Session Facade

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

In a multitiered J2EE application environment, the following problems arise:

- Tight coupling, which leads to direct dependence between clients and business objects;
- 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.

A multitiered J2EE application has numerous server-side objects that are implemented as enterprise beans. In addition, some other arbitrary objects may provide services, data, or both. These objects are collectively referred to as business objects, since they encapsulate business data and business logic.

J2EE applications implement business objects that provide processing services as session beans. Coarse-grained business objects that represent an object view of persistent storage and are shared by multiple users are usually implemented as entity beans.

Application clients need access to business objects to fulfill their responsibilities and to meet user requirements. Clients can directly interact with these business objects because they expose their interfaces. When you expose business objects to the client, the client must understand and be responsible for the business data object relationships, and must be able to handle business process flow.

However, direct interaction between the client and the business objects leads to tight coupling between the two, and such tight coupling makes the client directly dependent on the implementation of the business objects. Direct dependence means that the client must represent and implement the complex interactions regarding business object lookups and creations, and must manage the relationships between the participating business objects as well as understand the responsibility of transaction demarcation.

As client requirements increase, the complexity of interaction between various business objects increases. The client grows larger and more complex to fulfill these requirements. The client becomes very susceptible to changes in the business object layer; in addition, the client is unnecessarily exposed to the underlying complexity of the system.

Tight coupling between objects also results when objects manage their relationship within themselves. Often, it is not clear where the relationship is managed. This leads to complex relationships between business objects and rigidity in the application. Such lack of flexibility makes the application less manageable when changes are required.

When accessing the enterprise beans, clients interact with remote objects. Network performance problems may result if the client directly interacts with all the participating business objects. When invoking enterprise beans, every client invocation is potentially a remote method call. Each access to the business object is relatively fine-grained. As the number of participants increases in a scenario, the number of such remote method calls increases. As the number of remote method calls increases, the chattiness between the client and the server-side business objects increases. This may result in network performance degradation for the application, because the high volume of remote method calls increases the amount of interaction across the network layer.

A problem also arises when a client interacts directly with the business objects. Since the business objects are directly exposed to the clients, there is no unified strategy for accessing the business objects. Without such a uniform client access strategy, the business objects are exposed to clients and may reduce consistent usage.

Forces

- Provide a simpler interface to the clients by hiding all the complex interactions between business components.
- Reduce the number of business objects that are exposed to the client across the service layer over the network.
- Hide from the client the underlying interactions and interdependencies between business components. This provides better manageability, centralization of interactions (responsibility), greater flexibility, and greater ability to cope with changes.
- Provide a uniform coarse-grained service layer to separate business object implementation from business service abstraction.
- Avoid exposing the underlying business objects directly to the client to keep tight coupling between the two tiers to a minimum.

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.

The Session Facade abstracts the underlying business object interactions and provides a service layer that exposes only the required interfaces. Thus, it hides from the client's view the complex interactions between the participants. The Session Facade manages the interactions between the business data and business service objects that participate in the workflow, and it encapsulates the business logic associated with the requirements. Thus, the session bean (representing the Session Facade) manages the relationships between business objects. The session bean also manages the life cycle of these participants by creating, locating (looking up), modifying, and deleting them as required by the workflow. In a complex application, the Session Facade may delegate this lifestyle management to a separate object. For example, to manage the lifestyle of participant session and entity beans, the Session Facade may delegate that work to a Service Locator object.

It is important to examine the relationship between business objects. Some relationships between business objects are transient, which means that the relationship is applicable to only that interaction or scenario. Other relationships may be more permanent. Transient relationships are best modeled as workflow in a facade, where the facade manages the relationships between the business objects. Permanent relationships between two business objects should be studied to determine which business object (if not both objects) maintains the relationship.

Use Cases and Session Facades

So, how do you identify the Session Facades through studying use cases? Mapping every use case to a Session Facade will result in too many Session Facades. This defeats the intention of having fewer coarse-grained session beans. Instead, as you derive the Session Facades during your modeling, look to consolidate them into fewer numbers of session beans based on some logical partitioning.

For example, for a banking application, you may group the interactions related to managing an account into a single facade. The use cases Create New Account, Change Account Information, View Account information, and so on all deal with the coarse-grained entity object Account. Creating a session bean facade for each use case is not recommended. Thus, the functions required to support these related use cases could be grouped into a single Session Facade called AccountSessionFacade.

In this case, the Session Facade will become a highly coarse-grained controller with high-level methods that can facilitate each interaction (that is, `createNewAccount`, `changeAccount`, `getAccount`). Therefore, we recommend that you design Session Facades to aggregate a group of the related interactions into a single Session Facade. This results in fewer Session Facades for the application, and leverages the benefits of the Session Facade pattern.

Structure

Figure 1.1 shows the class diagram representing the Session Facade pattern.

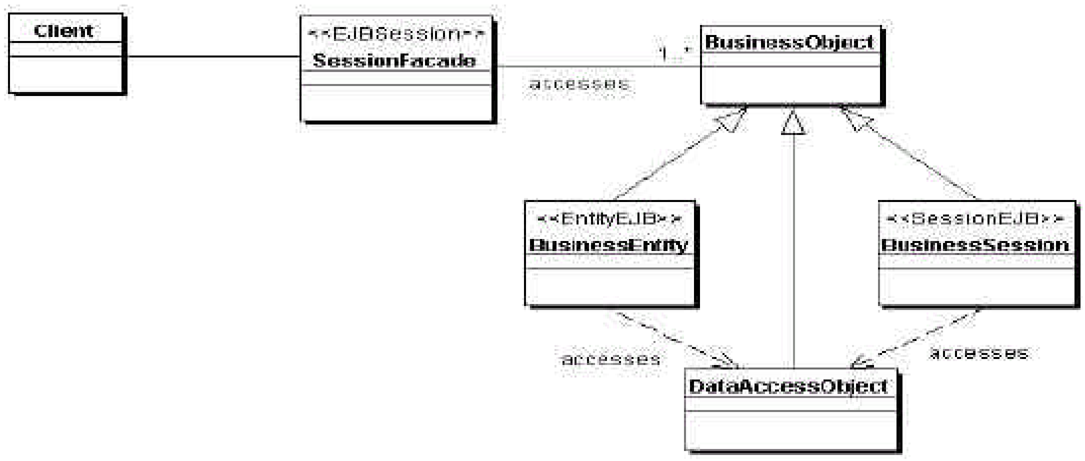

Figure 1.1 Session Facade class diagram.

Participants and Collaborations

Figure 1.2 contains the sequence diagram that shows the interactions of a Session Facade with two entity beans, one session bean, and a DAO, all acting as participants in fulfilling the request from the client.

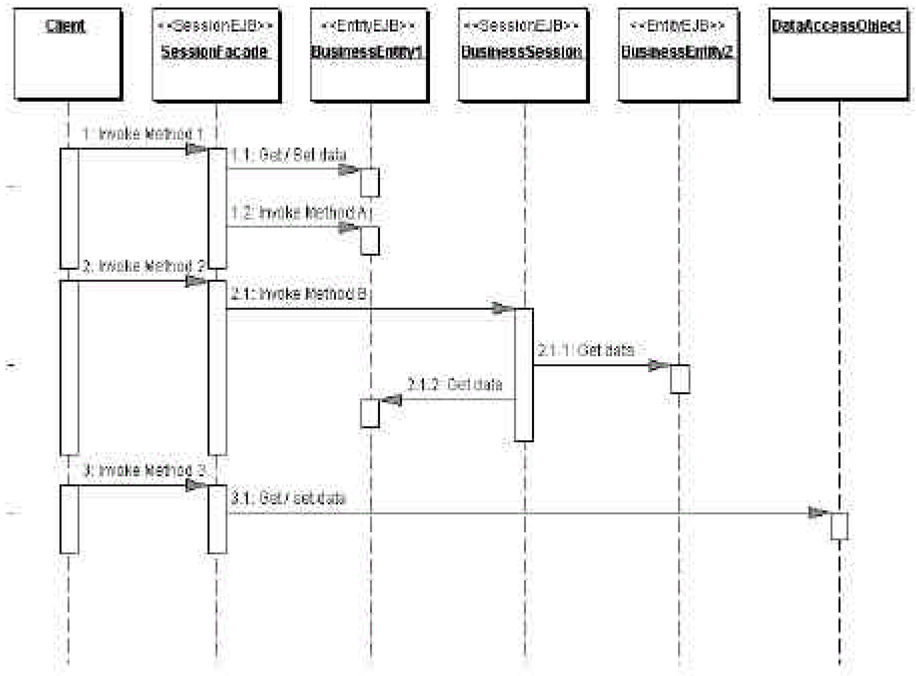

Figure 1.2 Session Facade sequence diagram.

Client

This represents the client of the Session Facade, which needs access to the business service. This client can be another session bean (Session Facade) in the same business tier or a business delegate in another tier.

SessionFacade

The SessionFacade is implemented as a session bean. The SessionFacade manages the relationships between numerous BusinessObjects and provides a higher level abstraction to the client. The SessionFacade offers coarse-grained access to the participating BusinessObject represented by the Invoke invocation to the session bean.

BusinessObject

The BusinessObject is a role object that facilitates applying different strategies, such as session beans entity beans and a DAO (see the next section, “Strategies”). A BusinessObject provides data and/or some service in the class diagram. The SessionFacade interacts with multiple BusinessObject instances to provide the service.