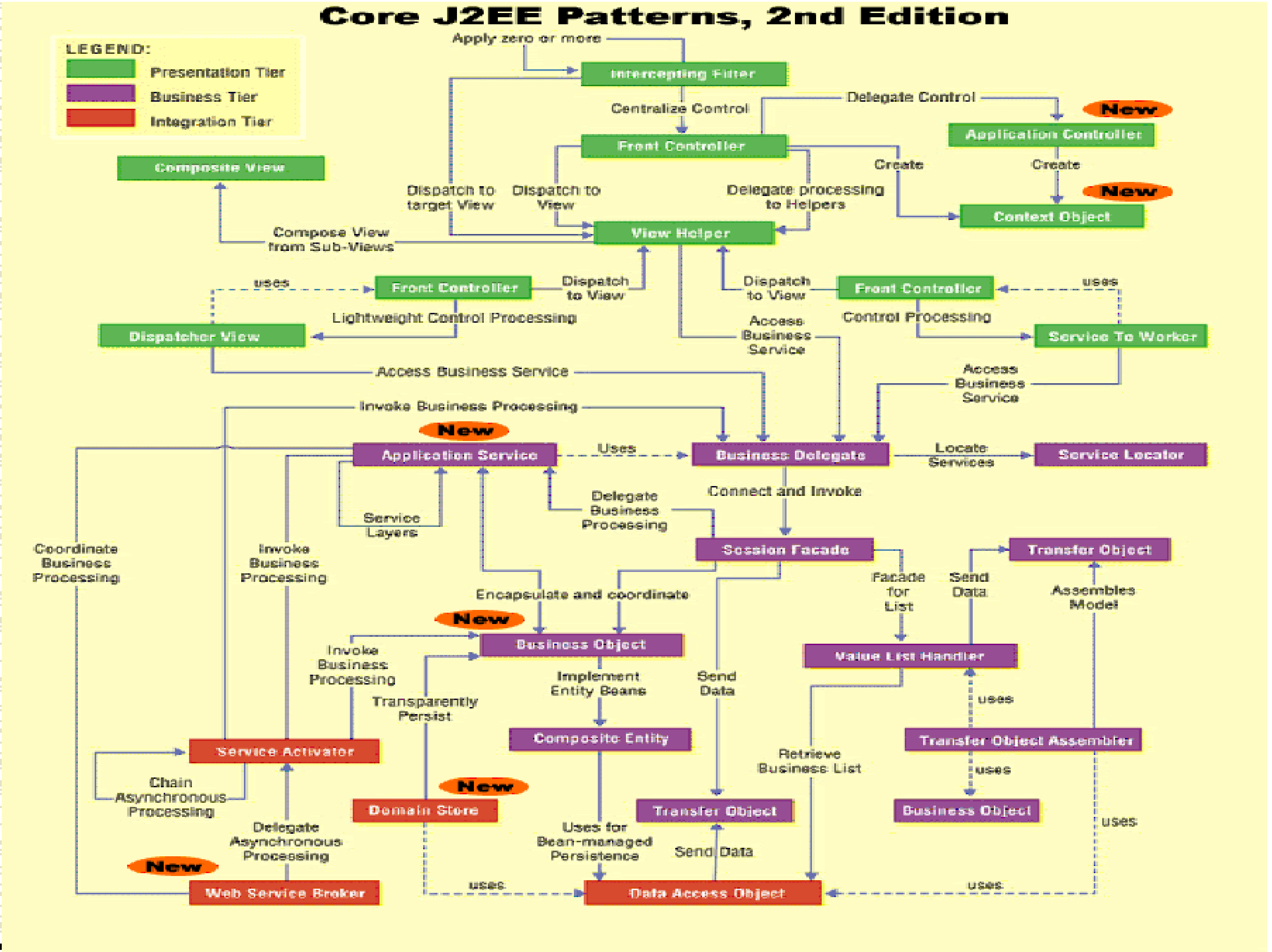
# J2EE Integration Tier Patterns



# J2EE Integration Tier Patterns

## 1 Data Access Object

Problem

*You want to encapsulate data access and manipulation in a separate layer.*

Forces

You want to implement data access mechanisms to access and manipulate data in a persistent storage.

You want to decouple the persistent storage implementation from the rest of your application.

You want to provide a uniform data access API for a persistent mechanism to various types of data sources, such as RDBMS, LDAP, OODB, XML repositories, flat files, and so on.

You want to organize data access logic and encapsulate proprietary features to facilitate maintainability and portability.

# J2EE Integration Tier Patterns

## 1 Data Access Object

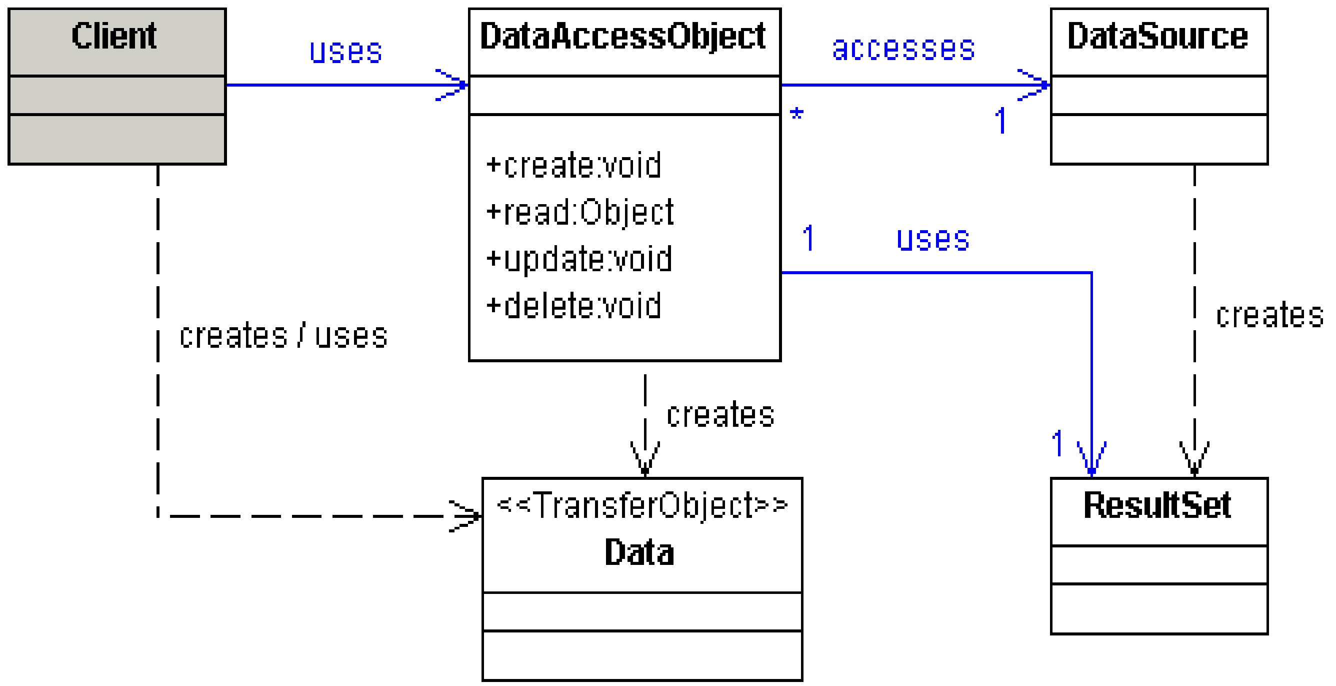
Solution

Use a *Data Access Object* to abstract and encapsulate all access to the persistent store.

The *Data Access Object* manages the connection with the data source to obtain and store data.

## 1 Data Access Object

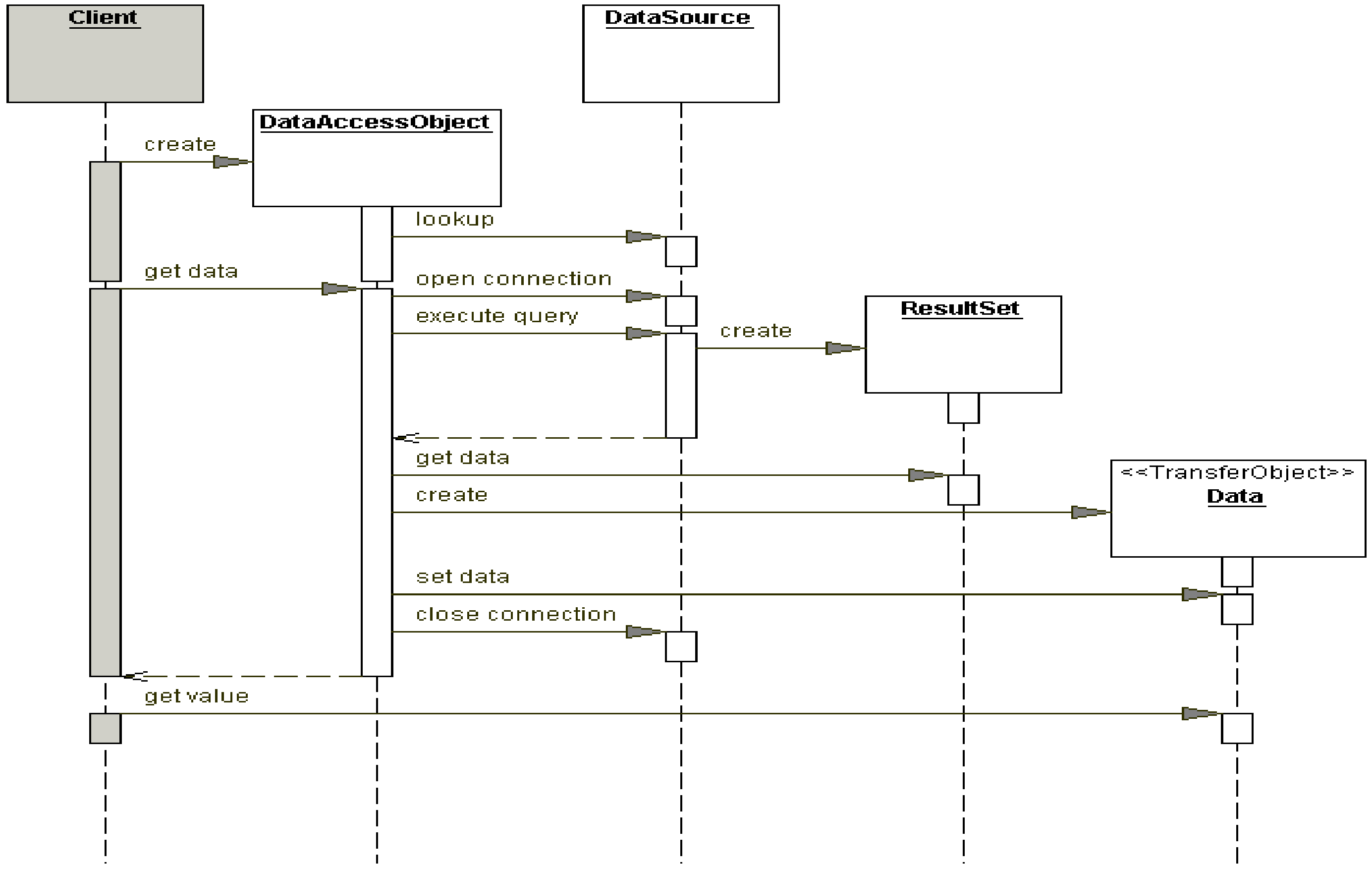
### Class Diagram :



# J2EE Integration Tier Patterns

## 1. Data Access Object

### Sequence Diagram :



## J2EE Integration Tier Patterns

### 1. Data Access Object

***Strategies***

*Custom Data Access Object Strategy*

*Data Access Object Factory Strategies*

*Transfer Object Collection Strategy*

*Cached RowSet Strategy*

*Read Only RowSet Strategy*

*RowSet Wrapper List Strategy*

***Consequences***

*Centralizes control with loosely coupled handlers*

*Enables transparency*

*Provides object-oriented view and encapsulates database schemas*

*Enables easier migration*

*Reduces code complexity in clients*

*Organizes all data access code into a separate layer*

*Adds extra layer*

## 2. Service Activator

*Problem*

*You want to invoke services asynchronously.*

*Forces*

*You want to invoke business services, POJOs, or EJB components in an asynchronous manner.*

*You want to integrate publish/subscribe and point-to-point messaging to enable asynchronous processing services.*

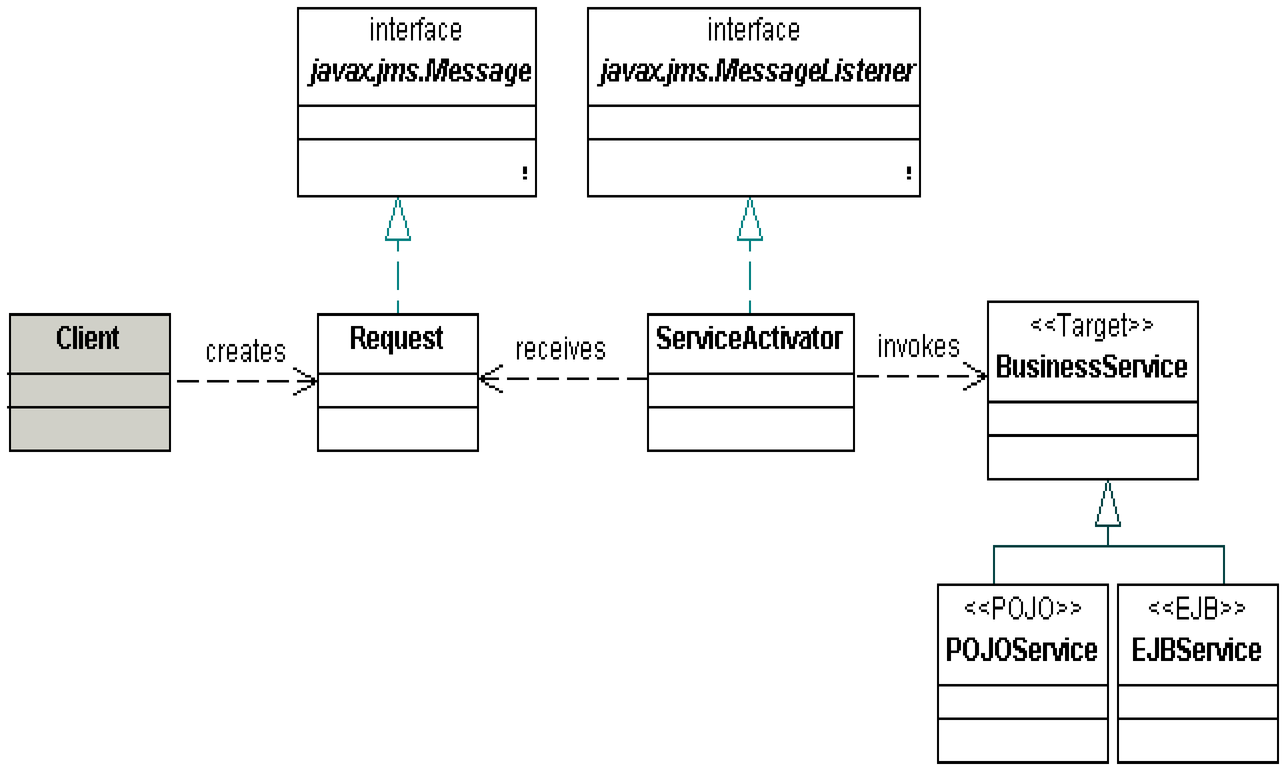
*You want to perform a business task that is logically composed of several business tasks.*

*Solution*

*Use a Service Activator to receive asynchronous requests and invoke one or more business services.*

## 2. Service Activator

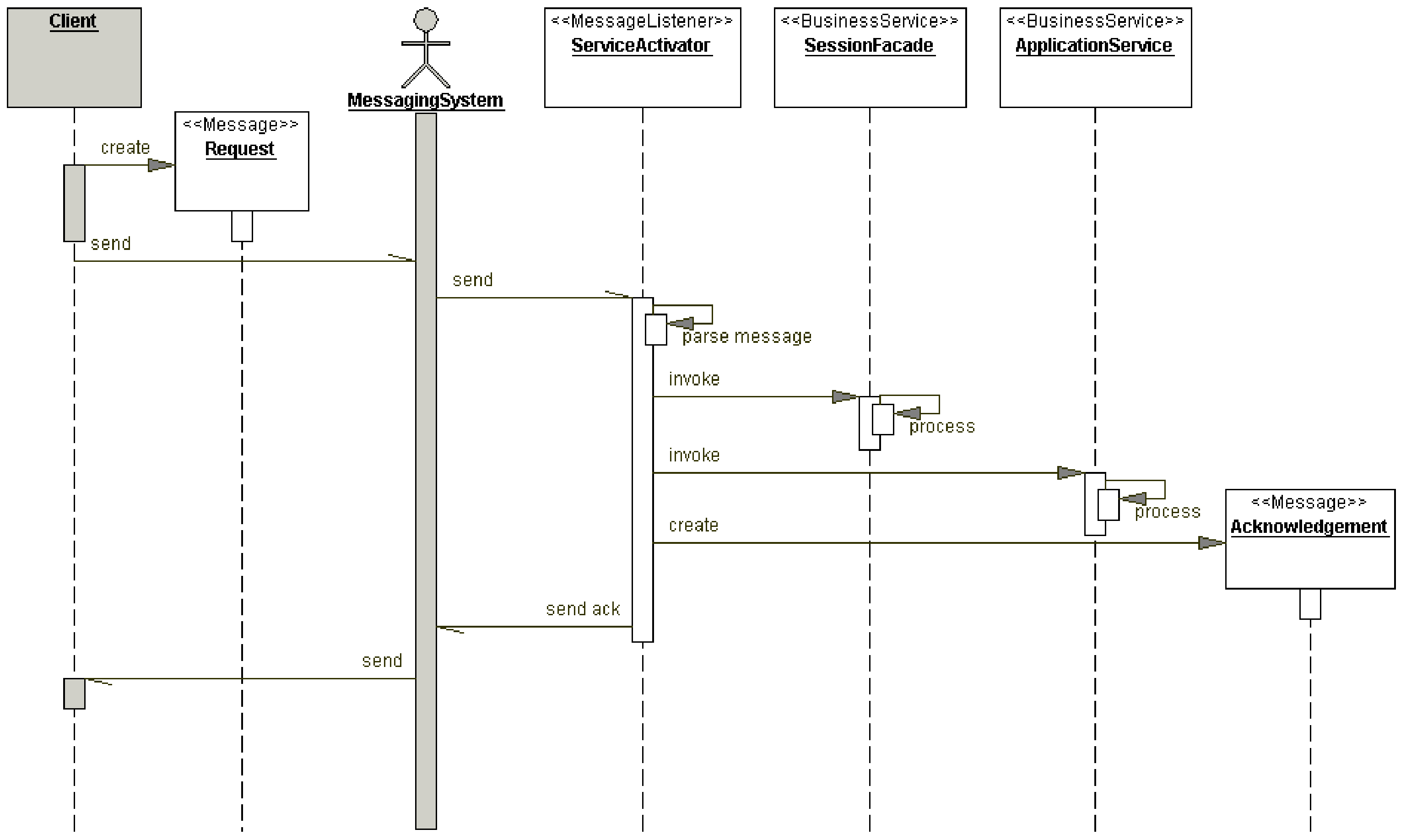
Class Diagram :



# J2EE Integration Tier Patterns

## 2. Service Activator

### Sequence Diagram :



## 2. Service Activator

***Strategies***

*POJO Service Activator Strategy*

*MDB Service Activator Strategy*

*Service Activator Aggregator Strategy*

*Response Strategies*

*Database Response Strategy*

*Email Response Strategy*

*JMS Message Response Strategy*

***Consequences***

*Integrates JMS into enterprise applications*

*Provides asynchronous processing for any business-tier component*

*Enables standalone JMS listener*

# J2EE Integration Tier Patterns

## 3. 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.*

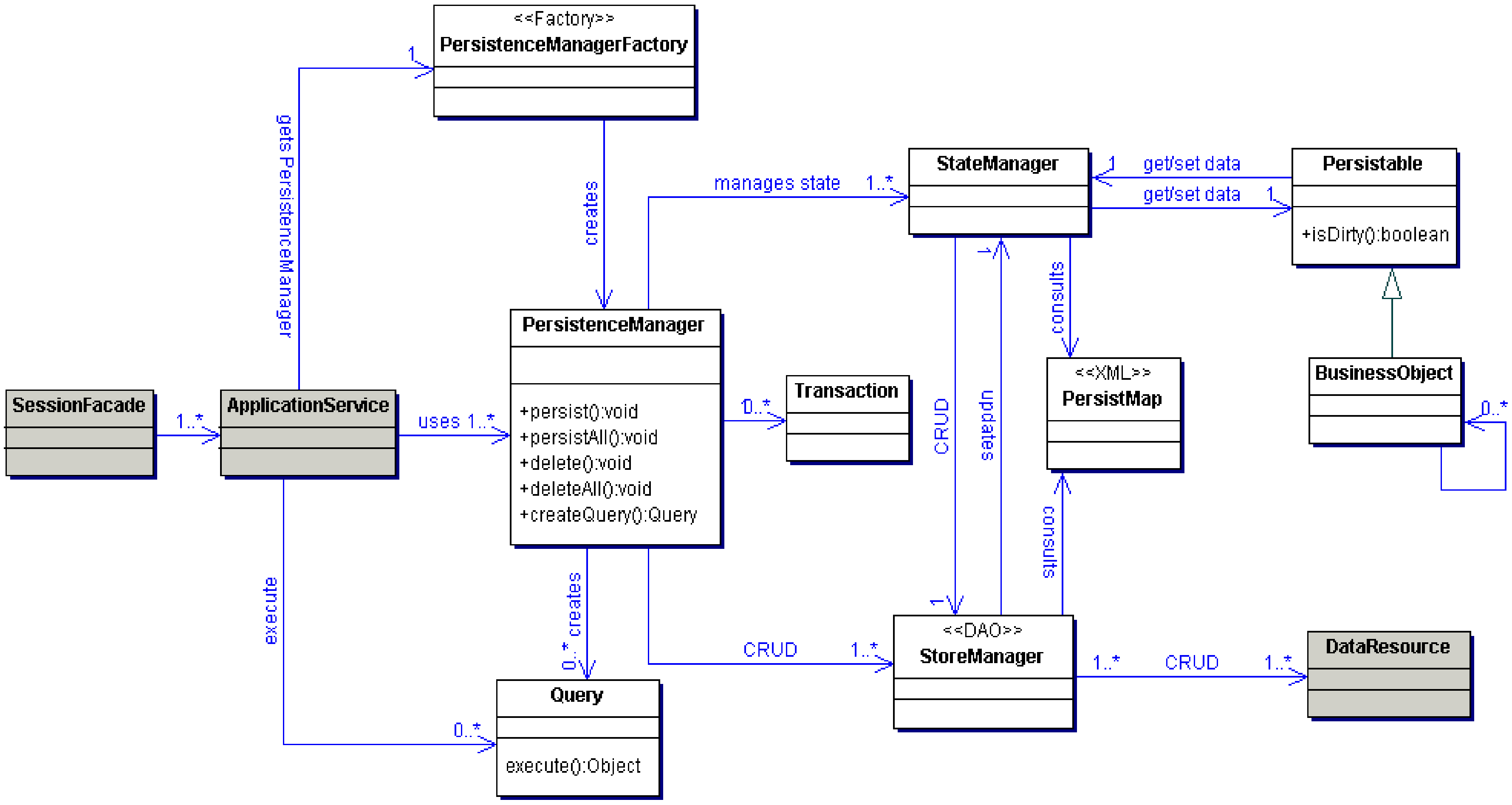
Solution

Use a *Domain Store* to transparently persist an object model. Unlike J2EE's container-managed persistence and bean-managed persistence, which include persistence support code in the object model, *Domain Store*'s persistence mechanism is separate from the object model.

# J2EE Integration Tier Patterns

## 3. Domain Store

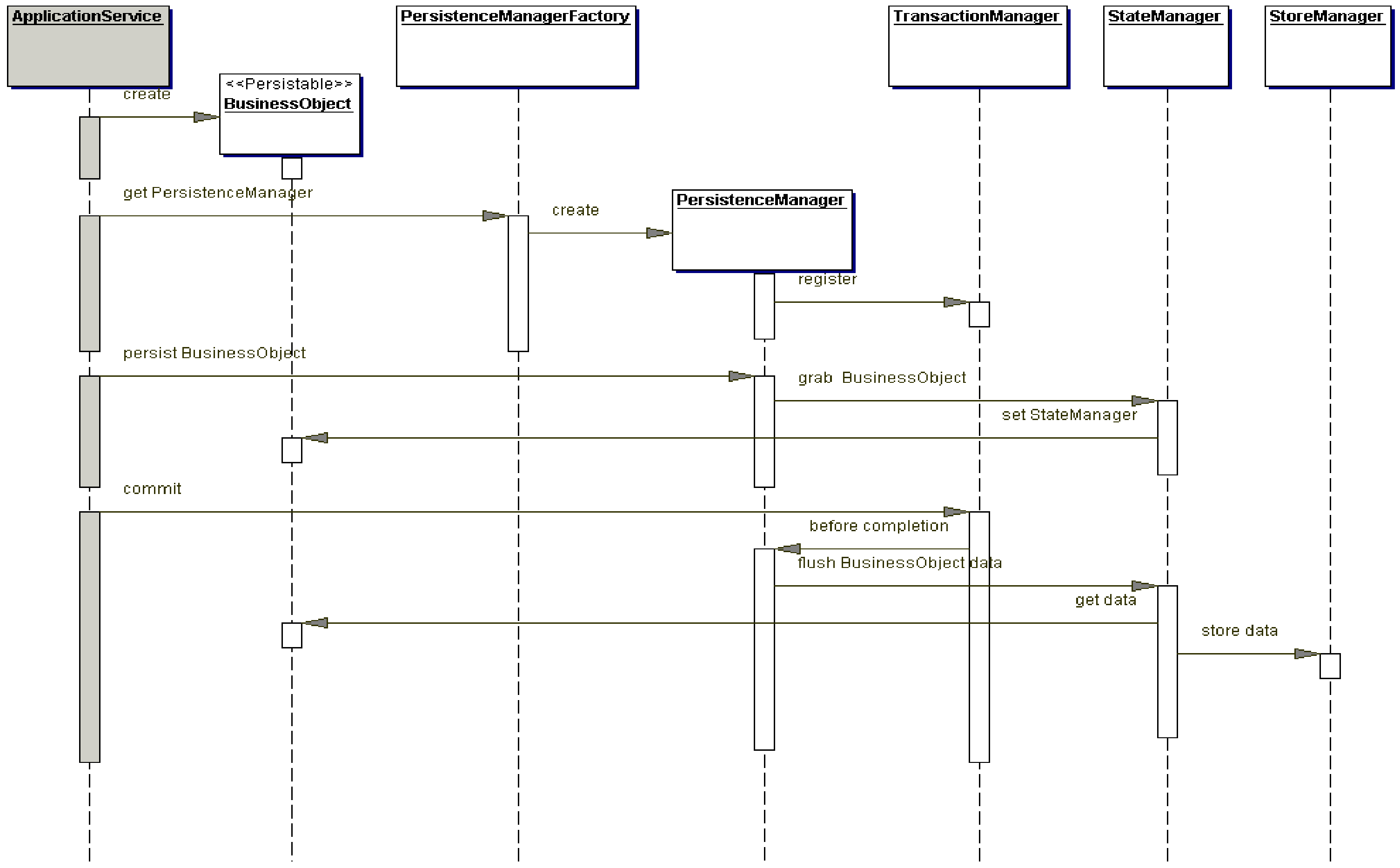
### Class Diagram :



# J2EE Integration Tier Patterns

## 3. Domain Store

### Sequence Diagram :



J2EE Integration Tier Patterns

## 3. Domain Store

***Strategies***

*Custom Persistence Strategy*

*JDO Strategy*

*ORM Strategy*

***Consequences***

*Creating a custom persistence framework is a complex task*

*Multi-layer object tree loading and storing requires optimization techniques*

*Improves understanding of persistence frameworks*

*A full-blown persistence framework might be overkill for a small object model*

*Improves testability of your persistent object model*

*Separates business object model from persistence logic*

## 4. 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, based on your business requirements and system resource usage.

Your services must be exposed using open standards to enable integration of heterogeneous applications.

You want to bridge the gap between business requirements and existing service capabilities

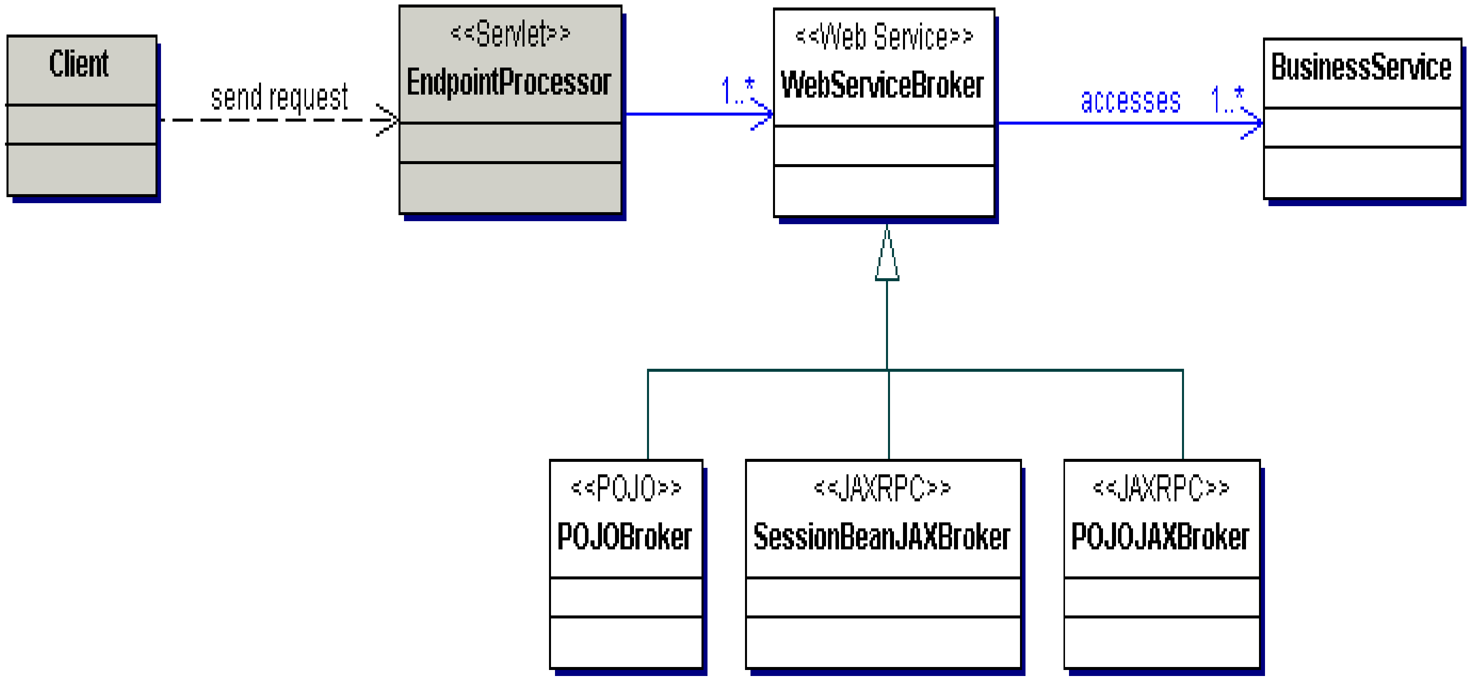
Solution

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

J2EE Integration Tier Patterns

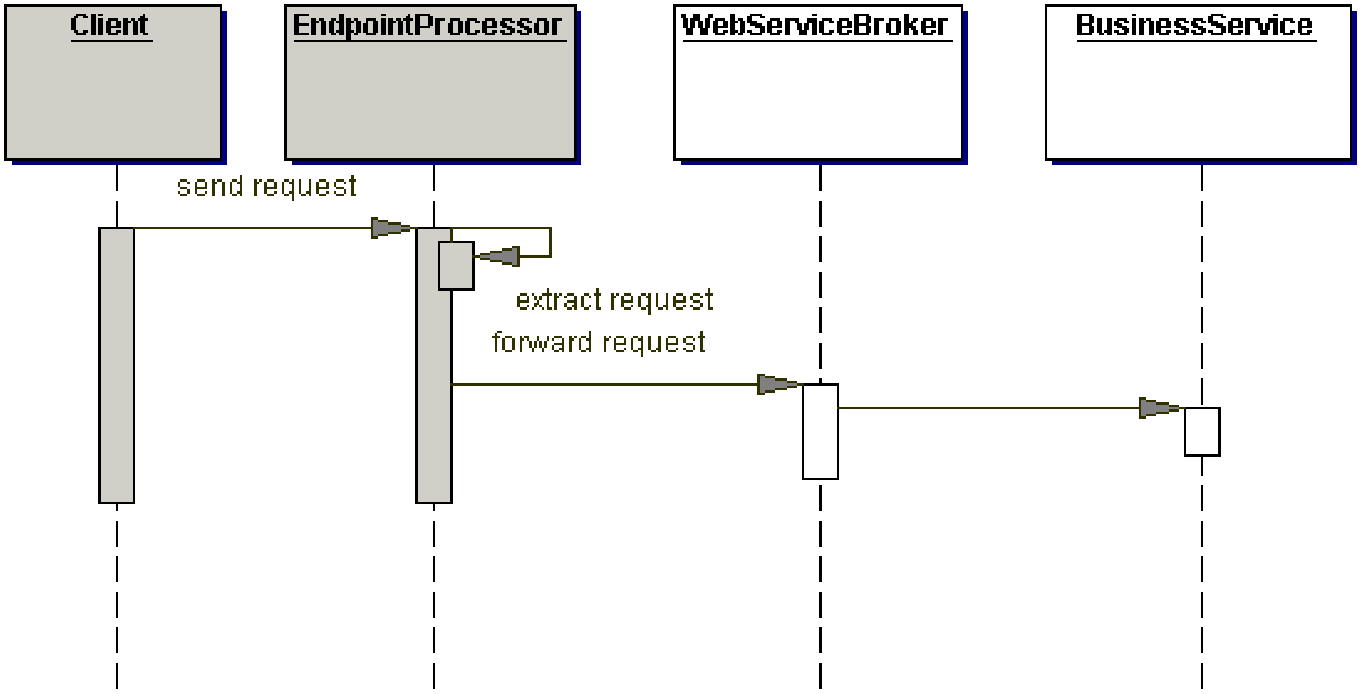
## 4. Web Service Broker

Class Diagram :



## 4. Web Service Broker

### Sequence Diagram :



# J2EE Integration Tier Patterns

## 4. Web Service Broker

***Strategies***

*Custom XML Messaging Strategy*

*Java Binder Strategy*

*JAX-RPC Strategy*

***Consequences***

*Introduces a layer between client and service*

*Existing remote Session Façades (341) need be refactored to support local access*

*Network performance may be impacted due to web protocols*