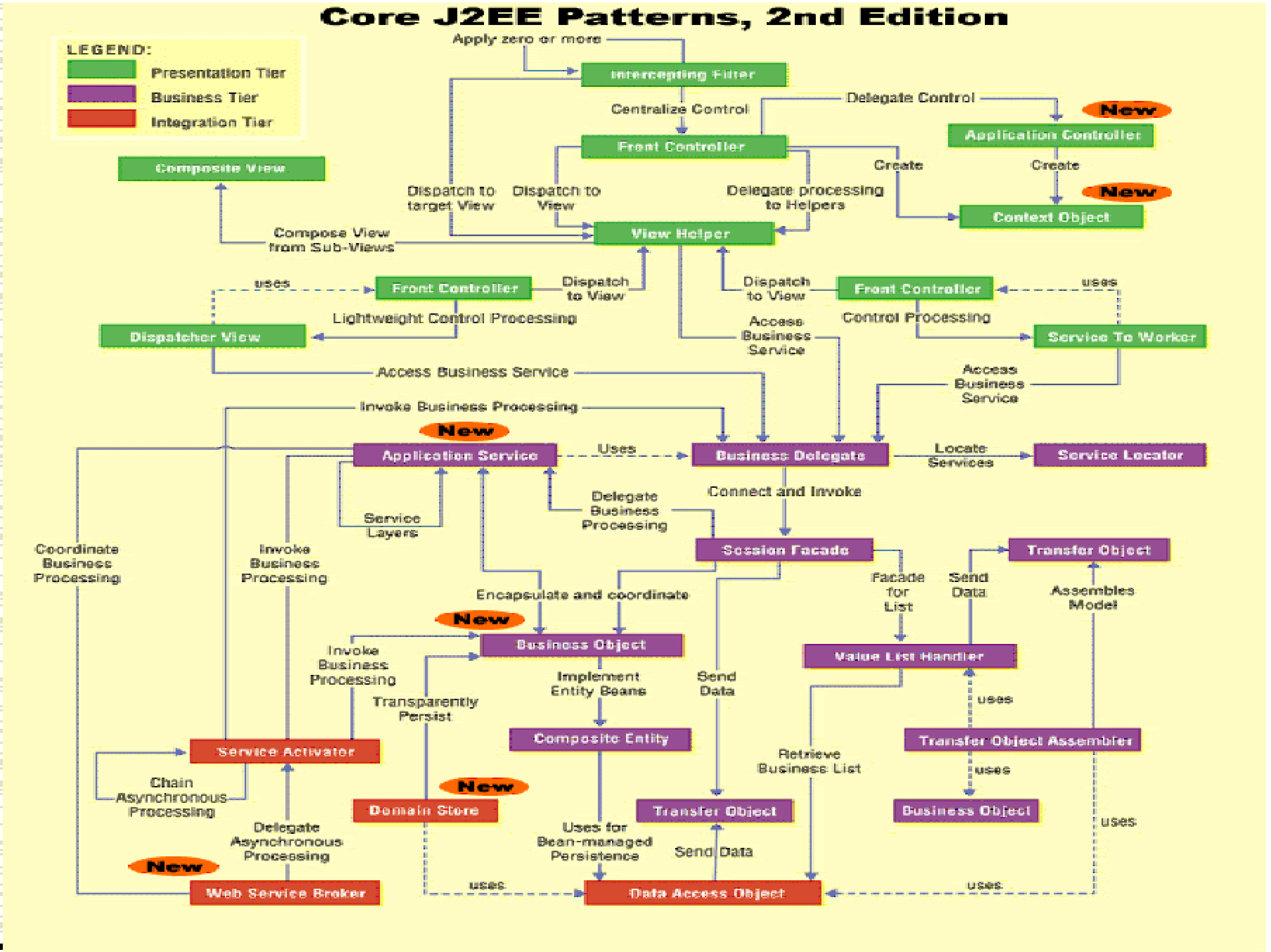
J2EE Business Tier Patterns



J2EE Business Tier Patterns

1 Business Delegate

Problem

You want to hide clients from the complexity of remote communication with business service components.

Forces

You want to access the business-tier components from your presentation-tier components and clients, such as devices, web services, and rich clients.

You want to minimize coupling between clients and the business services, thus hiding the underlying implementation details of the service, such as lookup and access.

You want to avoid unnecessary invocation of remote services.

You want to translate network exceptions into application or user exceptions.

You want to hide the details of service creation, reconfiguration, and invocation retries from the clients.

1 Business Delegate

Solution

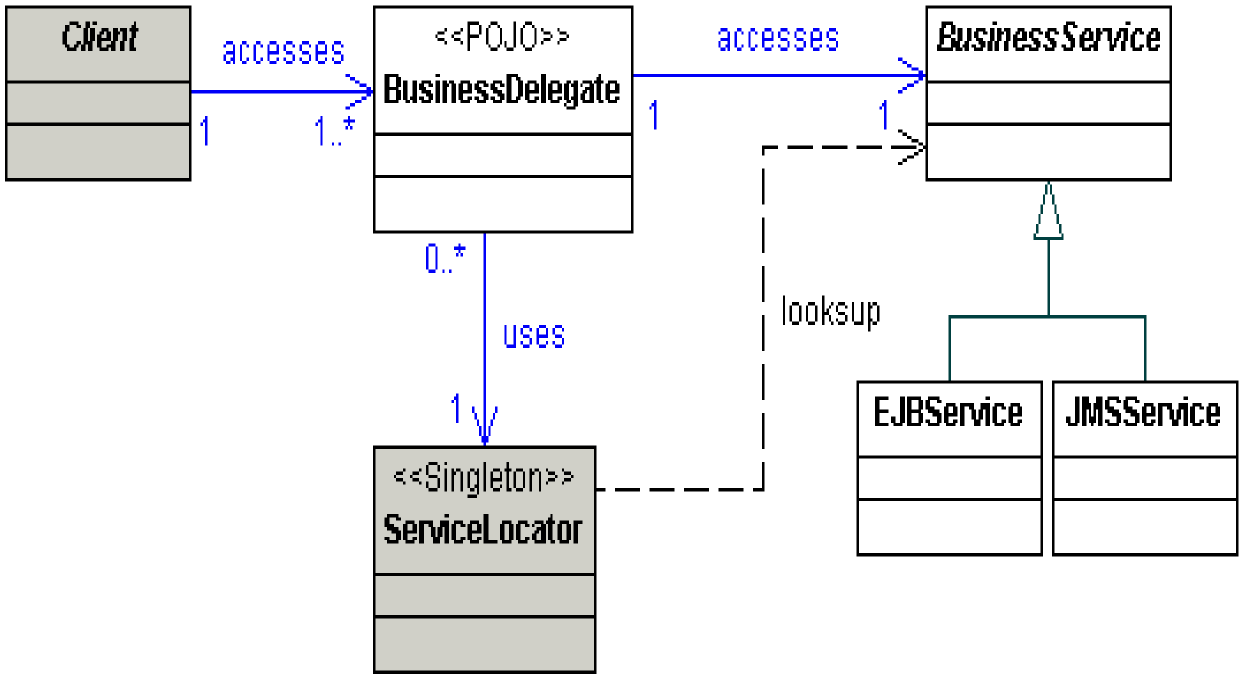
Use a Business Delegate to encapsulate access to a business service.

The Business Delegate hides the implementation details of the business service, such as lookup and access mechanisms.

J2EE Business Tier Patterns

1 Business Delegate

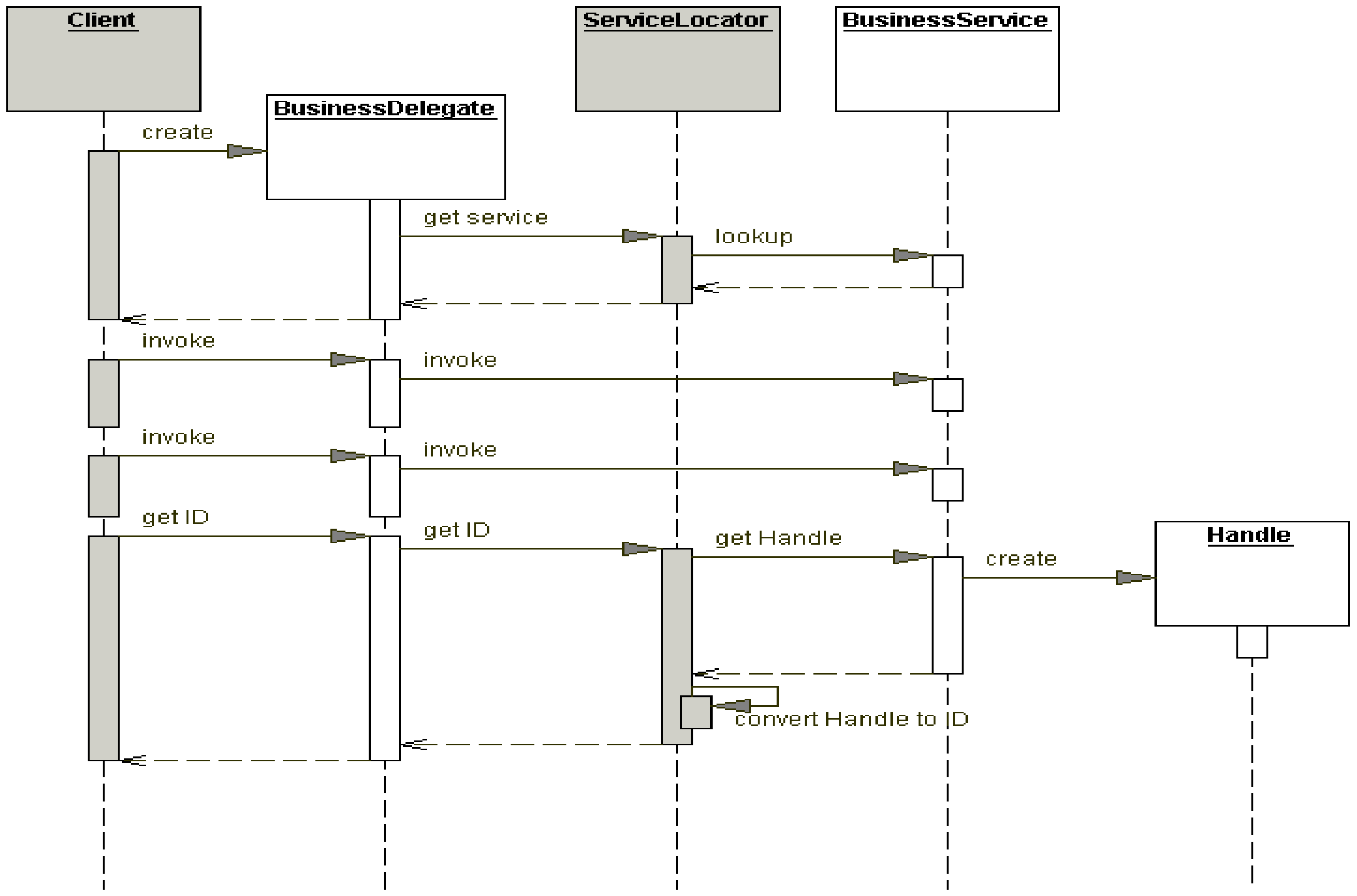
Class Diagram :



J2EE Business Tier Patterns

1 Business Delegate

Sequence Diagram :



J2EE Business Tier Patterns

1 Business Delegate

Strategies

Delegate Proxy Strategy

Delegate Adapter Strategy

Consequences

Reduces coupling, improves maintainability

Translates business service exceptions

Improves availability

Exposes a simpler, uniform interface to the business tier

Improves performance

Introduces an additional layer

Hides remoteness

J2EE Business Tier Patterns

2. Service Locator

Problem

You want to transparently locate business components and services in a uniform manner.

Forces

You want to use the JNDI API to look up and use business components, such as enterprise beans and JMS components, and services such as data sources.

You want to centralize and reuse the implementation of lookup mechanisms for J2EE application clients.

You want to encapsulate vendor dependencies for registry implementations, and hide the dependency and complexity from the clients.

You want to avoid performance overhead related to initial context creation and service lookups.

You want to reestablish a connection to a previously accessed enterprise bean instance, using its Handle object.

2. Service Locator

Solution

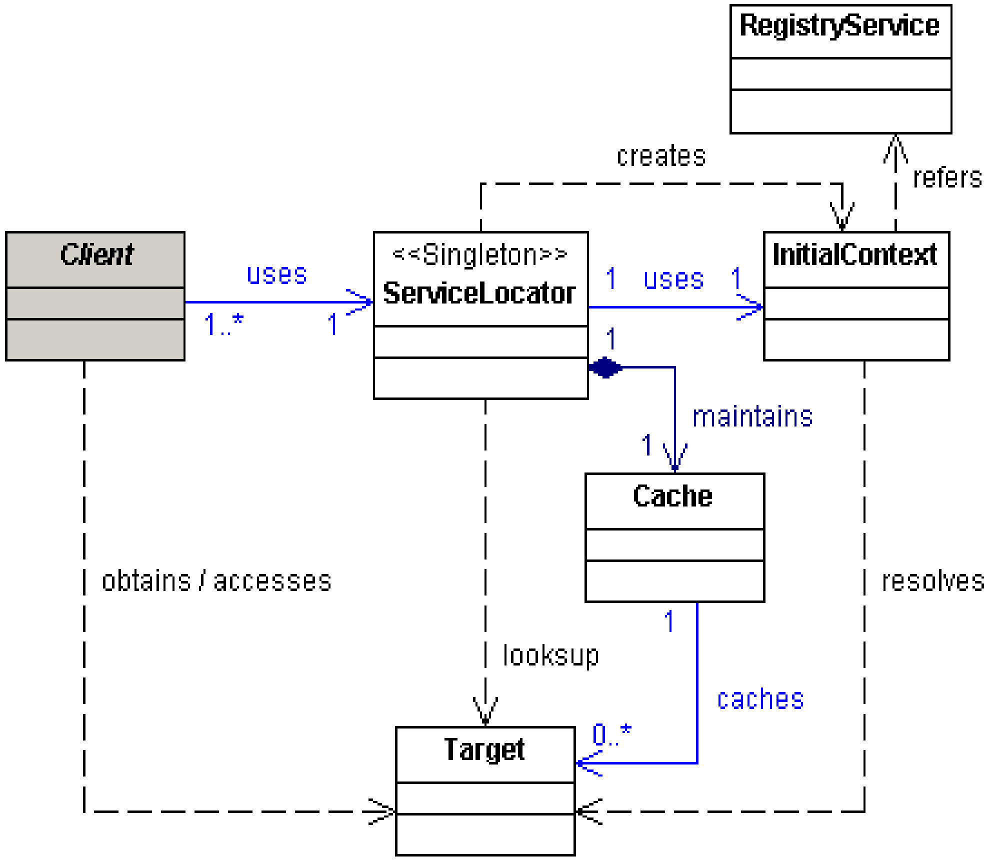
Use a Service Locator to implement and encapsulate service and component lookup.

A Service Locator hides the implementation details of the lookup mechanism and encapsulates related dependencies.

J2EE Business Tier Patterns

2. Service Locator

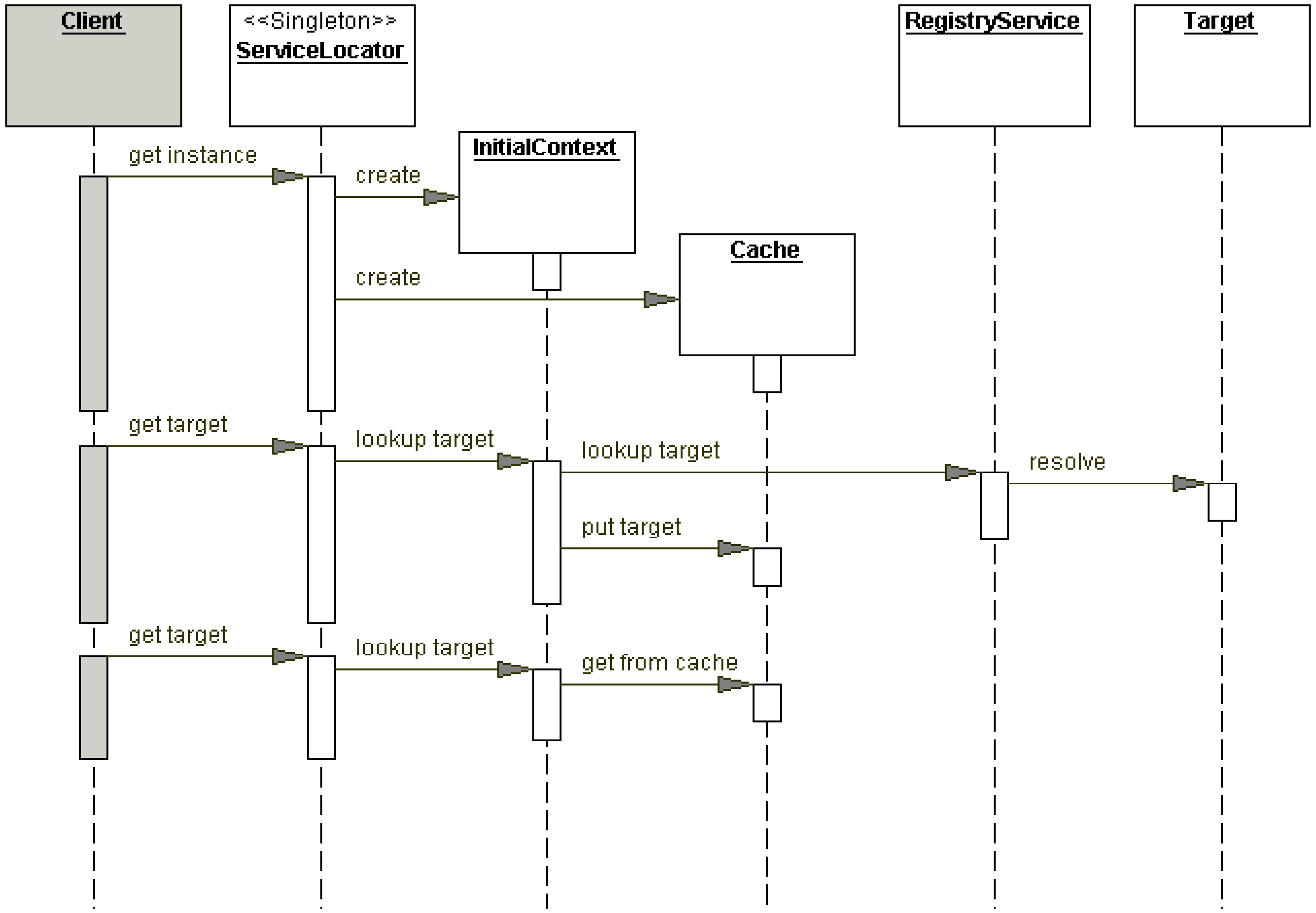
Class Diagram :



J2EE Business Tier Patterns

2. Service Locator

Sequence Diagram :



J2EE Business Tier Patterns

2. Service Locator

Strategies

EJB Service Locator Strategy

JDBC DataSource Service Locator Strategy

JMS Service Locator Strategies

- JMS Queue Service Locator Strategy
- JMS Topic Service Locator Strategy

Web Service Locator Strategy

Consequences

Abstracts complexity

Provides uniform service access to clients

Facilitates adding EJB business components

Improves network performance

Improves client performance by caching

J2EE Business Tier Patterns

3. Session Facade

Problem

You want to expose business components and services to remote clients.

Forces

You want to avoid giving clients direct access to business-tier components, to prevent tight coupling with the clients.

You want to provide a remote access layer to your Business Objects and other business-tier components.

You want to aggregate and expose your Application Services and other services to remote clients.

You want to centralize and aggregate all business logic that needs to be exposed to remote clients.

You want to hide the complex interactions and interdependencies between business components and services to improve manageability, centralize logic, increase flexibility, and improve ability to cope with changes.

3. Session Facade

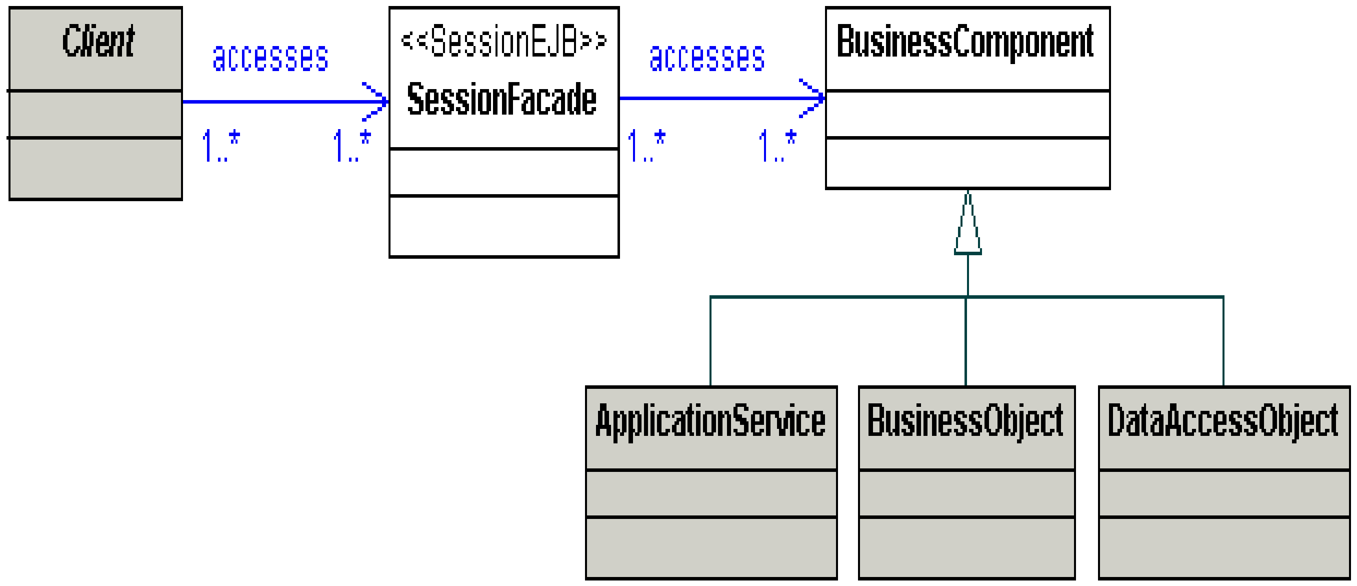
Solution

Use a *Session Façade* to encapsulate business-tier components and expose a coarse-grained service to remote clients.

Clients access a *Session Façade* instead of accessing business components directly.

3. Session Facade

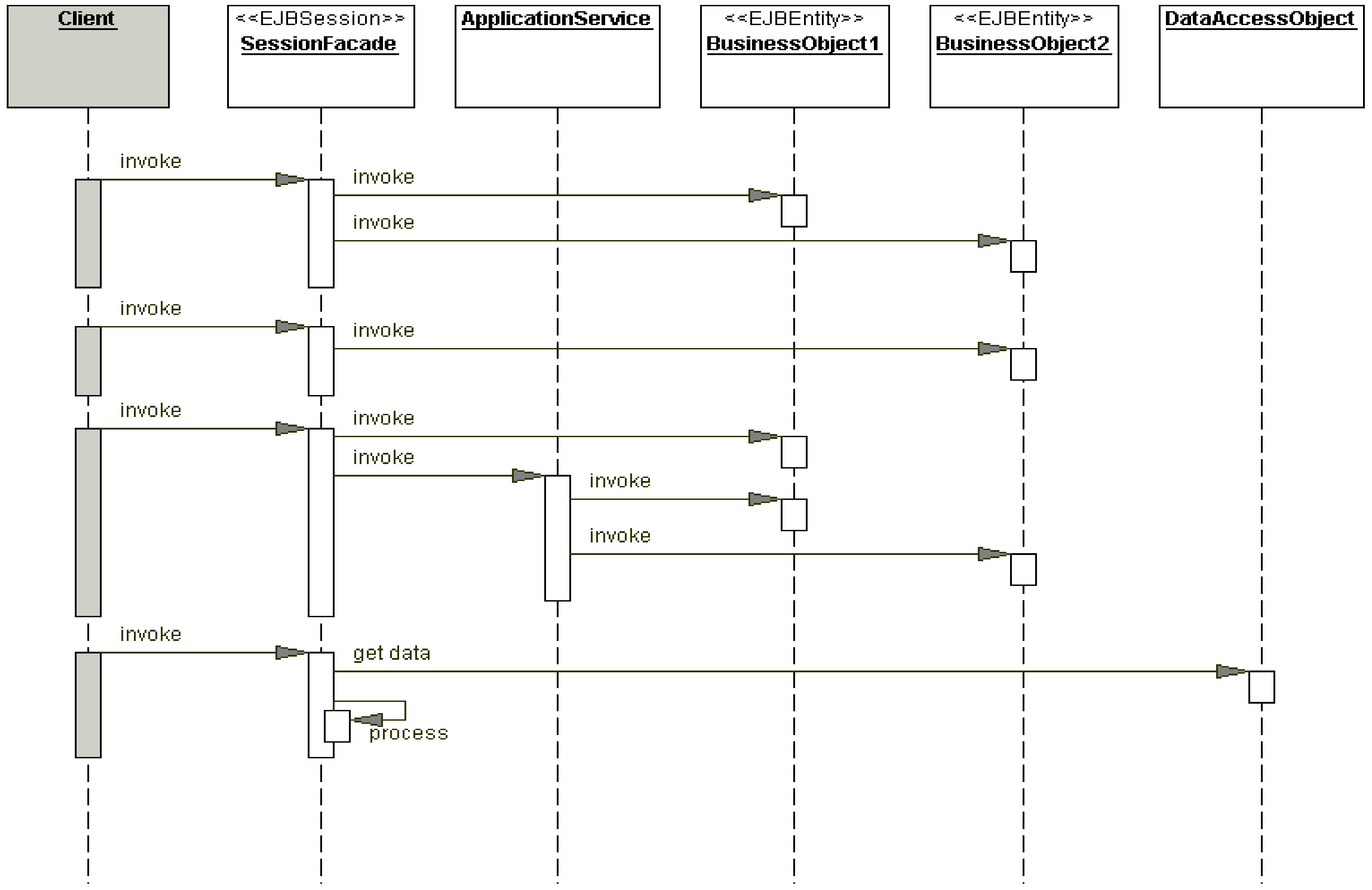
Class Diagram :



J2EE Business Tier Patterns

3. Session Facade

Sequence Diagram :



3. Session Facade

Strategies

Stateless Session Façade Strategy

Stateful Session Façade Strategy

Consequences

Introduces a layer that provides services to remote clients

Exposes a uniform coarse-grained interface

Reduces coupling between the tiers

Promotes layering, increases flexibility and maintainability

Reduces complexity

Improves performance, reduces fine-grained remote methods

Centralizes security management

Centralizes transaction control

Exposes fewer remote interfaces to clients

4. 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 provide a coarser-grained service API over existing business-tier components and services.

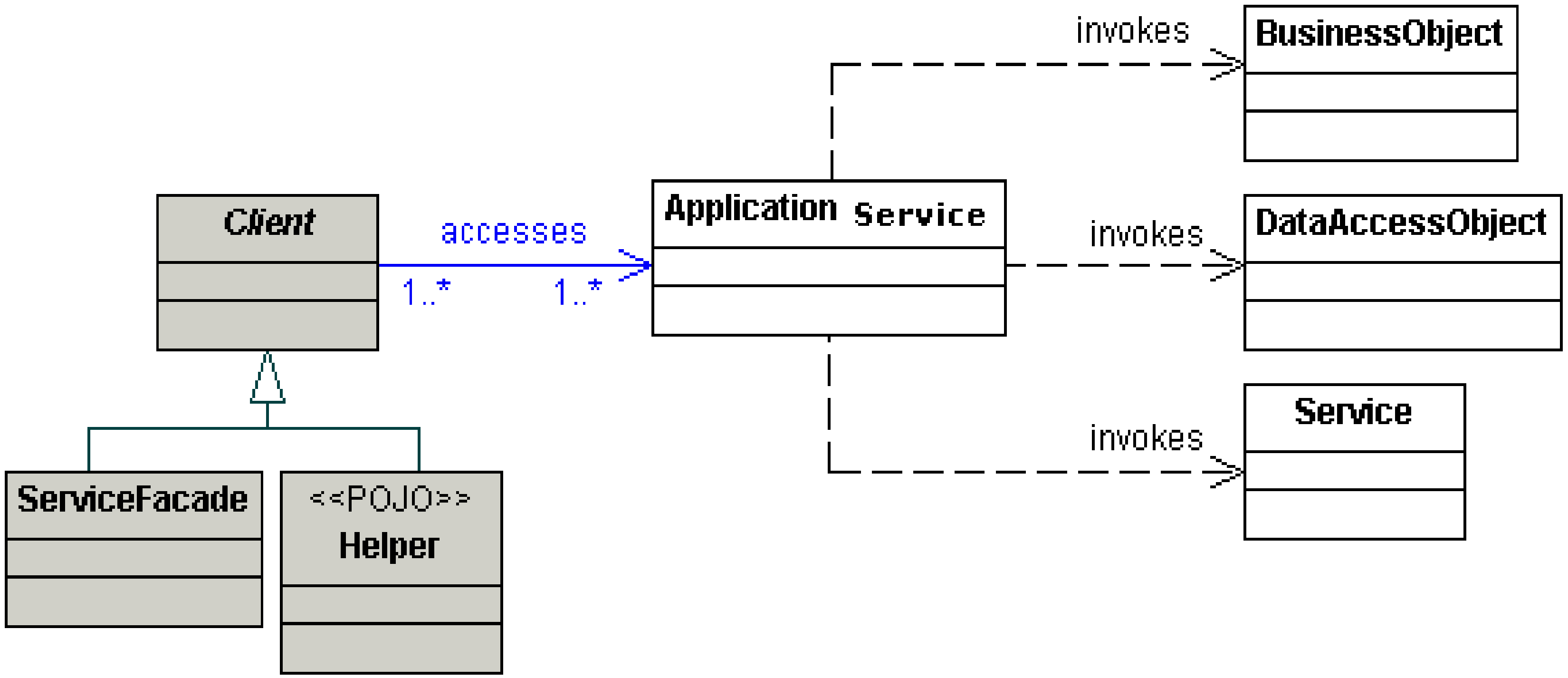
You want to encapsulate use case-specific logic outside of individual Business Objects.

Solution

Use an Application Service to centralize and aggregate behavior to provide a uniform service layer.

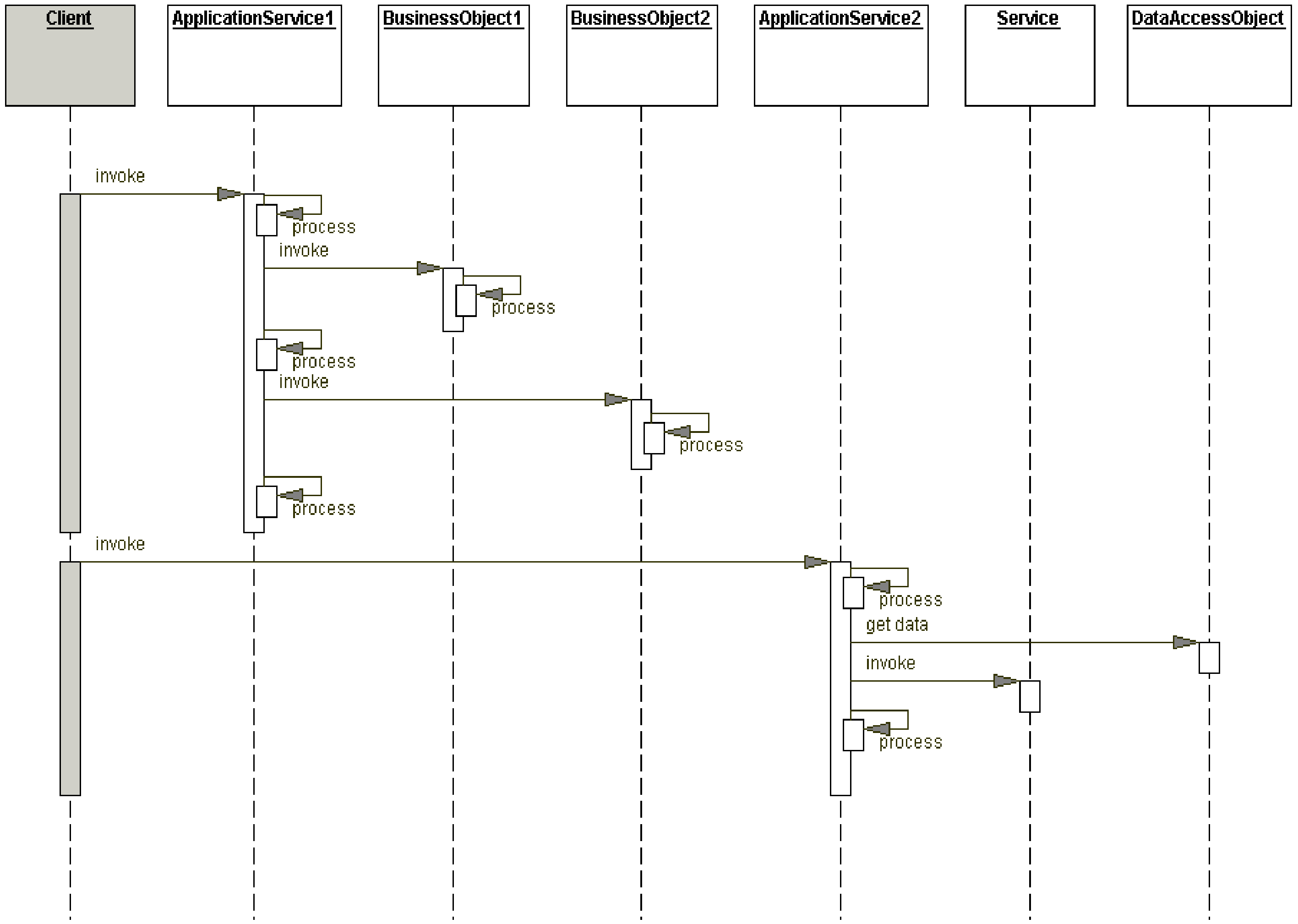
4. Application Service

Class Diagram :



4. Application Service

Sequence Diagram :



4. Application Service

Strategies

Application Service Command Strategy

GoF Strategy for Application Service Strategy

Application Service Layer Strategy

Consequences

Centralizes reusable business and workflow logic

Improves reusability of business logic

Avoids duplication of code

Simplifies facade implementations

Introduces additional layer in the business tier

5. Business Object

Problem

You have a conceptual domain model with business logic and relationship.

Forces

You have a conceptual model containing structured, interrelated composite objects.

You have a conceptual model with sophisticated business logic, validation and business rules.

You want to separate the business state and related behavior from the rest of the application, improving cohesion and reusability.

You want to centralize business logic and state in an application.

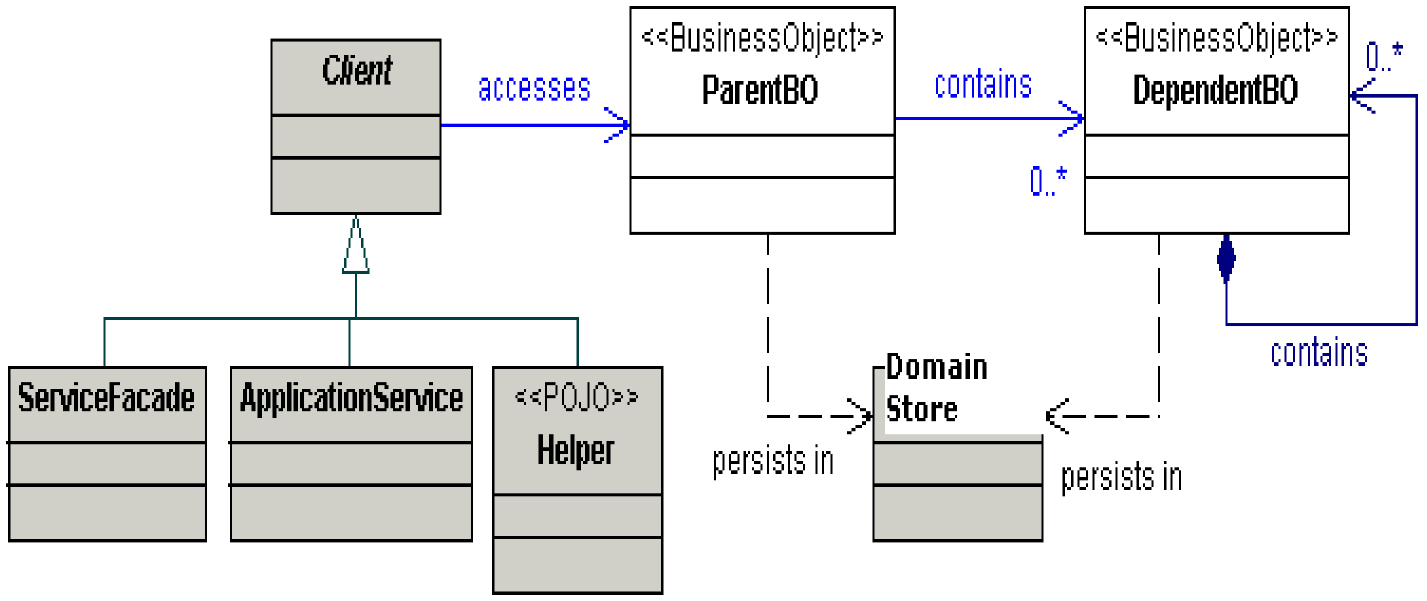
You want to increase reusability of business logic and avoid duplication of code.

Solution

Use Business Objects to separate business data and logic using an object model.

5. Business Object

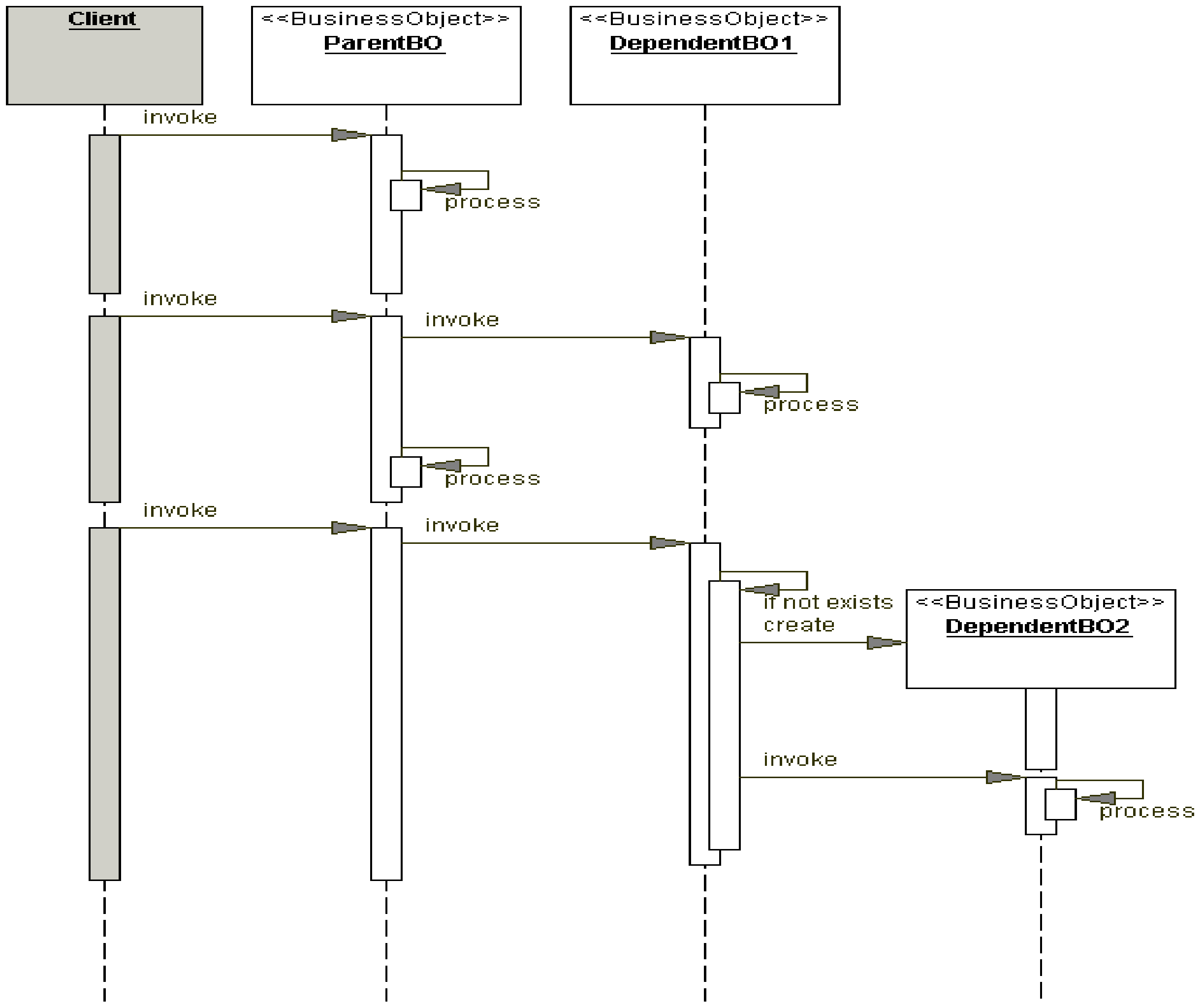
Class Diagram :



J2EE Business Tier Patterns

5. Business Object

Sequence Diagram :



5. Business Object

Strategies

POJO Business Object Strategy

Composite Entity Business Object Strategy

Consequences

Promotes object-oriented approach to the business model implementation

Centralizes business behavior and state, and promotes reuse

Avoids duplication of and improves maintainability of code

Separates persistence logic from business logic

Promotes service-oriented architecture

POJO implementations can induce, and are susceptible to, stale data

Adds extra layer of indirection

Can result in bloated objects

6. Composite Entity

Problem

You want to use entity beans to implement your conceptual domain model.

Forces

You want to avoid the drawbacks of remote entity beans, such as network overhead and remote inter-entity bean relationships.

You want to leverage bean-managed persistence (BMP) using custom or legacy persistence implementations.

You want to implement parent-child relationships efficiently when implementing Business Objects as entity beans.

You want to encapsulate and aggregate existing POJO Business Objects with entity beans.

You want to leverage EJB container transaction management and security features.

You want to encapsulate the physical database design from the clients.

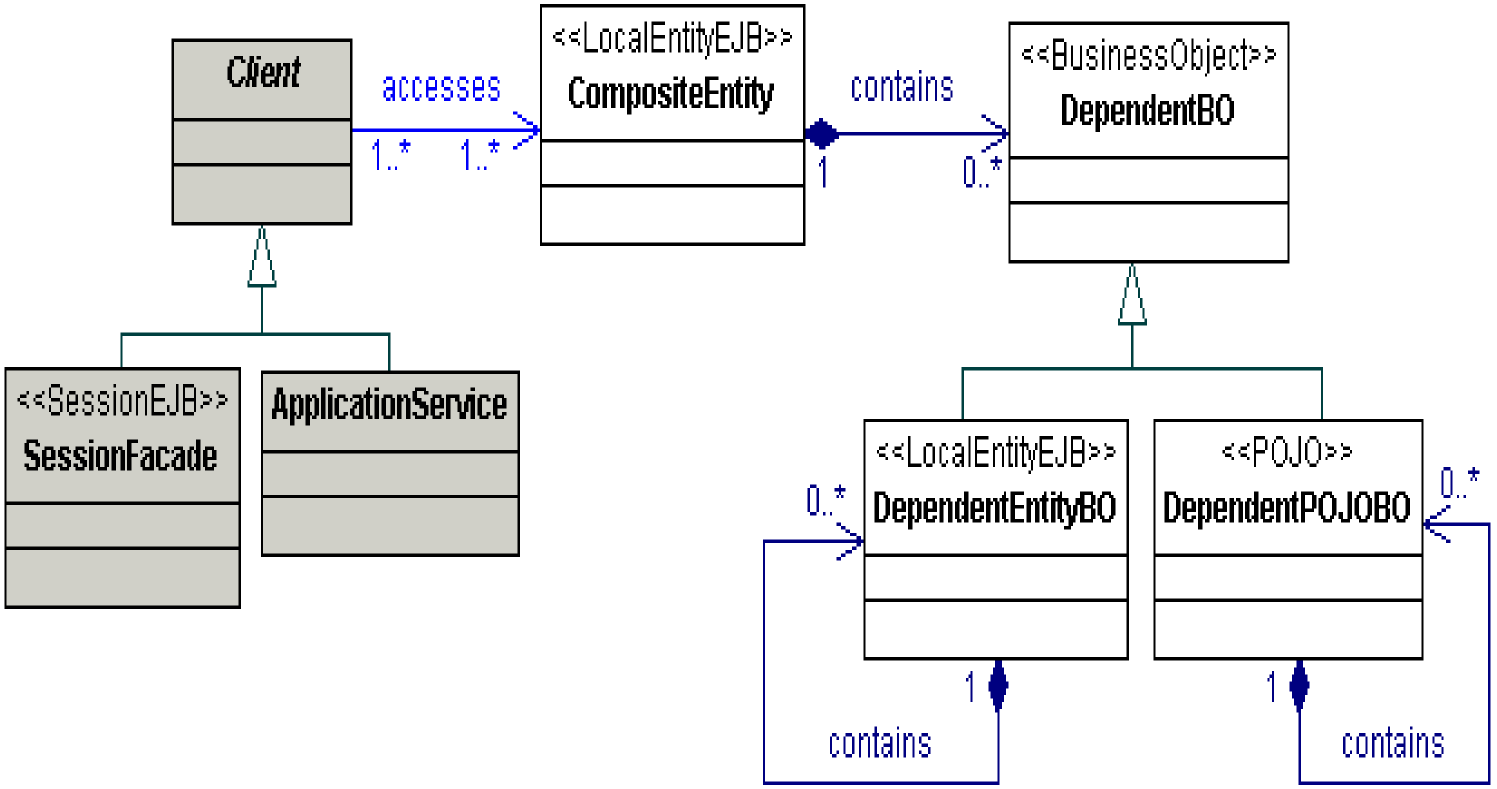
6. Composite Entity

Solution

Use a Composite Entity to implement persistent Business Objects using local entity beans and POJOs. Composite Entity aggregates a set of related Business Objects into coarse-grained entity bean implementations.

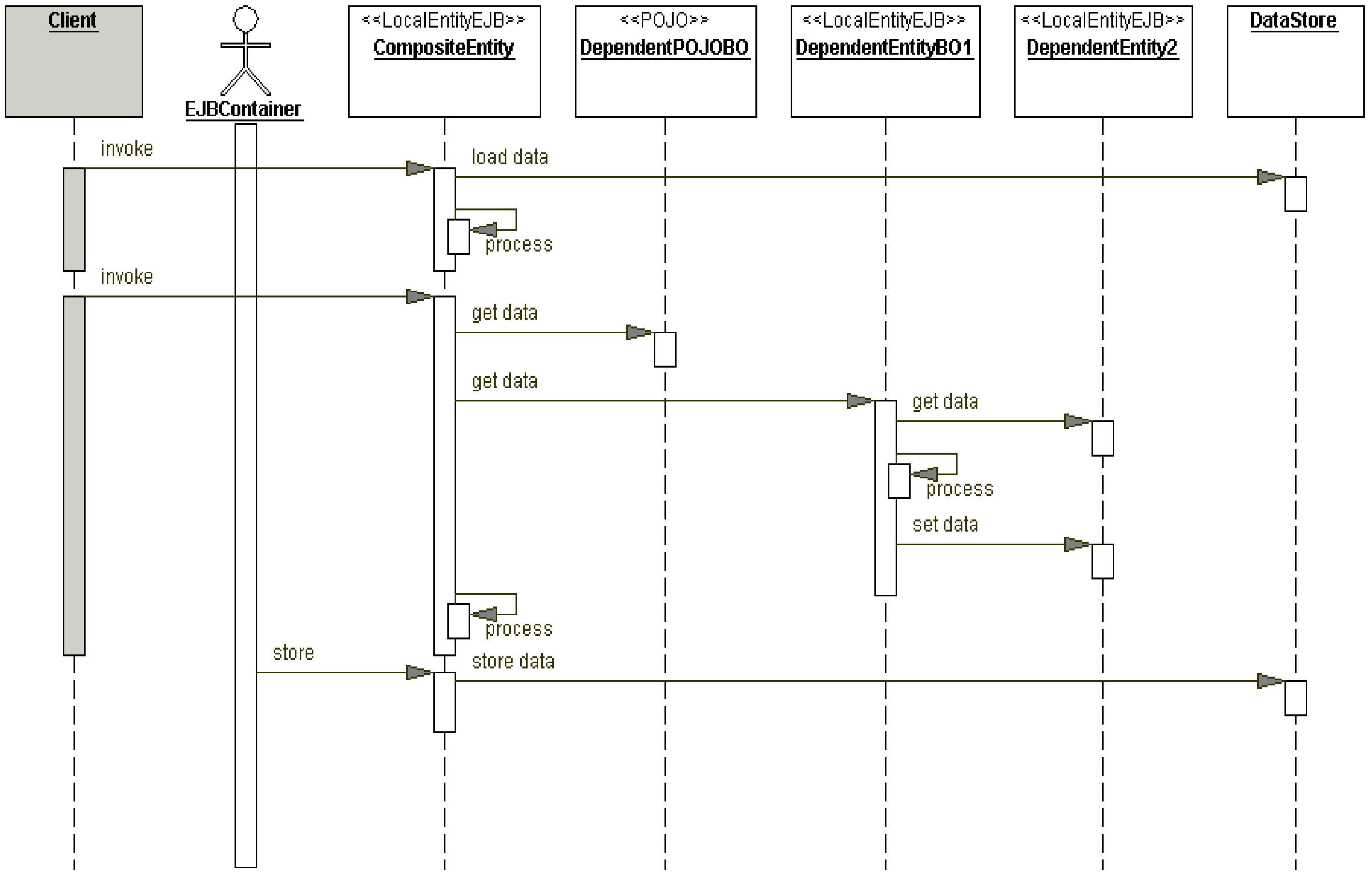
6. Composite Entity

Class Diagram :



6. Composite Entity

Sequence Diagram :



6. Composite Entity

Strategies

Composite Entity Remote Facade Strategy

Composite Entity BMP Strategies

Lazy Loading Strategy

Store Optimization (Dirty Marker) Strategy

Composite Transfer Object Strategy

Consequences

Increases maintainability

Improves network performance

Reduces database schema dependency

Increases object granularity

Facilitates composite transfer object creation

7. Transafer Object

Problem

You want to transfer multiple data elements over a tier.

Forces

You want clients to access components in other tiers to retrieve and update data.

You want to reduce remote requests across the network.

You want to avoid network performance degradation caused by chattier applications that have high network traffic.

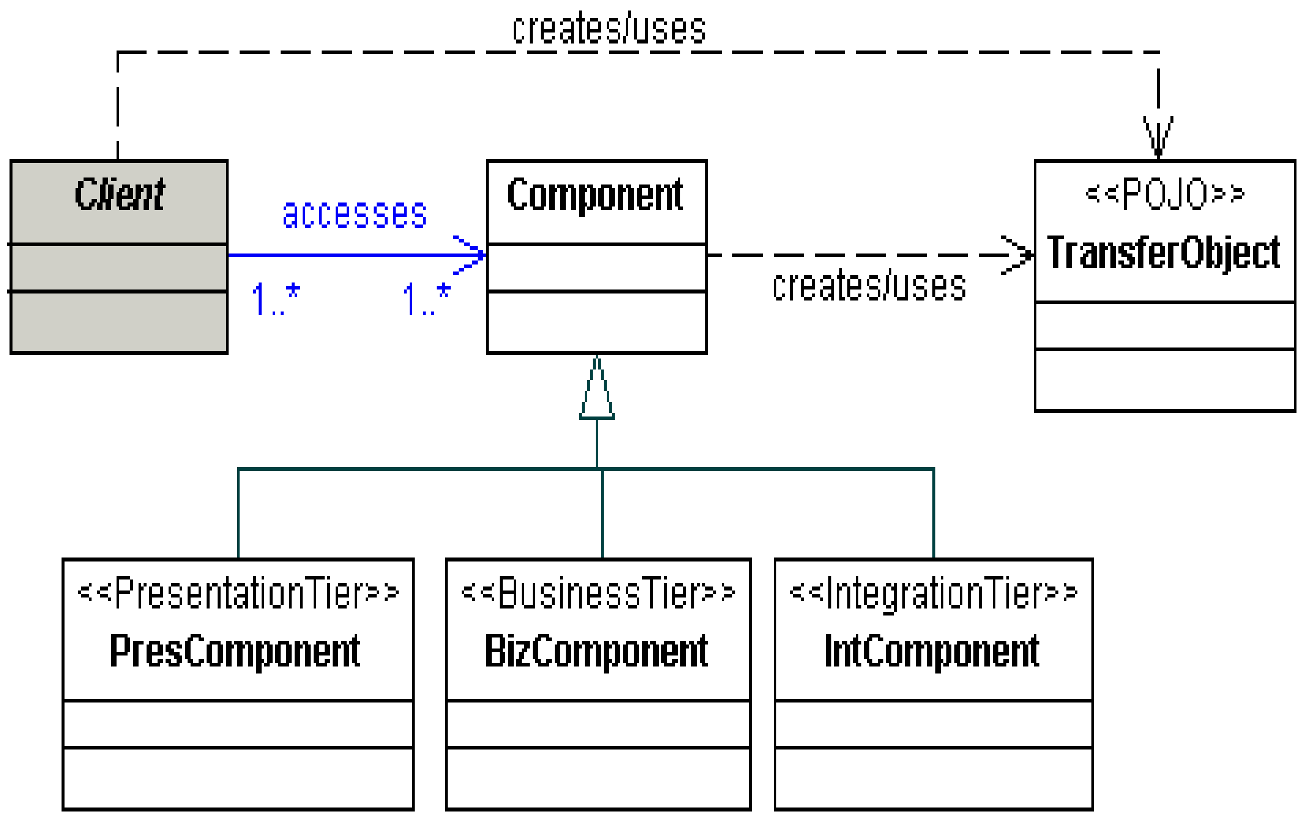
Solution

Use a Transfer Object to carry multiple data elements across a tier.

J2EE Business Tier Patterns

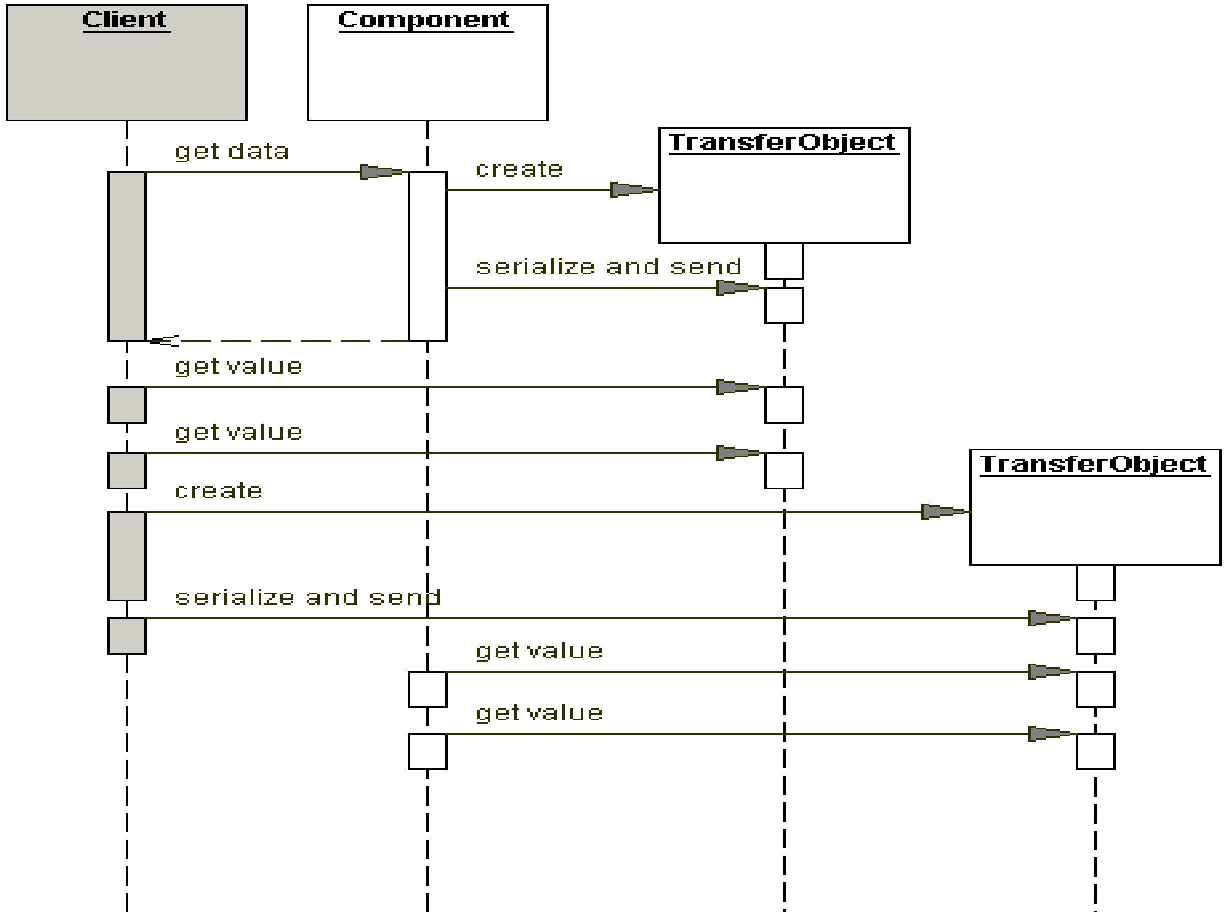
7. Transfer Object

Class Diagram :



7. Transfer Object

Sequence Diagram :



7. Transfer Object

Strategies

Updatable Transfer Objects Strategy

Multiple Transfer Objects Strategy

Entity Inherits Transfer Object Strategy

Consequences

Reduces network traffic

Simplifies remote object and remote interface

Transfers more data in fewer remote calls

Reduces code duplication

Introduces stale transfer objects

Increases complexity due to synchronization and version control

8. Transafer Object Assembler

Problem

You want to obtain an application model that aggregates transfer objects from several business components.

Forces

You want to encapsulate business logic in a centralized manner and prevent implementing it in the client.

You want to minimize the network calls to remote objects when building a data representation of the business-tier object model.

You want to create a complex model to hand over to the client for presentation purposes.

You want the clients to be independent of the complexity of model implementation, and you want to reduce coupling between the client and the business components.

8. Transafer Object Assembler

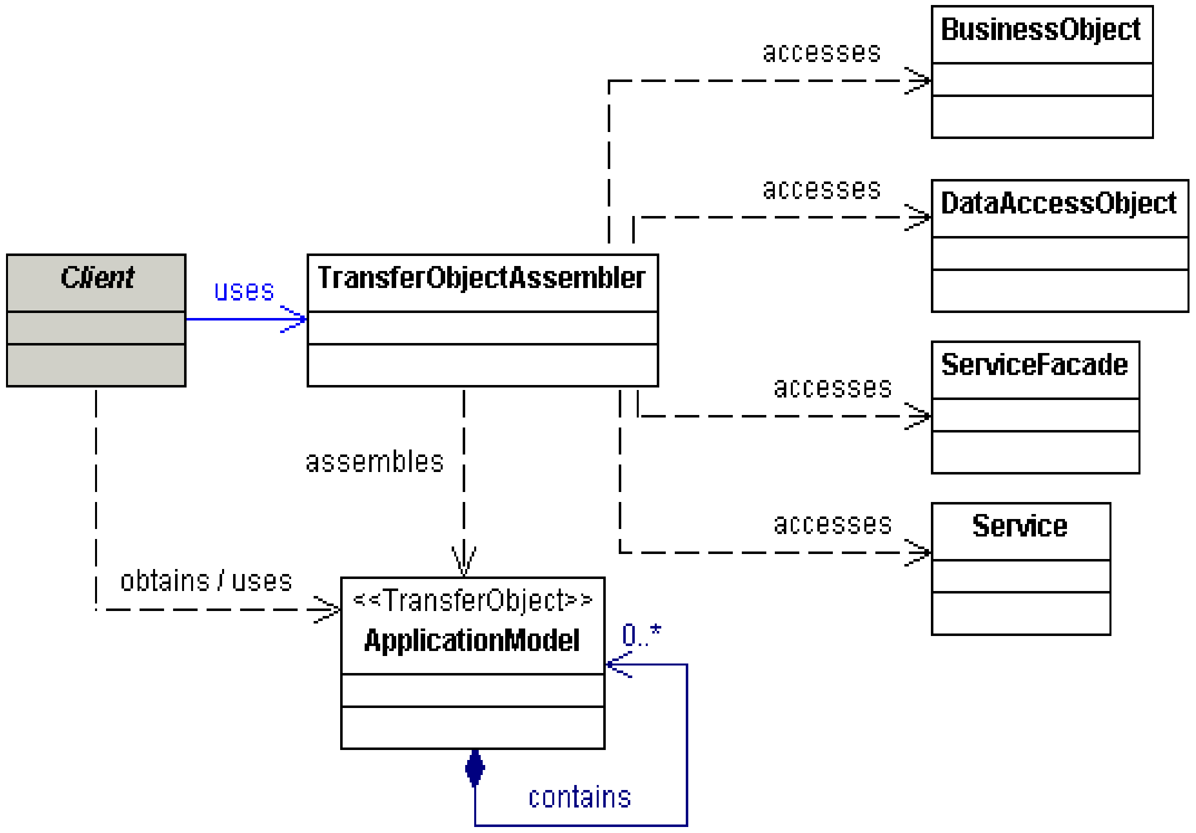
Solution

Use a Transfer Object Assembler to build an application model as a composite Transfer Object.

The Transfer Object Assembler aggregates multiple Transfer Objects from various business components and services, and returns it to the client.

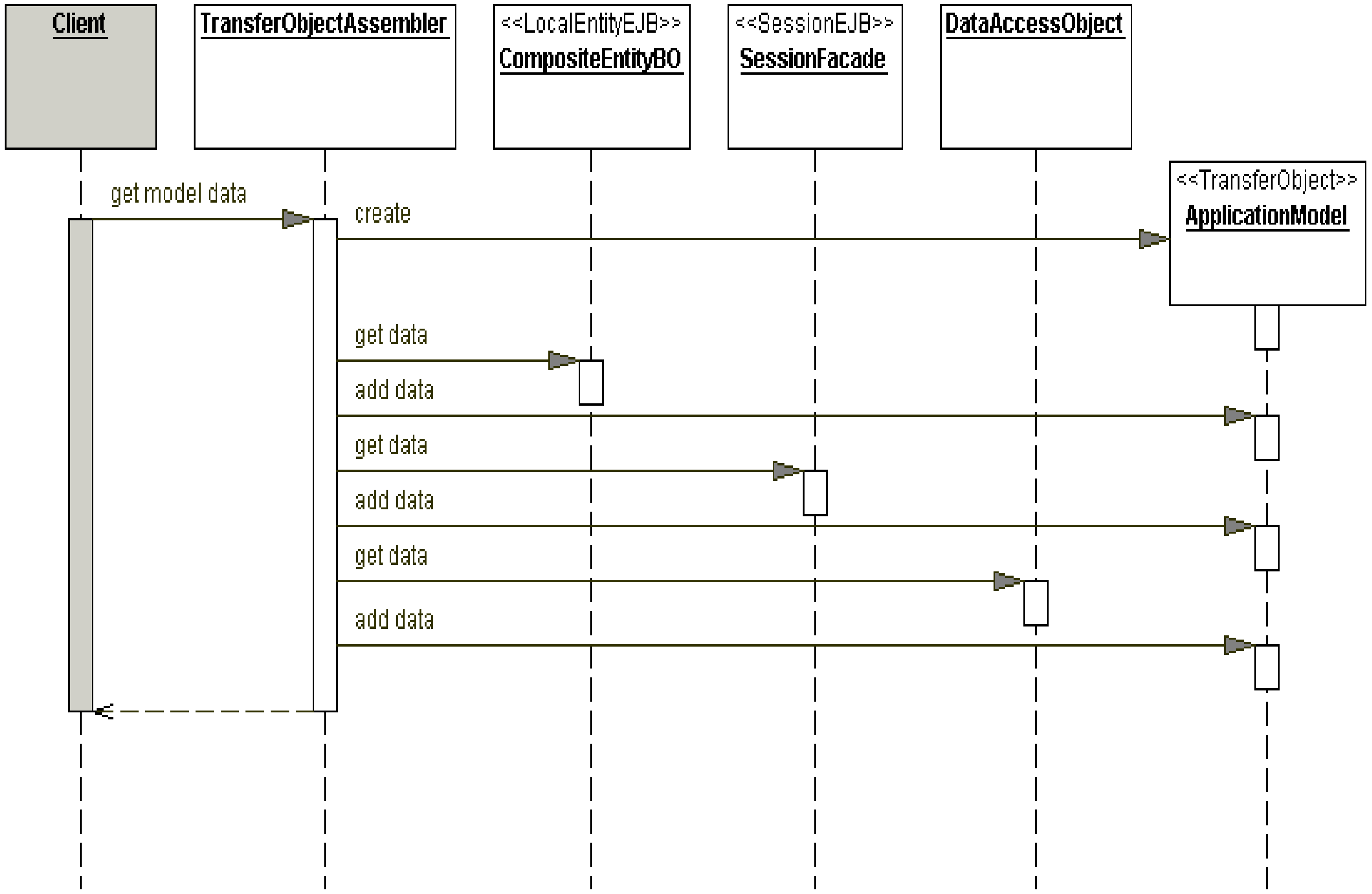
8. Transfer Object Assembler

Class Diagram :



8. Transfer Object Assembler

Sequence Diagram :



8. Transfer Object Assembler

Strategies

POJO Transfer Object Assembler Strategy

Session Bean Transfer Object Assembler Strategy

Consequences

Separates business logic, simplifies client logic

Reduces coupling between clients and the application model

Improves network performance

Improves client performance

Can introduce stale data

9. Value List Handler (2nd level cache)

Problem

You have a remote client that wants to iterate over a large results list.

Forces

You want to avoid the overhead of using EJB finder methods for large searches.

You want to implement a read-only use-case that does not require a transaction.

You want to provide the clients with an efficient search and iterate mechanism over a large results set.

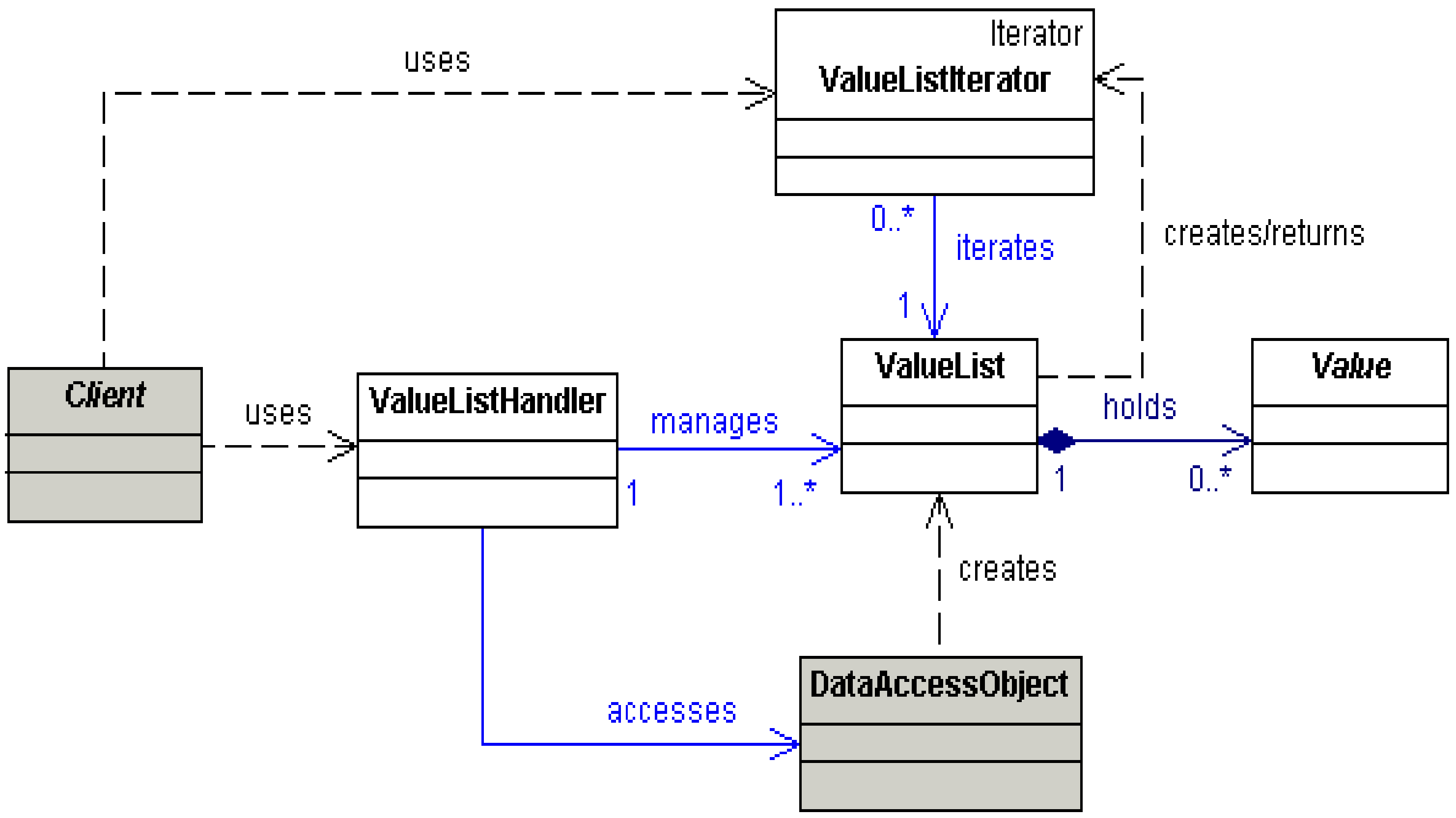
You want to maintain the search results on the server side.

Solution

Use a Value List Handler to search, cache the results, and allow the client to traverse and select items from the results.

9. Value List Handler

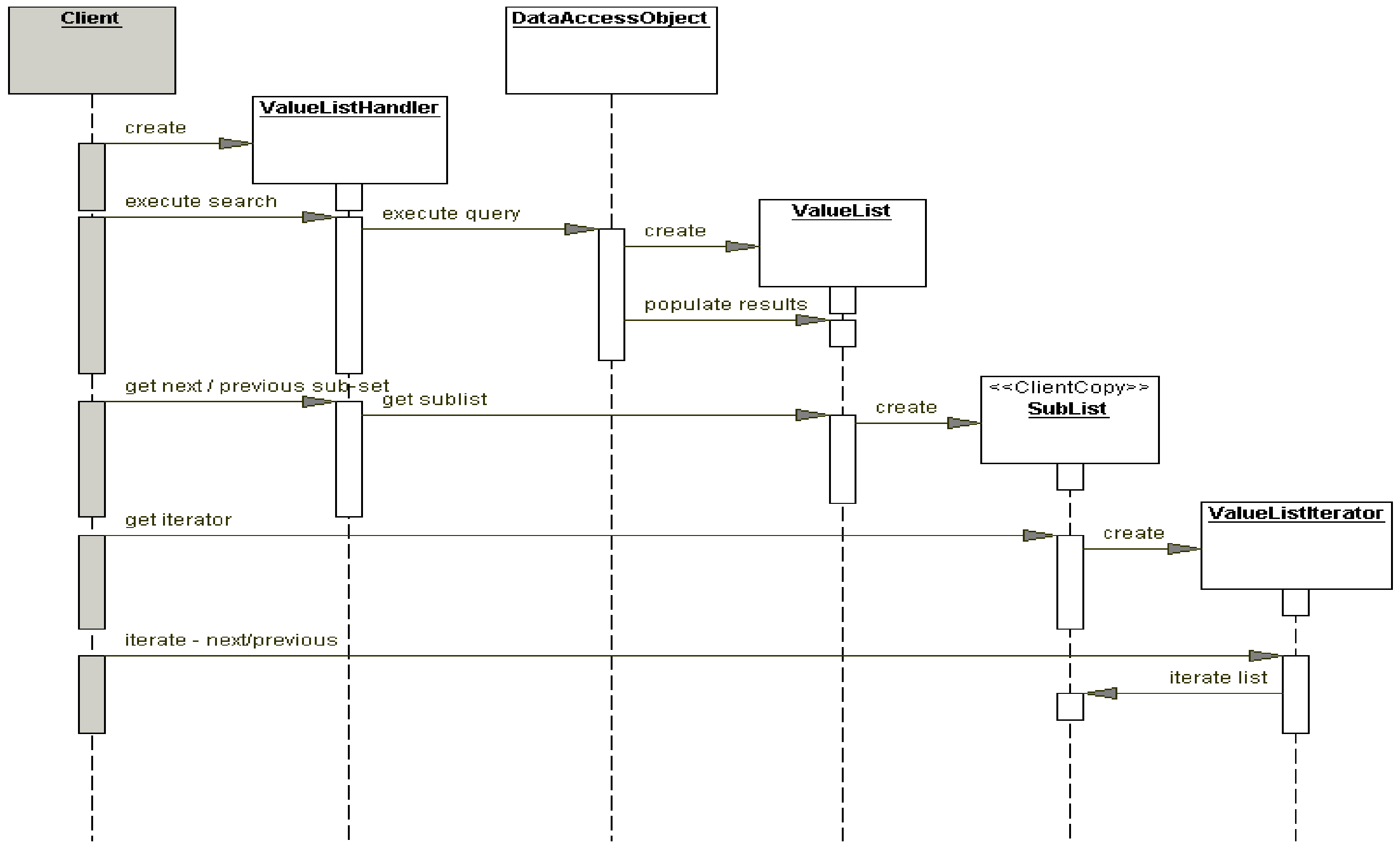
Class Diagram :



J2EE Business Tier Patterns

9. Value List Handler

Sequence Diagram :



J2EE Business Tier Patterns

9. Value List Handler

Strategies

POJO Handler Strategy

Value List Handler Session Façade Strategy

Value List from Data Access Object Strategy

Consequences

Provides efficient alternative to EJB finders

Caches search results

Provides flexible search capabilities

Improves network performance

Allows deferring entity bean transactions

Promotes layering and separation of concerns

Creating a large list of Transfer Objects can be expensive