Core JEE patterns - Business tier design patterns

Business delegate

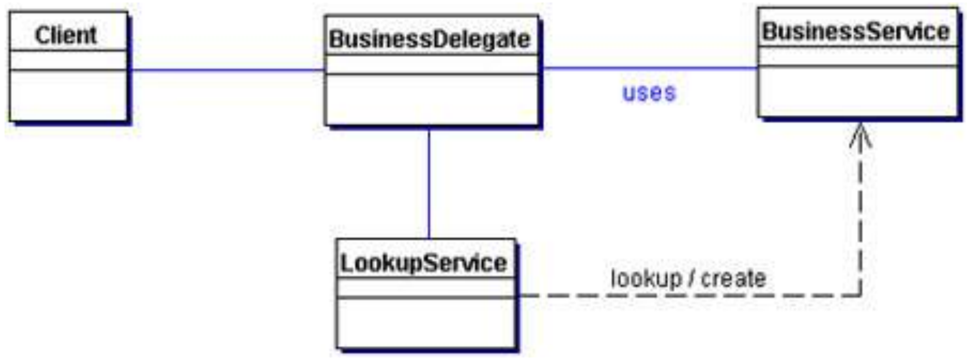
Problems it is trying to solve

- Minimize coupling between business and presentation layers in JEE distributed applications.
- Changes in business layer shall not affect the client (presentation layer).
- Hide complex communication details such as JNDI lookup from the presentation layer.
- Want to translate low level exceptions in to exceptions the client can understand.

Solution

- Business delegate encapsulate access to a business service.
- Business delegate is a logical abstraction not a physical one.
- This could even be placed at the presentation layer. However it is a logical extension of the business layer & business layer developers have to do it.
- Business delegate shall additionally cache data such as remote references.
- Business delegate could also transparently do retry in case of failures.

Class diagram



Reference

- http://www.oracle.com/technetwork/java/businessdelegate-137562.html

Business Delegate Vs Façade

- Delegation is standing between the client and the actual implementation, usually hiding/filtering/augmenting certain functionality of the implementation from the client.
- Facade is providing a course-grained API hiding more complex logic and/or coordination, usually bundling up several implementations that work together, and usually as a convenience to the client.

Business Object

When to use business object?

You have business logic associated with domain entities. You want to model them. Example- Order, Item. You want to package business logic, it's state together and ts persistence together.

Tips on business objects

** Business objects implement a layer of business entities that describe the business.

** Do not expose business objects directly to client. Instead expose through facade, application service.

** Do not manage persistence through business objects.

** Avoid bloating business objects. Any business behavior that acts in multiple business object shall be modeled as application service.

Data Transfer Object (DTO)

Purpose

Carry a set of data across tiers. Sometimes transfer object can even be used to update data across tiers.

Value List Handler

Purpose

- Remote client that wants to iterate over large set of search results.
- This pattern allows to search, cache results..allow the client to traverse and select items from the list.

Composite Entity

Purpose

- Represent a coarse grained entity containing dependent objects.
- This forms a graph of entities which are related.
- Client will access only the composite.
- Lifecycle of dependents are managed by composite.

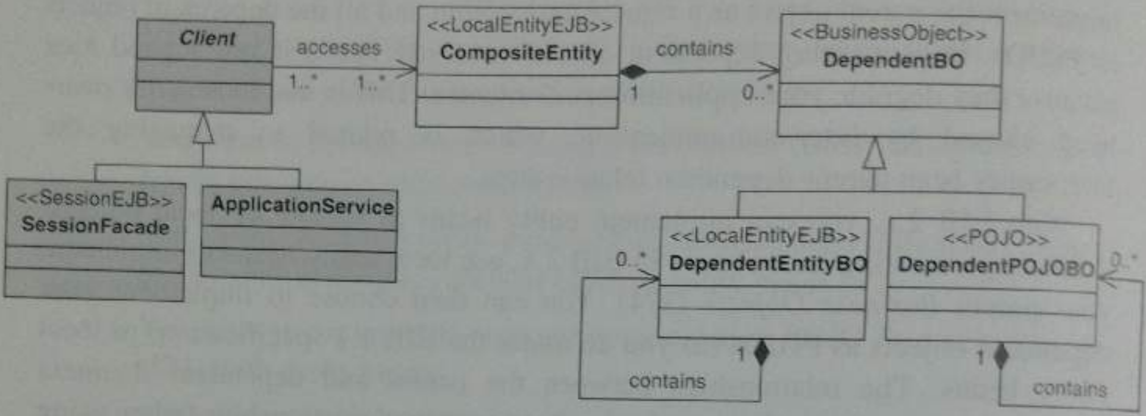


Figure 7.30 Composite Entity Class Diagram

The sequence diagram in Figure 7.31 shows the interactions in the *Composite Entity* pattern.

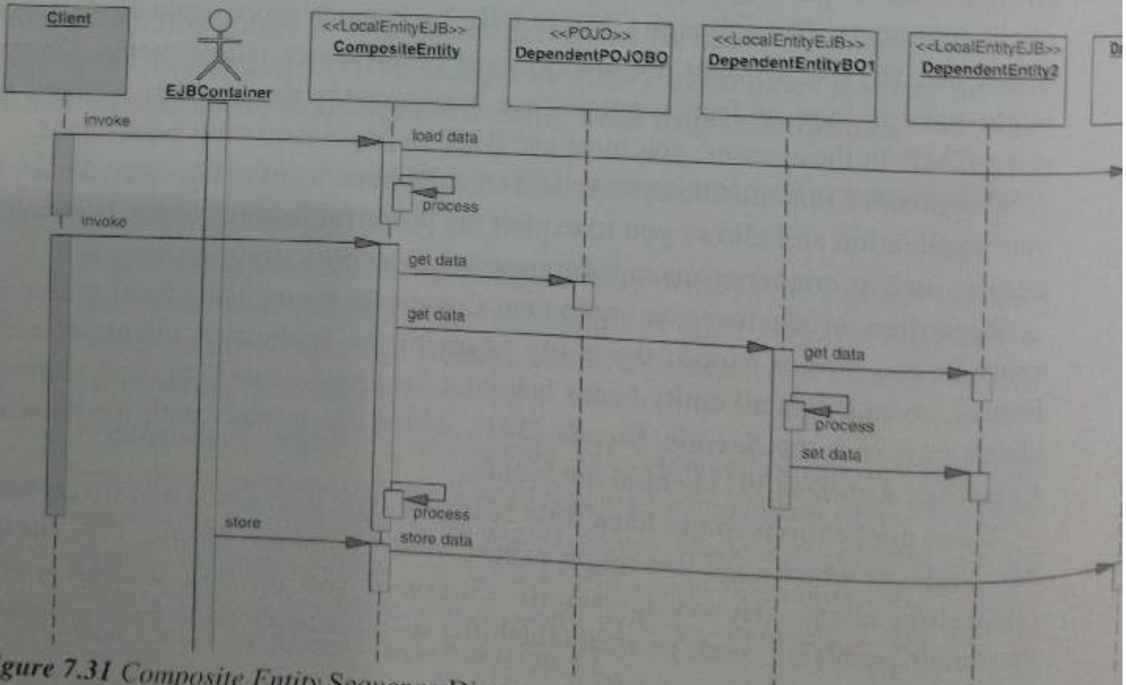


Figure 7.31 Composite Entity Sequence Diagram

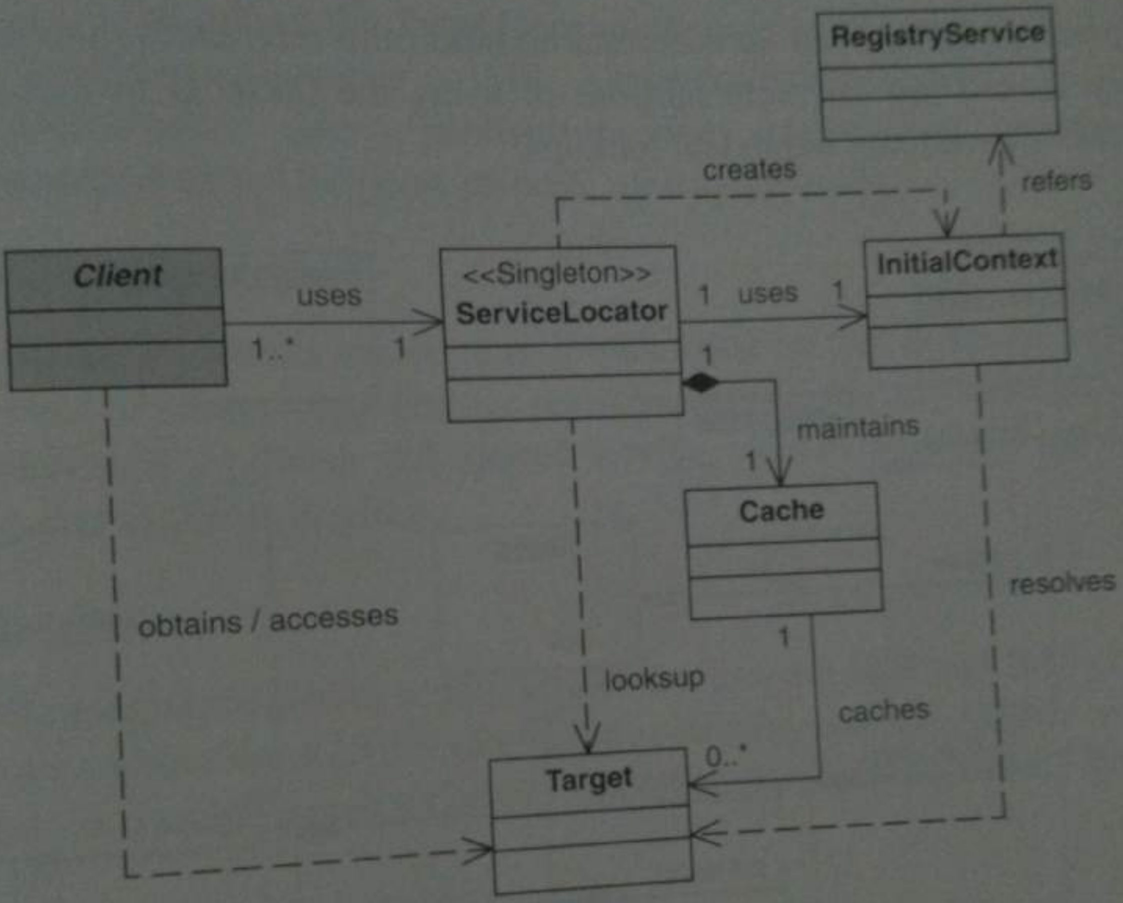
Service Locator

Purpose

- Mechanism to implement lookup at server side.

Solution

- Uses a cache to reuse lookups? Caches references.
- Uses registry and the initial context.
- Usually implemented as singletons.



ure 7.5 *Service Locator* Class Diagram

rticipants and Responsibilities

gure 7.6 contains the sequence diagram that shows the interaction be
rious participants of the *Service Locator* pattern.

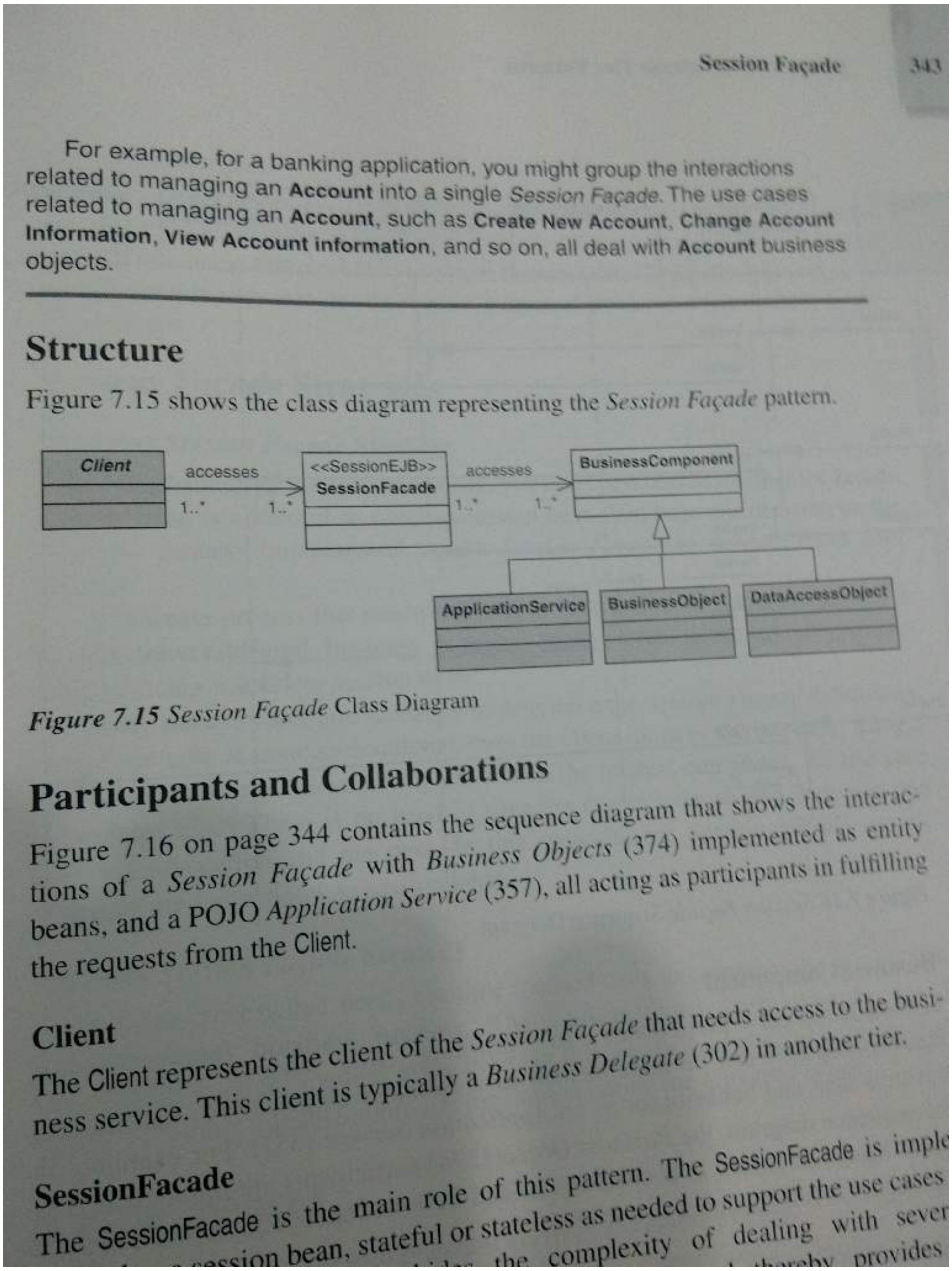
lient

Locator that needs to locate a

Session Facade

Purpose

- Expose business components to client in coarse grained manner.
- Client won't be accessing business components directly but this facade.



For example, for a banking application, you might group the interactions related to managing an **Account** into a single *Session Façade*. The use cases related to managing an **Account**, such as **Create New Account**, **Change Account Information**, **View Account information**, and so on, all deal with **Account** business objects.

Structure

Figure 7.15 shows the class diagram representing the *Session Façade* pattern.

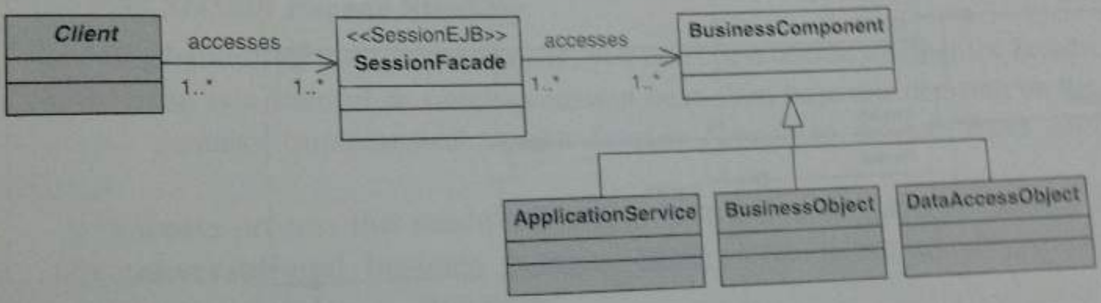


Figure 7.15 *Session Façade* Class Diagram

Participants and Collaborations

Figure 7.16 on page 344 contains the sequence diagram that shows the interactions of a *Session Façade* with *Business Objects* (374) implemented as entity beans, and a POJO *Application Service* (357), all acting as participants in fulfilling the requests from the Client.

Client

The Client represents the client of the *Session Façade* that needs access to the business service. This client is typically a *Business Delegate* (302) in another tier.

SessionFacade

The SessionFacade is the main role of this pattern. The SessionFacade is imple

session bean, stateful or stateless as needed to support the use cases

the complexity of dealing with sever

thereby provides

Solution

- Facade can be stateless or stateful. Decide based on the business process. If client needs multiple calls to facade (conversational), then it is more of stateful.

Related patterns

- Business delegate is client side abstraction of session facade.
-

Application Service

Purpose.

- If implementation requires using multiple business objects (BO), then implement application service. Example - email service.

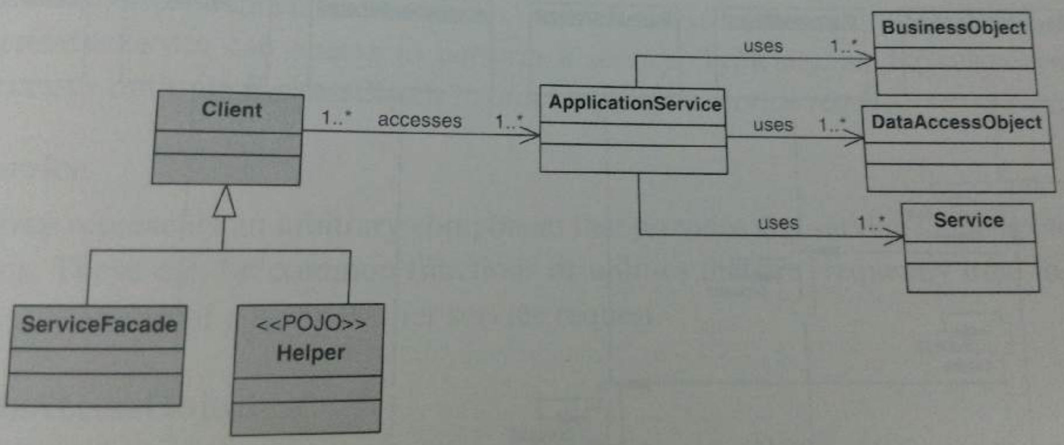


Figure 7.18 Application Service Class Diagram