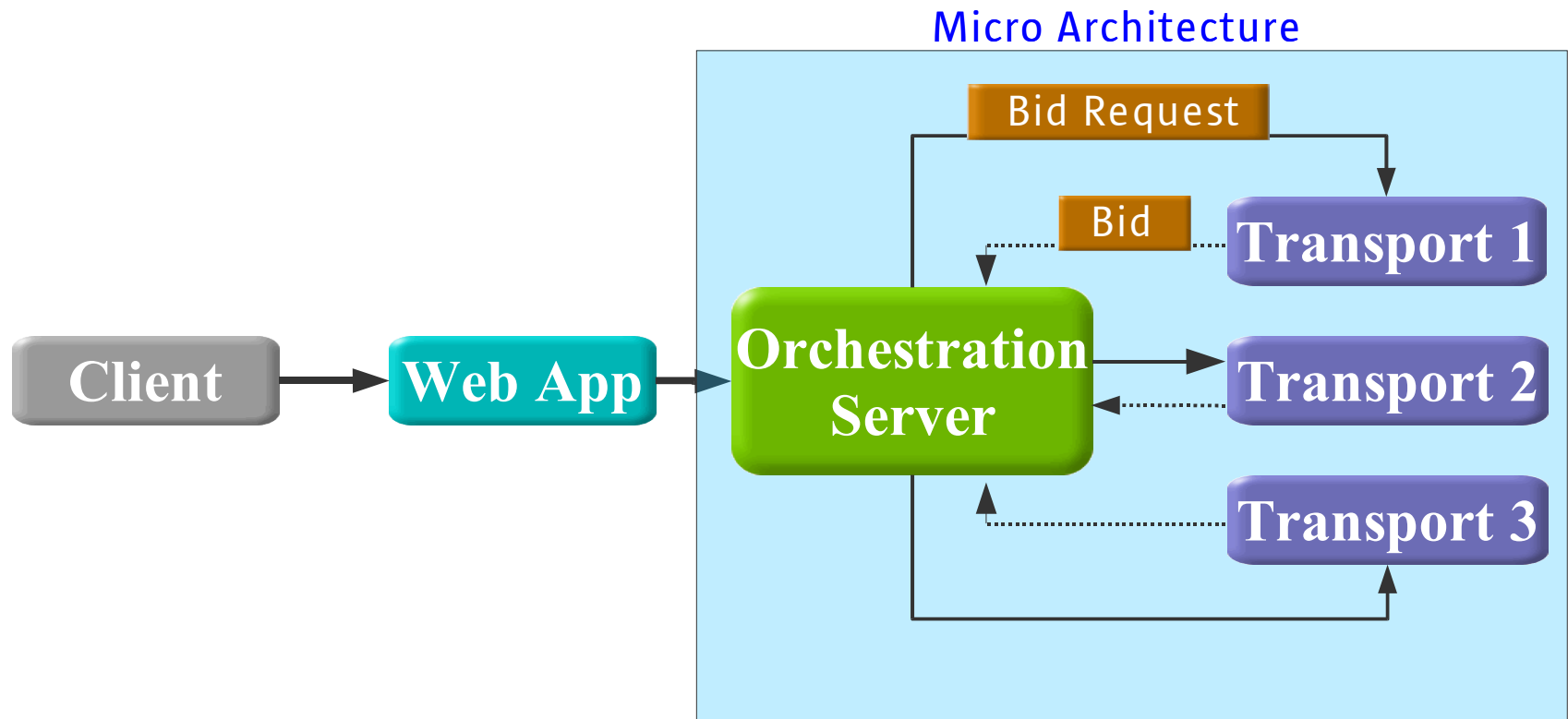


Shipping Example

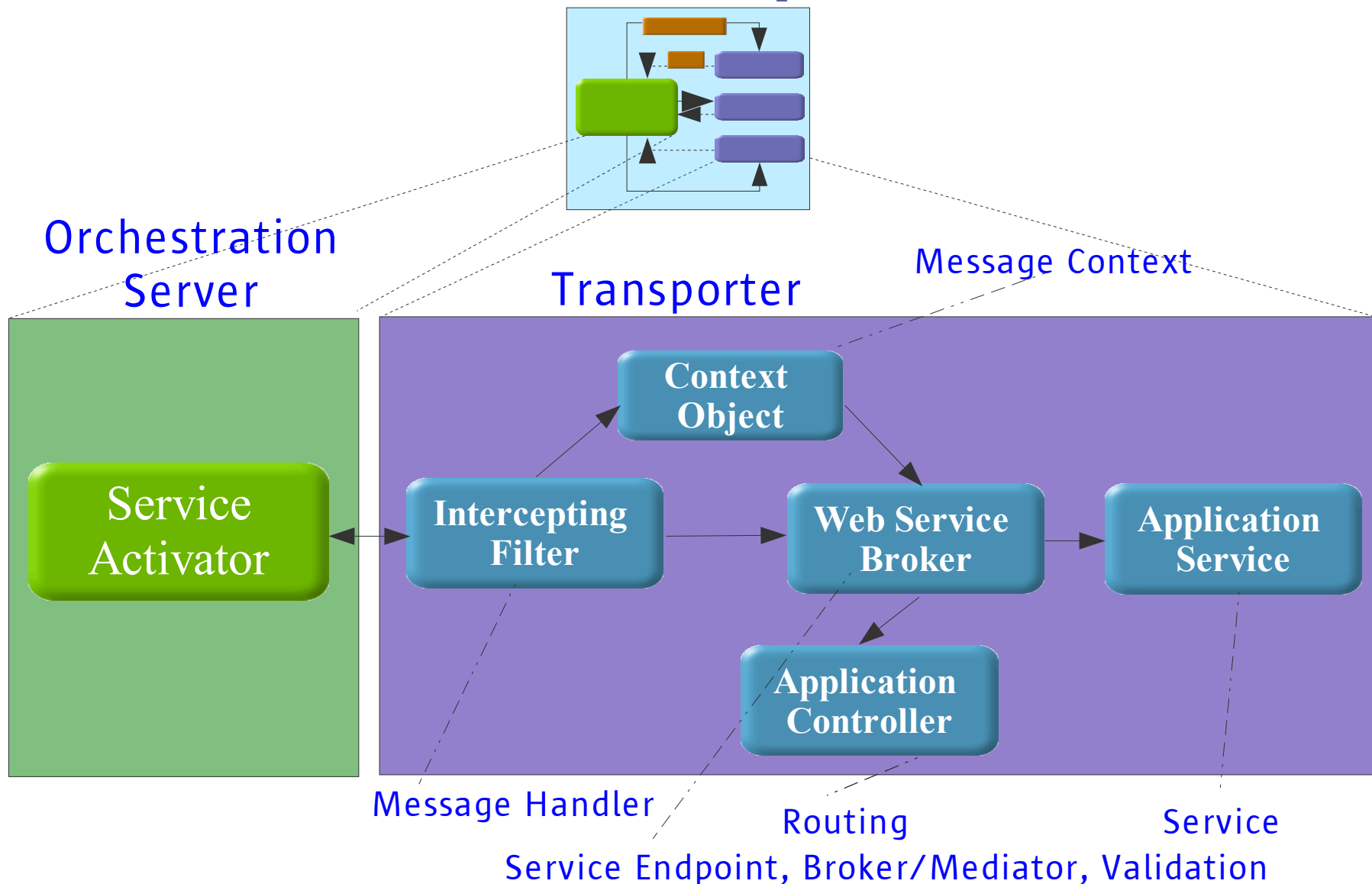


Async Web Service Orchestration

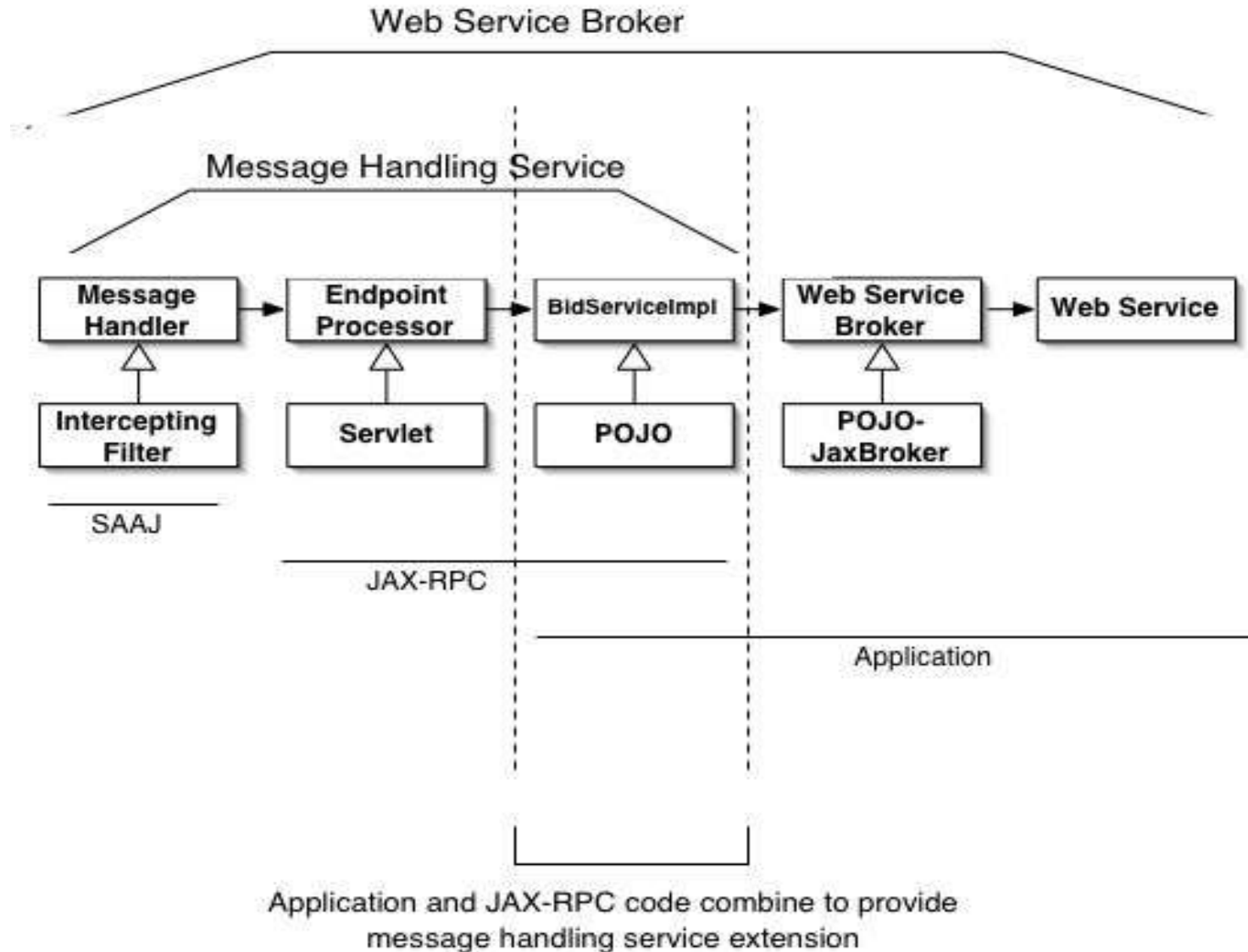
- Shipping Company contracts Transporters to ship products



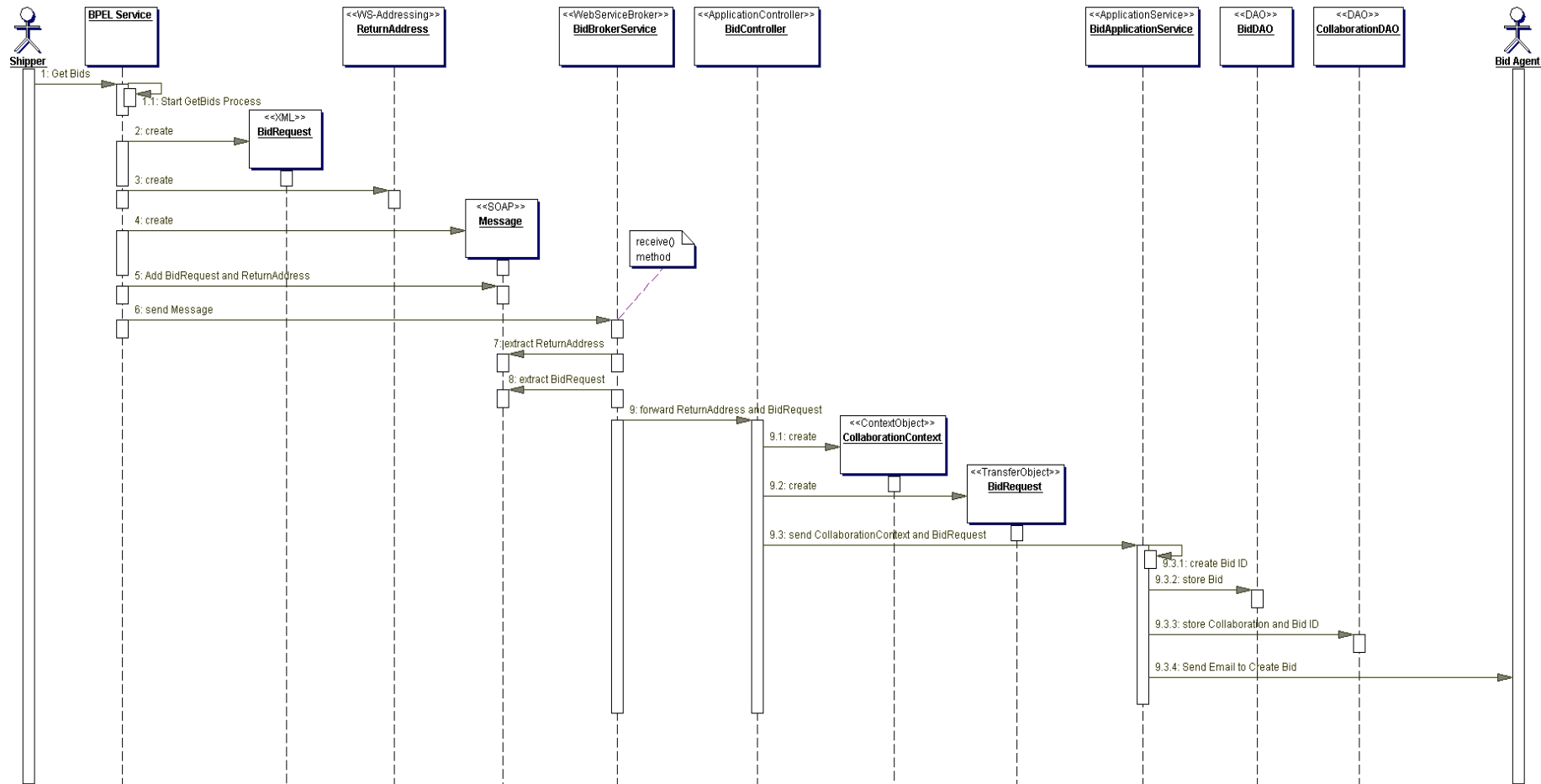
Micro Architecture composed of Patterns



Web Service Broker

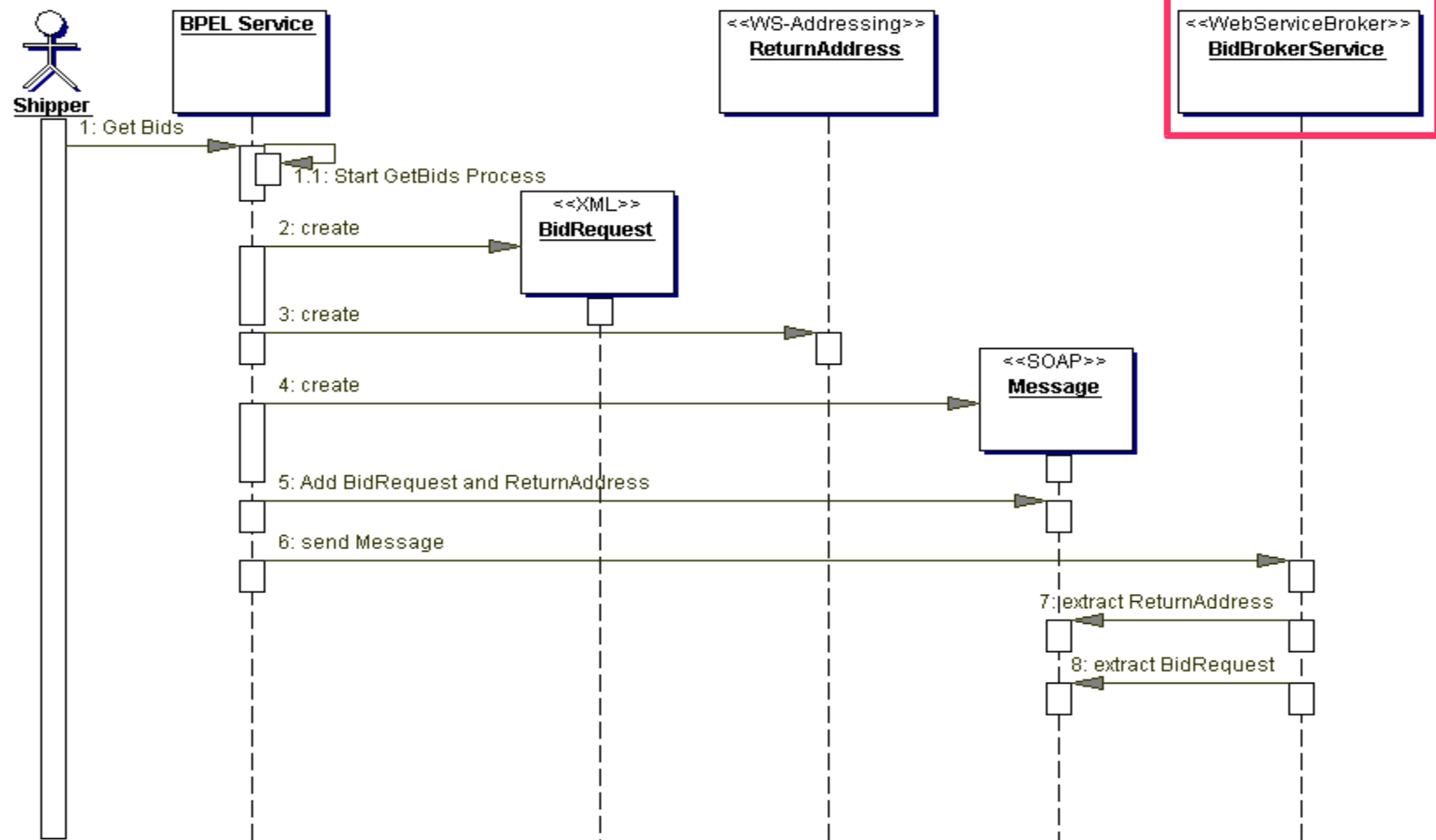


Get Bids Interaction Eye-Chart

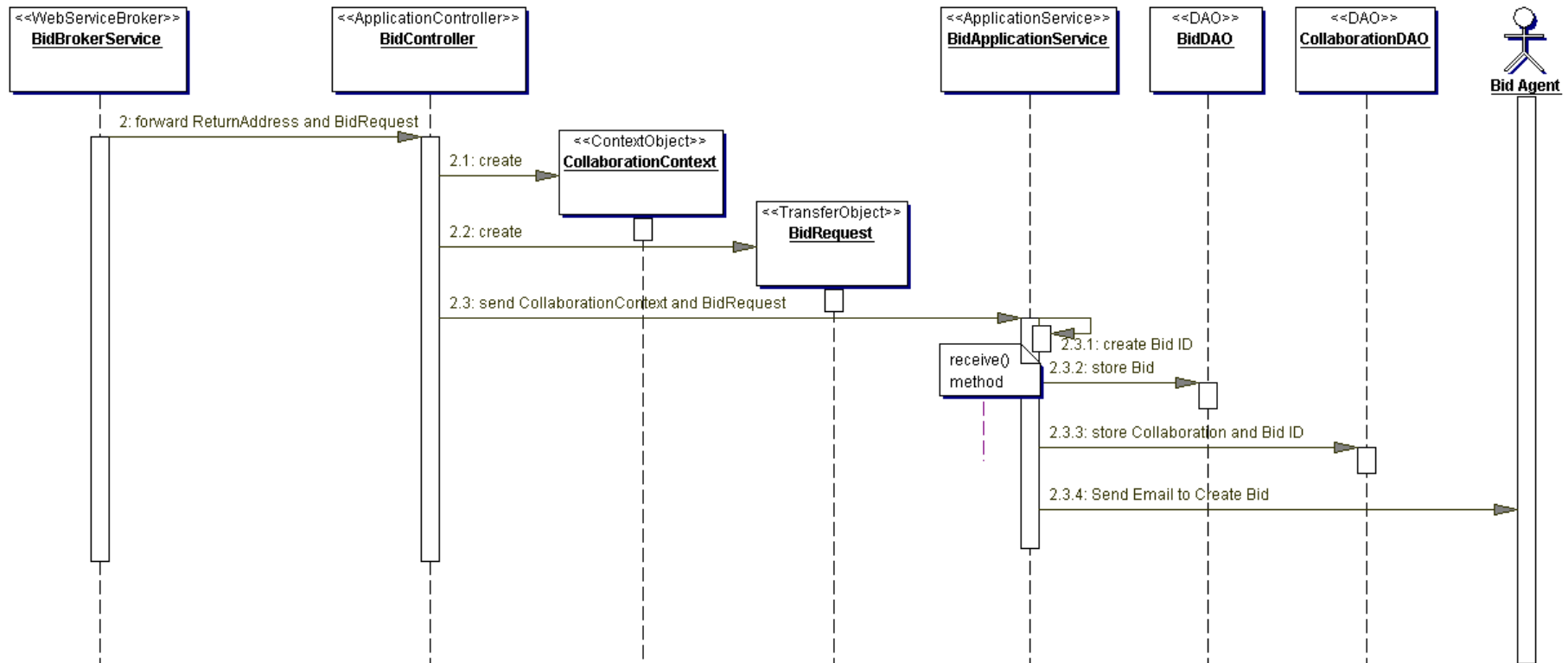


Get Bids Interaction – Part 1

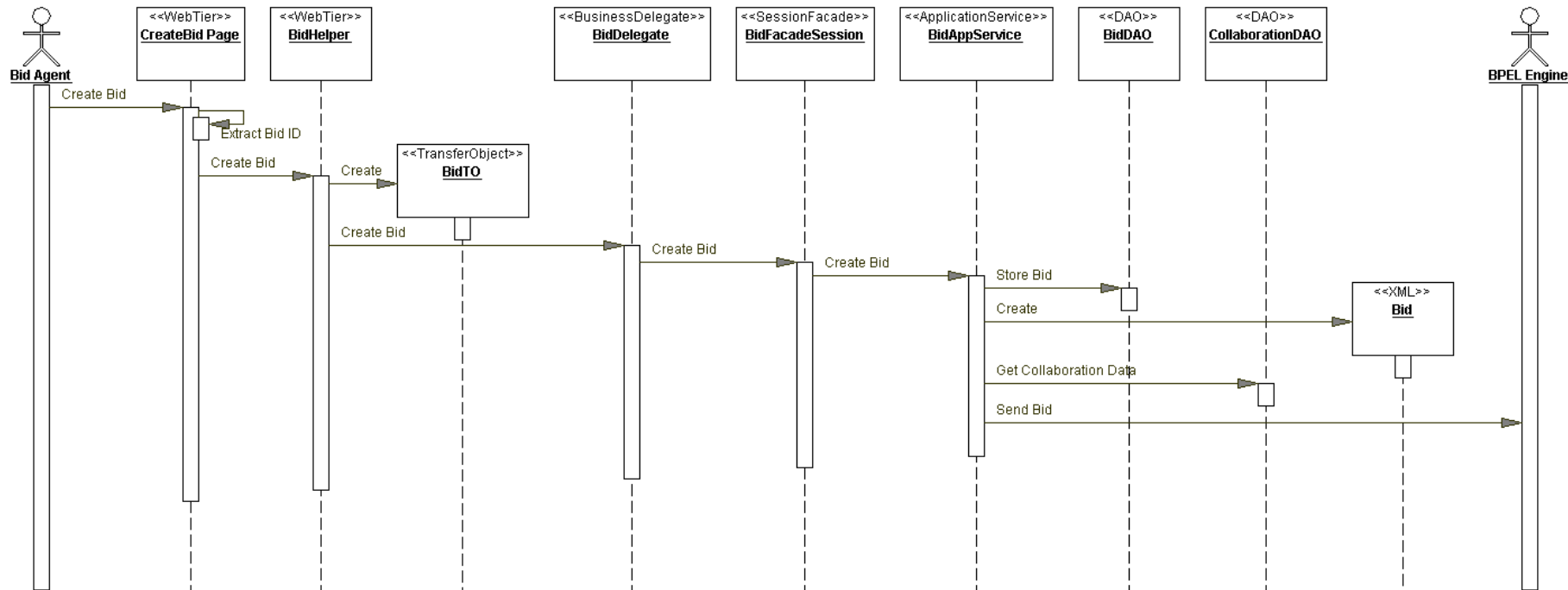
Transporter Web Service



Get Bids Interaction – Part 2

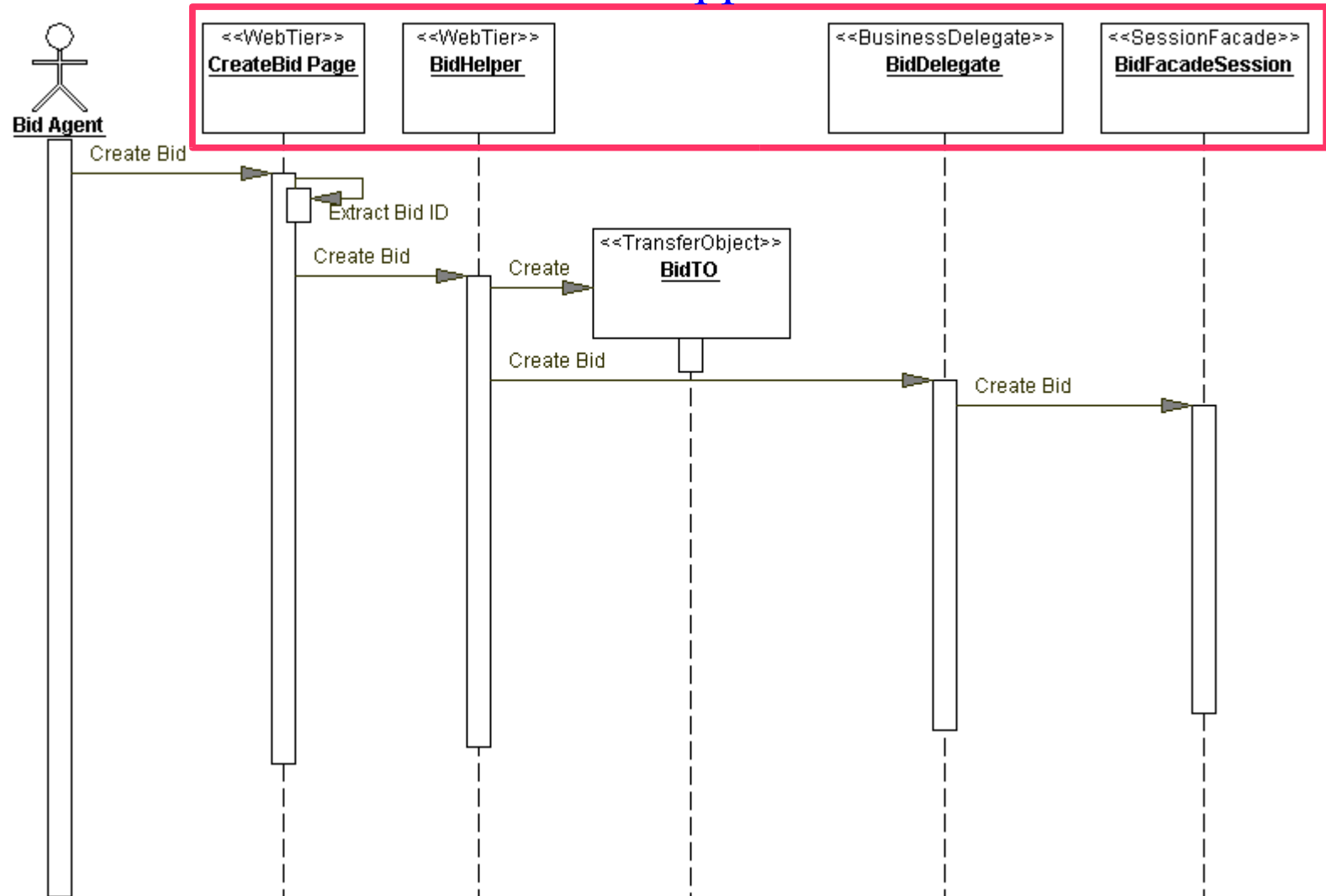


Create Bid Interaction Eye-Chart



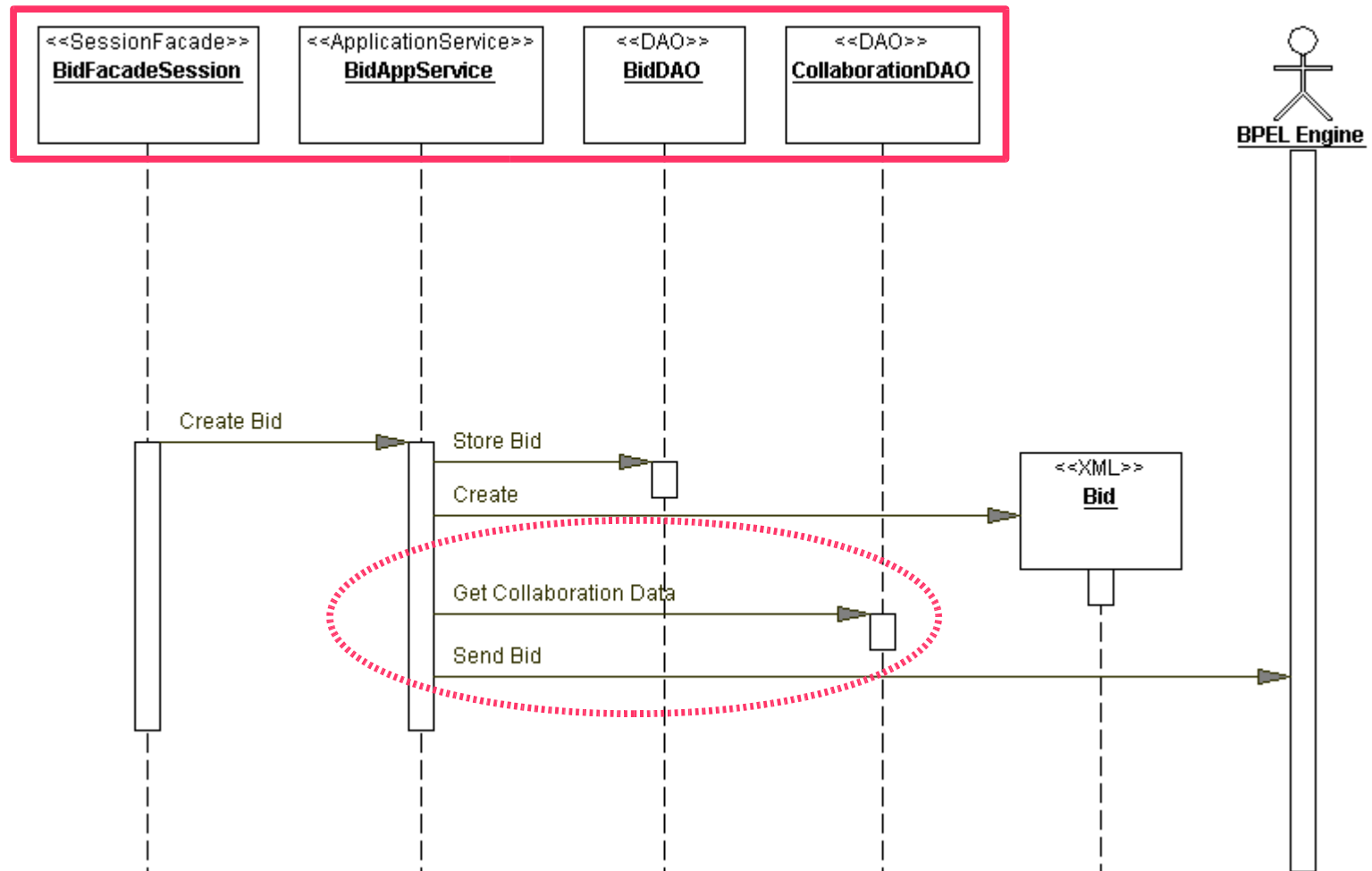
Create Bid Interaction – Part 1

Bid App



Create Bid Interaction – Part 2

Bid App





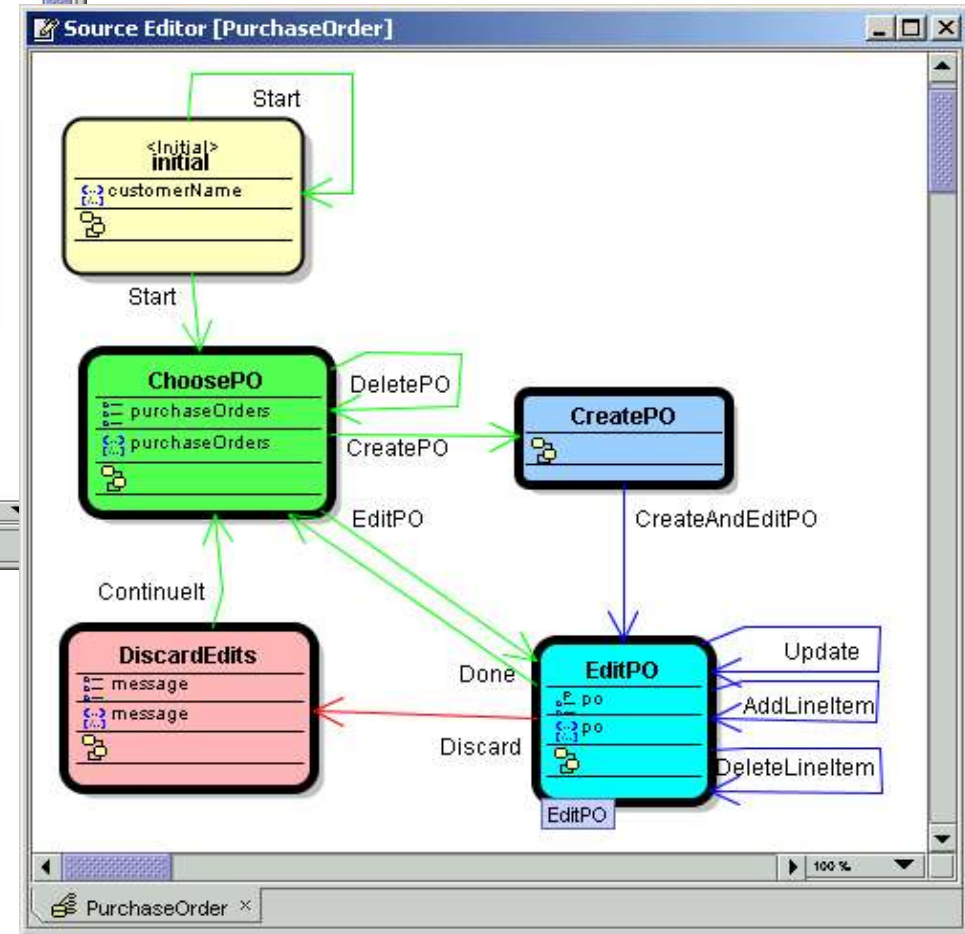
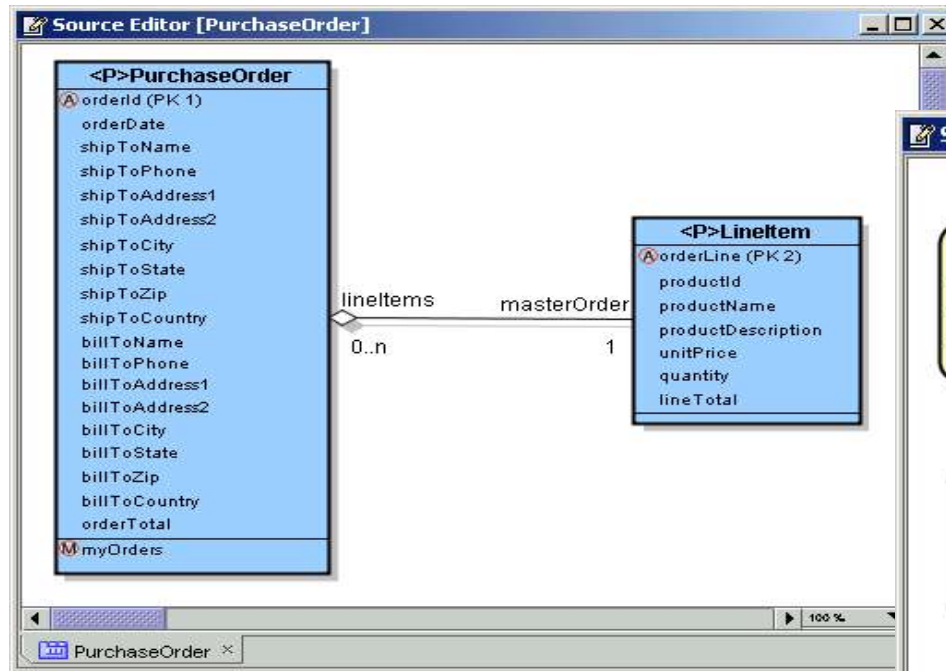
ACE: Design To Deploy Service

- Rapid intuitive design of enterprise applications
- Focus on design rather than coding
- Builds upon best practices, patterns and frameworks
- Fewer resources, faster development
- Automated deployment

DASL: Specification Language

- ACE uses a high level domain modeling language called DASL
- DASL is used to specify:
 - Business Objects, relationships
 - Core reusable business logic
 - User interaction
 - Transactions and Persistence

DASL: Graphic tools

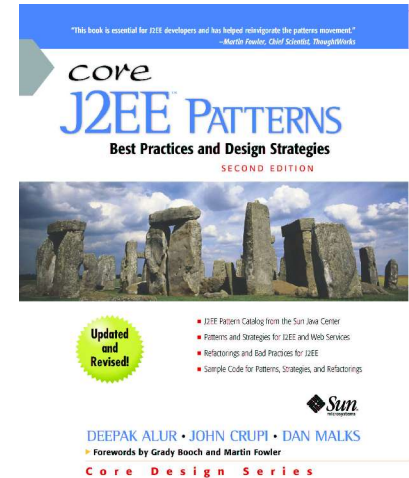


Summary

- Patterns are great! Use them effectively to improve software quality
 - Build New Architecture
 - Analyse / understand existing Architecture
 - Refactor
- Avoid re-inventing the wheel
- Promote design re-use
- Increase developer productivity, communication
- Micro Architectures leverage patterns
- Large and growing community around patterns

Stay Connected:

- Check out CJP:
 - <http://www.corej2eepatterns.com>
- Subscribe:
 - <http://archives.java.sun/j2eepatterns-interest.html>
- Write to us:
 - j2eepatterns-feedback@sun.com
- Java.Net – Patterns Community





Thanks!

deepak.alur@sun.com

January 2004

