Work groups meeting – 3

INF5040 (Open Distributed Systems)

Sabita Maharjan
sabita@simula.no

Department of Informatics
University of Oslo
September 07, 2009

Outline

- Design Patterns
- J2EE Design Patterns
  - EIS tier patterns
  - Business tier patterns
  - Presentation tier patterns

Design Patterns

- A design pattern is a solution to a common recurring problem.
- It provides cleaner development and easier maintenance for your applications.
- "Each pattern describes a **problem** that occurs **over and over** again in our environment, and then describes the core of the **solution** to that problem, in such a way that you can use this solution a **million times** over, without ever doing it the same way twice."

A Pattern Language, Christopher Alexander et al, 1977

Software Patterns History

- 1987 - Cunningham and Beck used Alexander's ideas to develop a small pattern language for Smalltalk
- 1990 - The Gang of Four (Gamma, Helm, Johnson and Vlissides) began work compiling a catalog of design patterns
- 1991 - Bruce Anderson gave the first Patterns Workshop at OOPSLA
- 1993 - Kent Beck and Grady Booch sponsored the first meeting of what is now known as the Hillside Group
- 1994 - First Pattern Languages of Programs (PLoP) conference
- 1995 - The GoF published the *Design Patterns book*

Why Use Design Patterns?

- Reuse successful designs and architectures
  - Design patterns are proven techniques made more accessible to developers of new systems
  - Leverage experience recorded in design patterns
- Help in choosing design alternatives that make a system and/or components of that system more reusable
  - Avoid alternatives that compromise reusability
- Improved documentation and maintainability
  - Specification of classes, objects, their interactions and intent
- Bottom line: get the design “right” faster

Design Patterns

A Design pattern consists of:

1. Name
   - A handle that can be used to describe a design problem, its solutions and consequences
2. The problem
   - When to apply the pattern
   - The context of the problem
   - Conditions before applying the pattern
3. The solution
   - Describes the elements that make up the design
   - Relationships, Responsibilities, Collaborations
4. The consequences
   - Results and trade-offs of applying the pattern
   - Necessary for evaluating design alternatives, costs and benefits

Describing Design Patterns

- Pattern Name and Classification
  - Essence of the pattern
- Intent
  - What does the design pattern do?
  - What is its rationale and intent?
  - What particular design issue or problem does it address?
- Motivation
  - A concrete scenario that illustrates the problem
  - Helps understand the abstract description of the problem
- Applicability
  - Where can it be used?
  - What poor designs can this pattern address?

Describing Design Patterns

- Structure
  - A graphical representation of the classes in the pattern
    - Relationships between classes and objects (not sufficient)
    - Decisions, alternatives and trade-offs are necessary for reuse
- Participants
  - Classes and or objects participating in the design pattern and their responsibilities
- Collaborations
  - How the participants collaborate to carry out their responsibilities
- Consequences
  - How does the pattern support its objectives?
  - What are results and trade-offs?

Describing Design Patterns

- Implementation
  - Pitfalls, hints, techniques to be aware of
  - Language-dependent issues
- Sample Code
  - Code fragments that illustrate how to implement the pattern in C++ /Smalltalk
- Known Uses
  - Use in real systems
- Related Patterns
  - Similar patterns and important differences
  - Patterns that can be used by or with this pattern

Categories

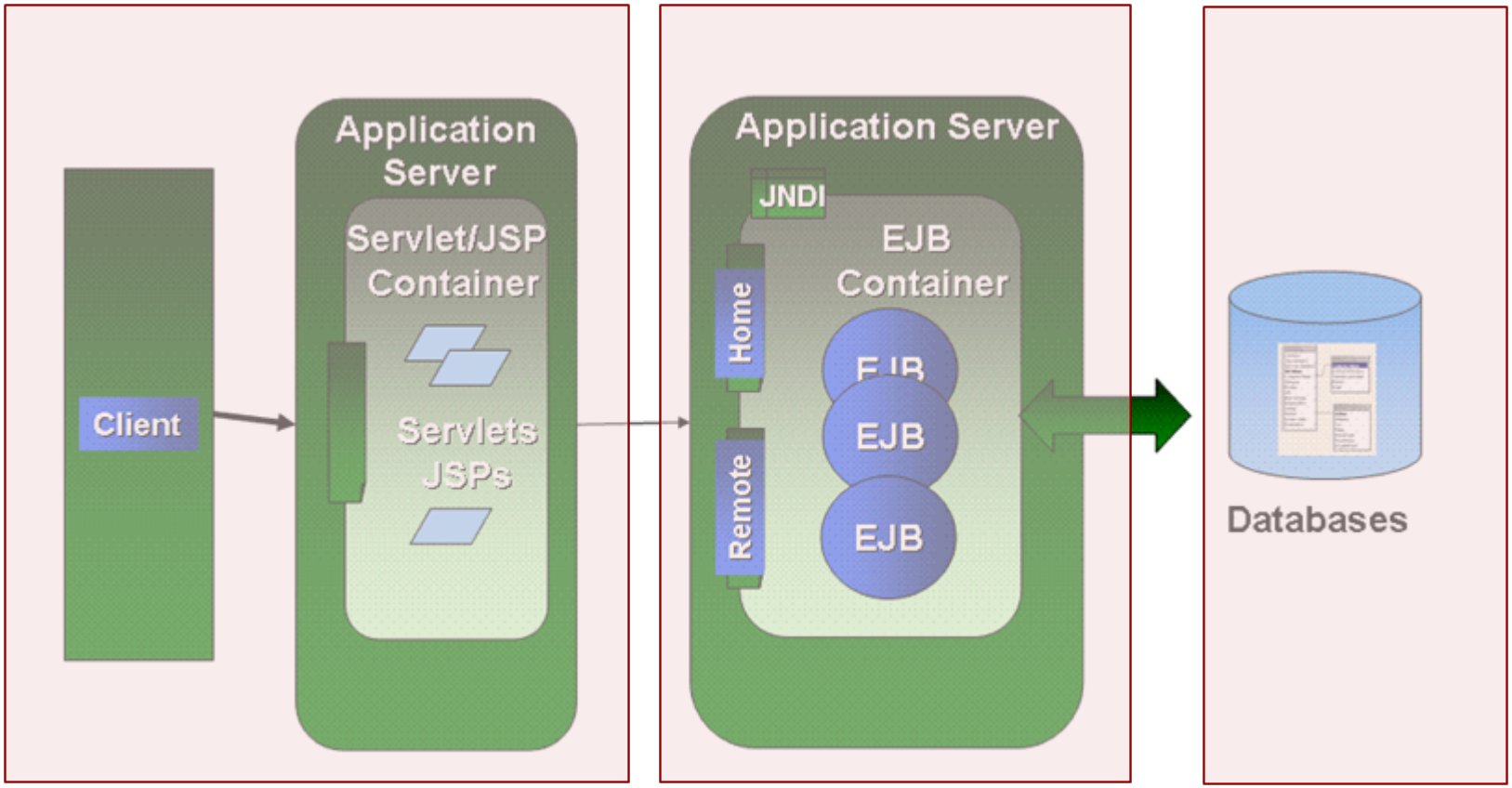
Based on Purpose

- **Creational design patterns** abstract the instantiation process.
- **Structural design patterns** are concerned with how classes and objects are composed to form larger structures
- **Behavioral design patterns** are concerned with algorithms and the assignment responsibilities between objects

J2EE Design Patterns

- A J2EE design pattern is a pattern that utilizes J2EE technology to solve a recurring problem.
- Patterns are typically classified as (logical):
  - Presentation tier
  - Business tier
  - Integration (EIS) tier

Pattern Categories – J2EE



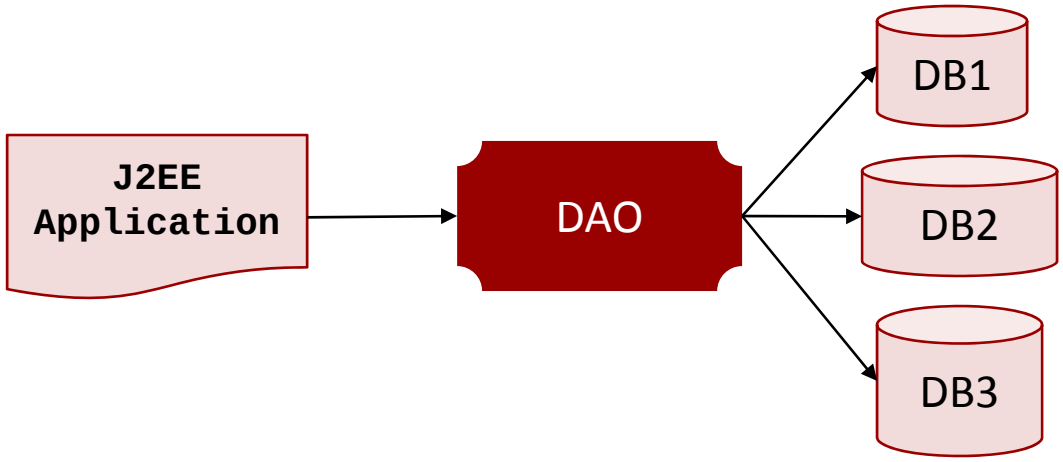
EIS Patterns: Data Access Object (DAO)

- Problem
  - Most J2EE applications need to use enterprise or business data from a persistent data store.
  - Data, however, can reside in many different kinds of repositories, from relational databases to mainframe or legacy systems.
  - Mixing application logic with persistence logic introduces makes maintenance a nightmare for an application.

DAO Pattern

Solution

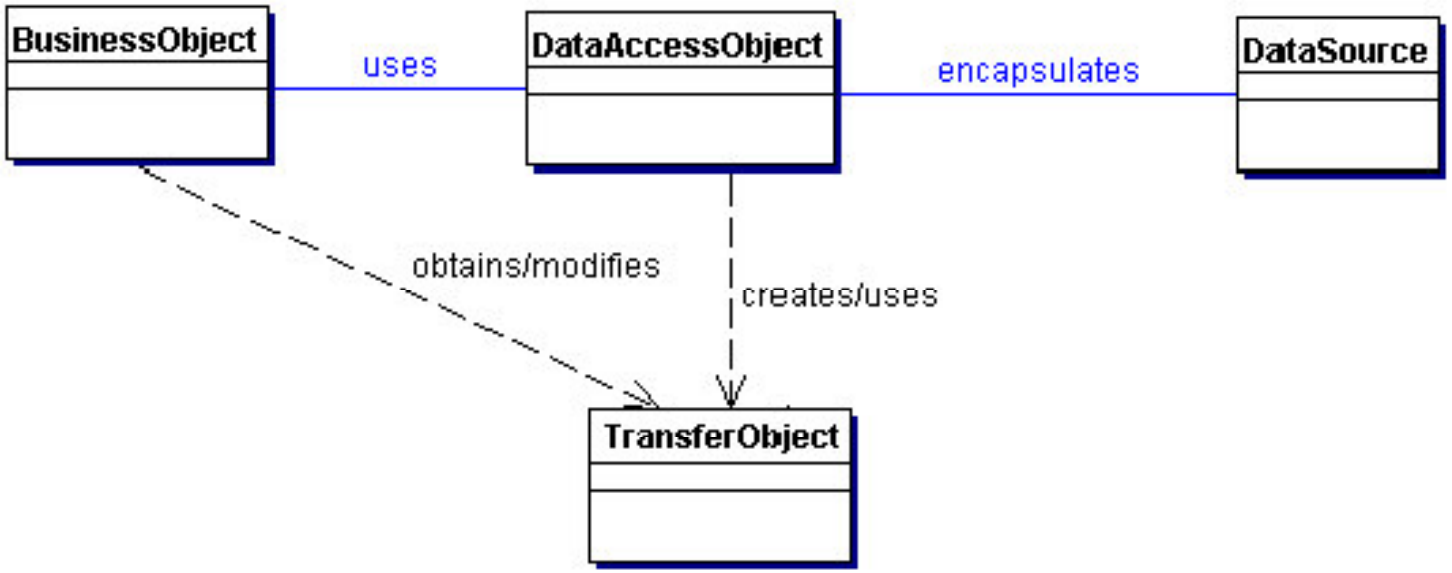
- Use a Data Access Object (DAO) to abstract and encapsulate all access to the data source.
- The DAO manages the connection with the data source to obtain and store data



DAO Pattern

Structure

- Essentially, the DAO acts as an adapter between the component and the data source.
- The DAO completely hides the data source implementation details from its clients.



DAO Pattern

- Advantages
  - Abstracts implementation details
    - Data source independent
  - Enables easier migration
    - Encapsulates proprietary APIs
  - Centralizes all data access into a separate layer
    - Reduces code complexity
    - Application is easier to manage and maintain

Business Tier Patterns

- Data Transfer Object
- Service Locator
- Session Façade
- Business Delegate

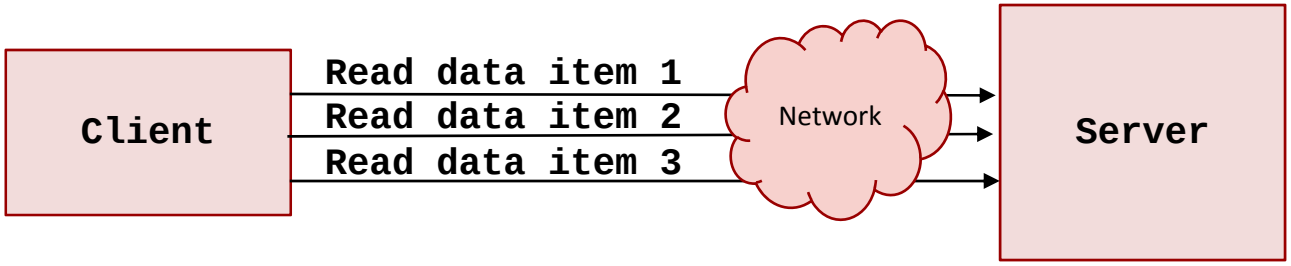
Data Transfer Object(DTO) Pattern

- A Data Transfer Object is a data envelope used to transfer groups of related attributes between application tiers

DTO

Problem:

- Each call to an EJB is potentially a remote call with network overhead
- Client usually requires more than one attribute of an entity
- Accessing each attribute individually increases network traffic which degrades performance

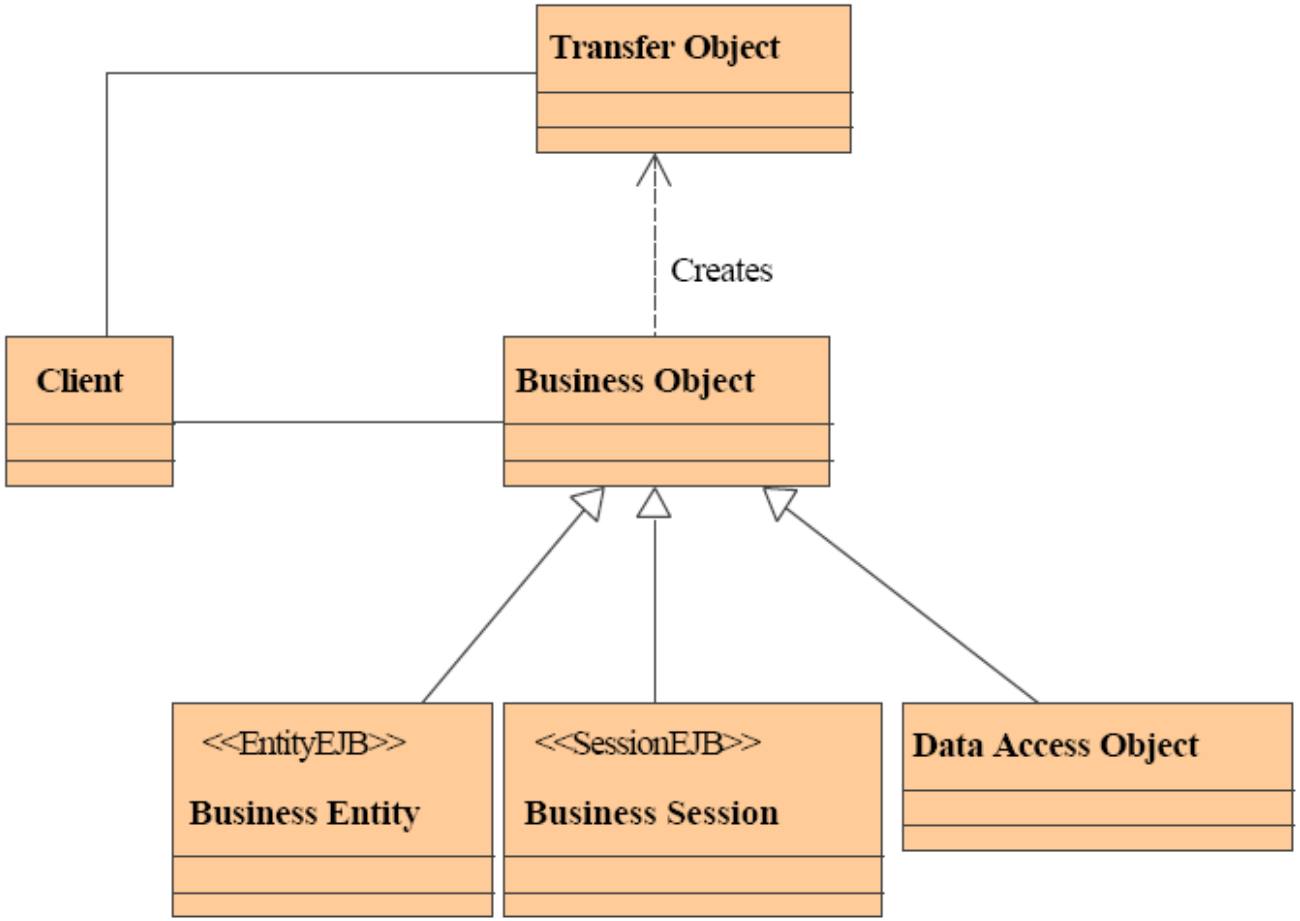


DTO

Solution

- Encapsulates related business data
- Single method call is used to send and retrieve the Transfer Object
- Passed by value to client via RMI

DTO Relationship



DTO

Adavantages

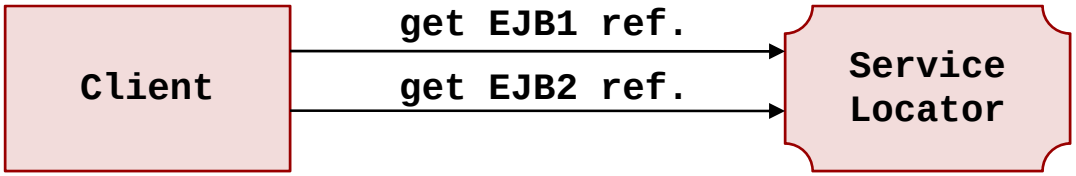
- Improves performance
  - Fewer remote calls
  - Reduced network traffic
- Can access arbitrary sets of data specific to client requirements

Business Tier Patterns

- Data Transfer Object
- Service Locator
- Session Façade
- Business Delegate

Service Locator Pattern

- Class that abstracts details of looking up and preparing a service
  - EJBs, JMS components, data sources, etc.
- Useful for reducing code complexity of service clients
- Provides a single point of control for Service Access



Service Locator

Advantages

- Encapsulates complexity of lookup and creation process
- Uniform service access point for clients
- Improves network performance
  - Lookup calls are aggregated on the server
- Improved client performance through caching
  - Reduces redundant lookups and object creation

Business Tier Patterns

- Data Transfer Object
- Service Locator
- Session Façade
- Business Delegate

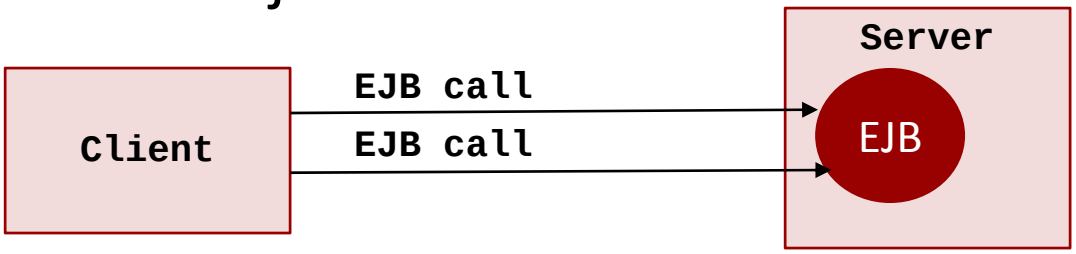
Session Façade

Context

- Enterprise beans encapsulate business logic and business data and **expose** their interfaces, and thus the complexity of the distributed services, to the client tier

Problem

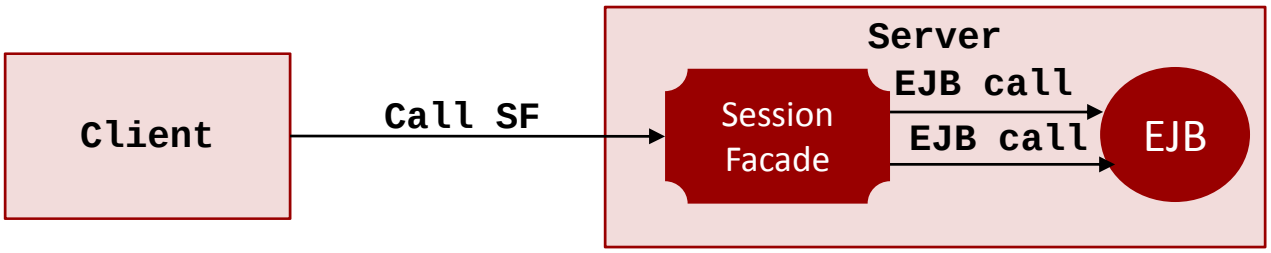
- Too many method invocations between client and server, leading to network performance problems
- Lack of a uniform client access strategy, exposing business objects to misuse



Session Façade

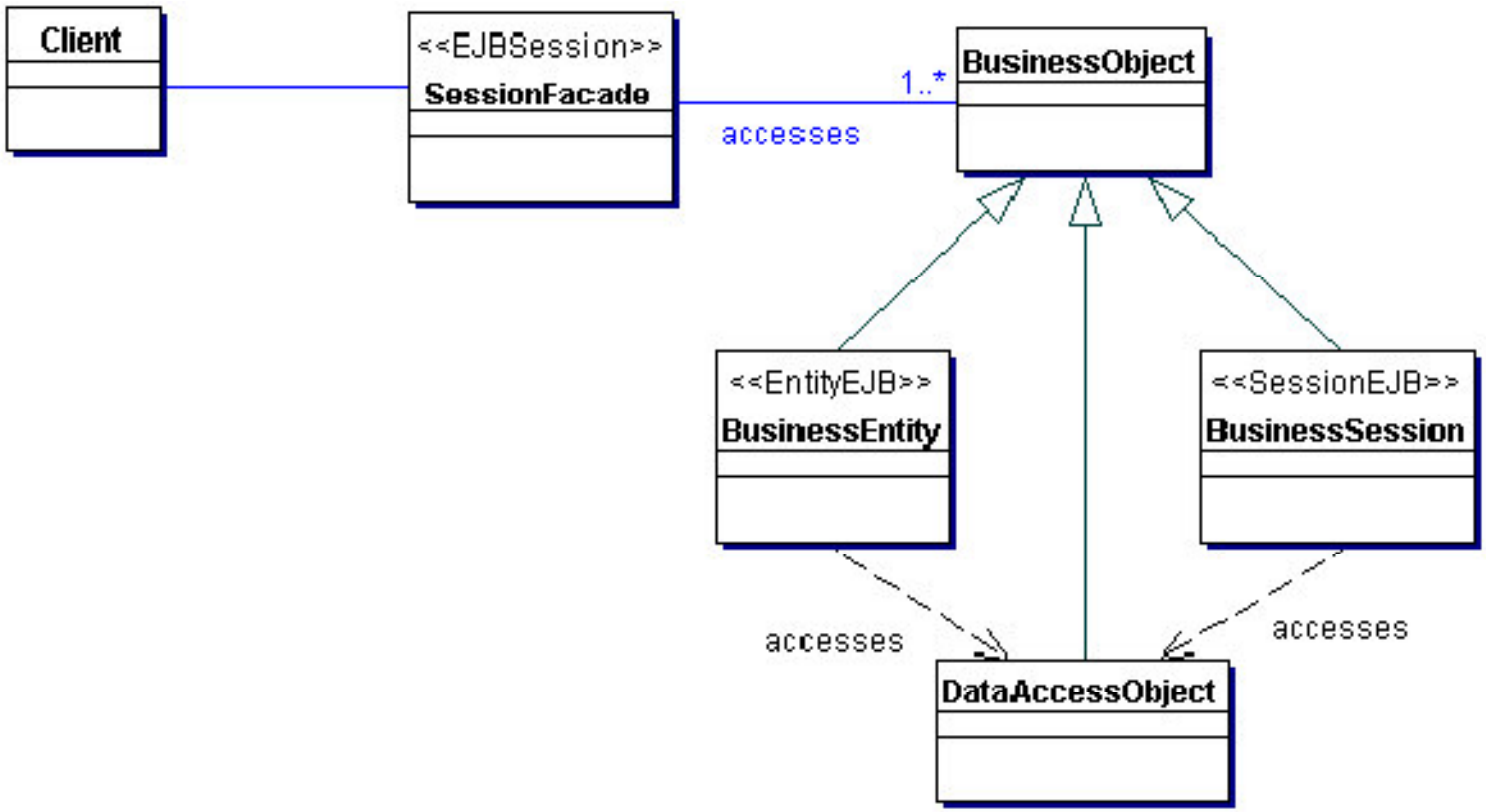
Solution

- Use a session bean as a facade to encapsulate the complexity of interactions between the business objects participating in a workflow.
- The Session Facade manages the business objects, and provides a uniform coarse-grained service access layer to clients



Session Façade

- Structure



Session Façade

Advantages

- Improves performance by reducing network overhead
- Introduces control layer for business tier
  - Centralizes security management and transaction control
  - Reduces coupling of tiers
- Exposes uniform interface for interaction
- Coarse grained access to business services
  - Exposes fewer remote interfaces to clients

Business Tier Patterns

- Data Transfer Object
- Service Locator
- Session Façade
- Business Delegate

Business Delegate Pattern

Context

- In distributed applications, lookup and exception handling for remote business components can be complex.

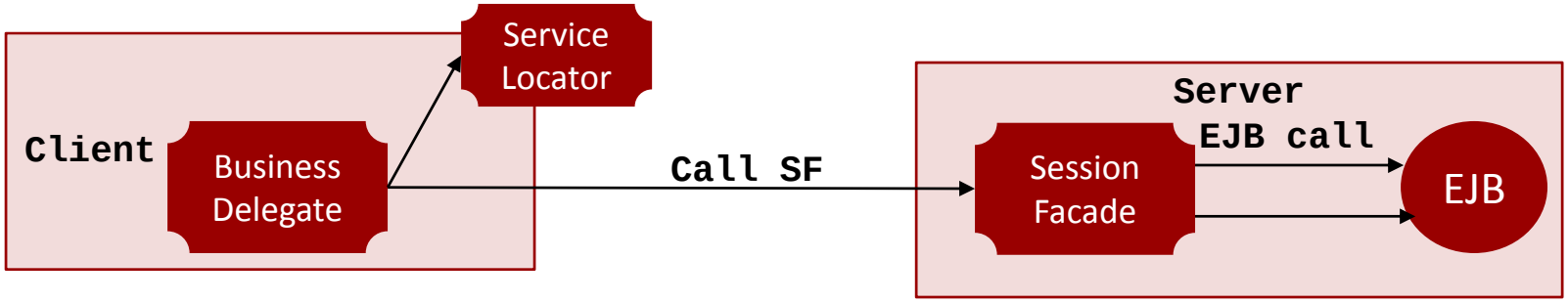
Problem

- Presentation-tier components interface directly with business services
  - Exposes details of business service implementation, thus exposed to complexity of network invocations
- Detrimental impact on performance as presentation-tier components make many invocations over network

Business Delegate

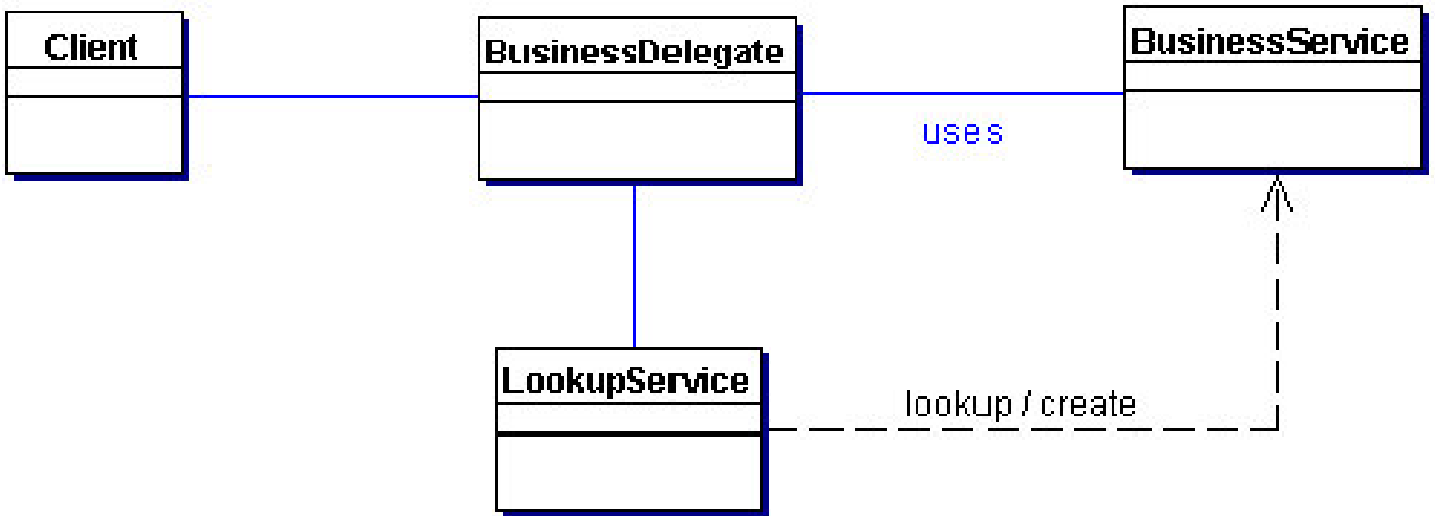
Solution

- Use an intermediate class “Business Delegate” to decouple business components from the code that uses them.
- The Business Delegate hides the underlying implementation details of the business service, such as lookup and access details of remote access



Business Delegate

Structure



Business Delegate

- Advantages
  - Reduces coupling between tiers
  - Translates business services exceptions
    - e.g. Network and infrastructure exceptions translated to business exceptions
  - Provides a uniform interface to the business tier
  - Can improve performance through caching

Web Tier Patterns

- Model View Controller (MVC) Pattern
  - Most Web-tier application frameworks use some variation of the MVC design pattern.

MVC Pattern

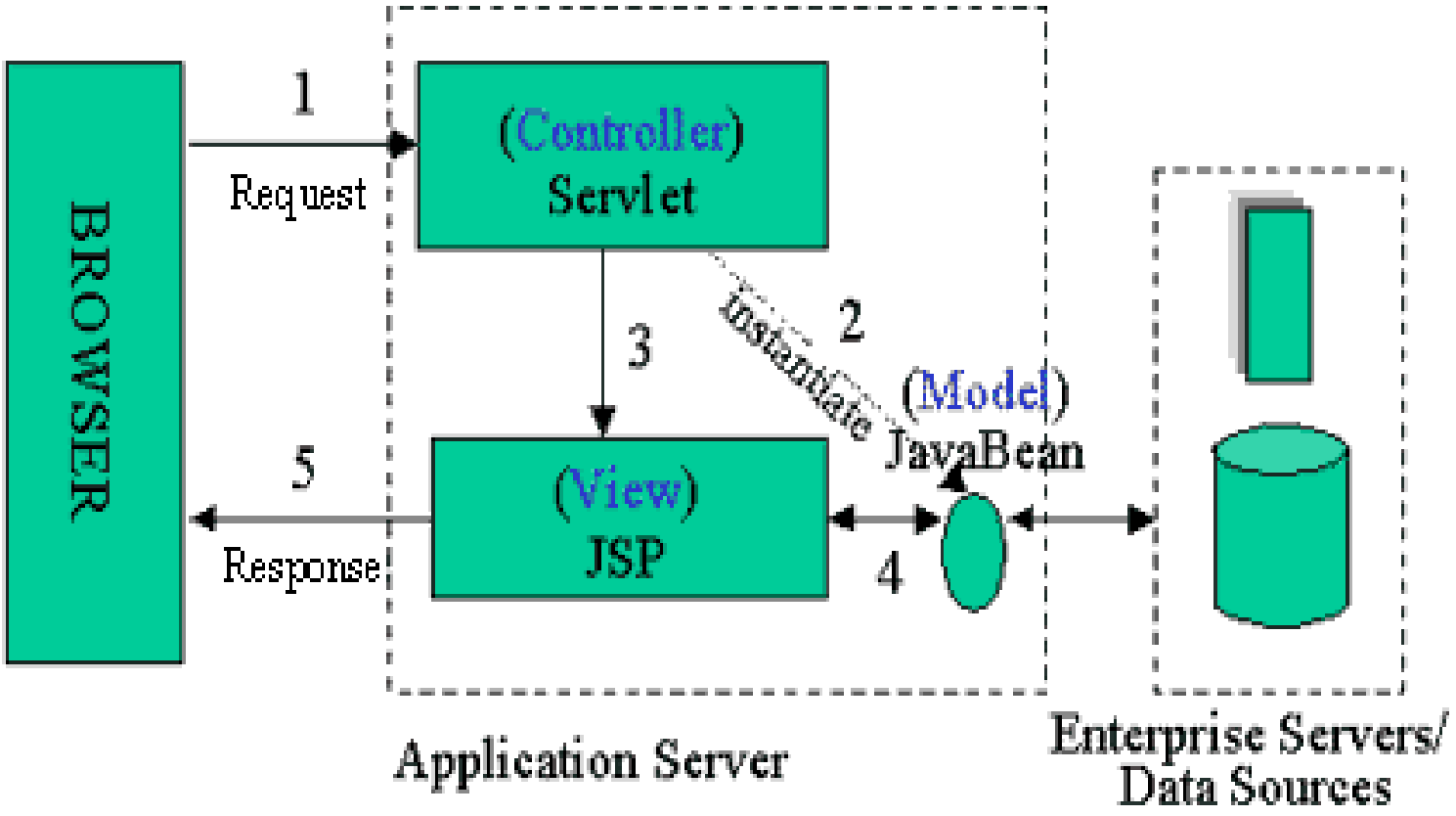
- **Name** (essence of the pattern)
  - Model View Controller MVC
- **Context** (where does this problem occur)
  - MVC is an architectural pattern that is used when developing interactive applications such as a shopping cart on the Internet.
- **Problem** (definition of the recurring difficulty)
  - User interfaces change often, especially on the internet where look-and-feel is a competitive issue. Also, the same information is presented in different ways. The core business logic and data is stable.

MVC Pattern

Solution

- Use the software engineering principle of "separation of concerns" to divide the application into three areas:
  - **Model** encapsulates the core data and functionality
  - **View** encapsulates the presentation of the data there can be many views of the common data
  - **Controller** accepts input from the user and makes request from the model for the data to produce a new view.

MVC Pattern


BROWSER
1
Request
(Controller)
Servlet
3
2
instantiate
(Model)
JavaBean
5
Response
(View)
JSP
4
Application Server
Enterprise Servers/
Data Sources

References

- EJB Design Patterns by Floyd Marinescu – Wiley
- Core J2EE Patterns: Best Practices and Design Strategies, Prentice Hall
- J2EE Design Patterns Catalog - http://java.sun.com/blueprints/patterns/catalog.html
- Core J2EE Patterns (online book) - http://java.sun.com/blueprints/corej2eepatterns/Patterns/