<http://www.oracle.com/technetwork/java/sessionfacade-141285.html>

**Session Facade** : 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.

**Core J2EE Patterns - 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.

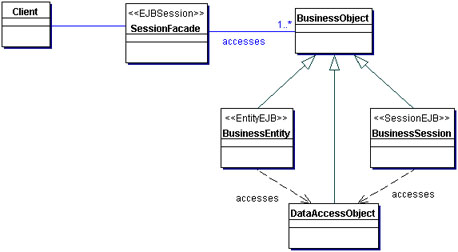
**Problem**

In a multitiered Java 2 Platform, Enterprise Edition (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.

### Structure

Figure 8.15 shows the class diagram representing the Session Facade pattern.

   
**Figure 8.15 Session Facade class 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 (see "Business Delegate" on page 248) 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.

### Strategies

The Session Facade is a business-tier controller object that controls the interactions between the client and the participant business data and business service objects. In a complex application, there may be numerous Session Facades that can intermediate between the client and these objects. You can identify where a Session Facade might be useful by studying the client requirements and interactions typically documented in use cases and scenarios. This analysis enables you to identify a controller layer-composed of Session Facades-that can act as facades for these scenarios.

This section explains different strategies for implementing a Session Facade.

### Session Facade Strategies

#### Stateless Session Facade Strategy

When implementing the Session Facade, you must first decide whether the facade session bean is a stateful or a stateless session bean. Base this decision on the business process that the Session Facade is modeling.

A business process that needs only one method call to complete the service is a nonconversational business process. Such processes are suitably implemented using a stateless session bean.

A careful study of the use cases and scenarios enables you to determine the Session Facade definitions. If the use case is nonconversational, then the client initiates the use case, using a single method in the Session Facade. When the method completes, the use case completes too. There is no need to save the conversational state between one method invocation and the next. In this scenario, the Session Facade can be implemented as a stateless session bean.

#### Stateful Session Facade Strategy

A business process that needs multiple method calls to complete the service is a conversational business process. The conversational state must be saved between each client method invocation. In this scenario, a stateful session bean may be a more suitable approach for implementing the Session Facade.

In both the Stateless Session Facade and the Stateful Session Facade strategies, the business object's role can be fulfilled in different ways, as explained next.

### Business Objects Strategies

You can implement a business object as a session bean, entity bean, DAO, or regular Java object. The following strategies discuss each of these choices.

#### Session Bean Strategy

The business object can be implemented as a session bean. The session bean typically provides a business service and, in some cases, it may also provide business data. When such a session bean needs access to data, it may use a DAO to manipulate the data. The Session Facade can wrap one or more such service-oriented or data-oriented session beans acting as business objects.

#### Entity Bean Strategy

Representing the business object by an entity bean is the most common use of the Session Facade. When multiple entity beans participate in the use case, it is not necessary to expose all the entity beans to the clients. Instead, the Session Facade can wrap these entity beans and provide a coarse-grained method to perform the required business function, thus hiding the complexity of entity bean interactions.

#### Data Access Object Strategy

The Session Facade can directly use one or more DAOs to represent the business data. This is done when the application is so simple that it requires no entity beans, or when the application's architecture is based only on session beans and does not use entity beans. Using DAOs inside session beans partially simulates the persistent nature of entity beans.

The application might need the services provided by an arbitrary Java object (that is, an object that is not an enterprise bean or a DAO, though a DAO can be viewed as a type of arbitrary Java object). In such cases, the Session Facade

**Session facade is one design pattern that is often used while developing enterprise applications.**

**It is implemented as a higher level component (i.e.: Session EJB), and it contains all the iteractions between low level components (i.e.: Entity EJB). It then provides a single interface for the functionality of an application or part of it, and it decouples lower level components simplifying the design.**

Think of a bank situation, where you have someone that would like to transfer money from one account to another.  
In this type of scenario, the client has to check that the user is authorized, get the status of the two accounts, check that there are enough money on the first one, and then call the transfer.

The entire transfer has to be done in a single transaction otherwise is something goes south, the situation has to be restored.

As you can see, multiple server-side objects need to be accessed and possibly modified. Multiple fine-grained invocations of Entity (or even Session) Beans add the overhead of network calls, even multiple transaction. In other words, the risk is to have a solution that has a high network overhead, high coupling, poor reusability and maintainability.