Value Object

Context

Application clients need to exchange data with enterprise beans.

Problem

J2EE applications implement server-side business components as session beans and entity beans. Some methods exposed by the business components return data to the client. Often, the client invokes a business object's get methods multiple times until it obtains all the attribute values.

Session beans represent the business services and are not shared between users. A session bean provides coarse-grained service methods when implemented per the Session Facade pattern.

Entity beans, on the other hand, are multiuser, transactional objects representing persistent data. An entity bean exposes the values of attributes by providing an accessor method (also referred to as a *getter* or *get method*) for each attribute it wishes to expose.

Every method call made to the business service object, be it an entity bean or a session bean, is potentially remote. Thus, in an EJB application such remote invocations use the network layer regardless of the proximity of the client to the bean, creating a network overhead. Enterprise bean method calls may permeate the network layers of the system even if the client and the EJB container holding the entity bean are both running in the same JVM, OS, or physical machine. Some vendors may implement mechanisms to reduce this overhead by using a more direct access approach and bypassing the network.

As the usage of these remote methods increases, application performance can significantly degrade. Therefore, using multiple calls to get methods that return single attribute values is inefficient for obtaining data values from an enterprise bean.

Forces

- All access to an enterprise bean is performed via remote interfaces to the bean. Every call to an enterprise bean is potentially a remote method call with network overhead.
- Typically, applications have a greater frequency of read transactions than update transactions. The client requires the data from the business tier for presentation, display, and other read-only types of processing. The client updates the data in the business tier much less frequently than it reads the data.
- The client usually requires values for more than one attribute or dependent object from an enterprise bean. Thus, the client may invoke multiple remote calls to obtain the required data.
- The number of calls made by the client to the enterprise bean impacts network performance. Chattier applications—those with increased traffic between client and server tiers—often degrade network performance.

Solution

Use a Value Object to encapsulate the business data. A single method call is used to send and retrieve the value object. When the client requests the enterprise bean for the business data, the enterprise bean can construct the value object, populate it with its attribute values, and pass it by value to the client.

Clients usually require more than one value from an enterprise bean. To reduce the number of remote calls and to avoid the associated overhead, it is best to use value objects to transport the data from the enterprise bean to its client.

When an enterprise bean uses a value object, the client makes a single remote method invocation to the enterprise bean to request the value object instead of numerous remote method calls to get individual attribute values. The enterprise bean then constructs a new value object instance, copies values into the object and returns it to the client. The client receives the value object and can then invoke accessor (or getter) methods on the value object to get the individual attribute values from the value object. Or, the implementation of the value object may be such that it makes all attributes public. Because the value object is passed by value to the client, all calls to the value object instance are local calls instead of remote method invocations.

Structure

Figure 1.1 shows the class diagram that represents the Value Object pattern in its simplest form.

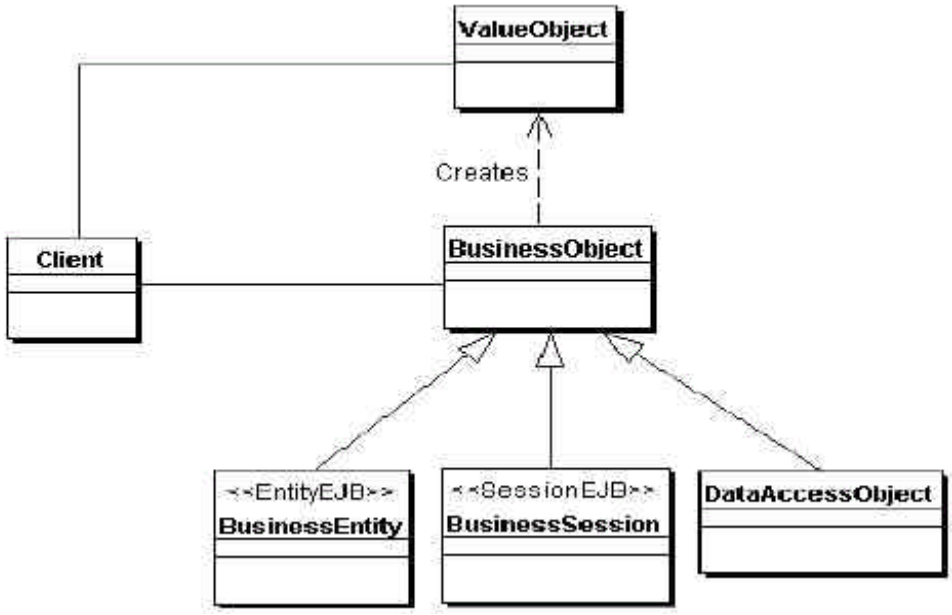

Figure 1.1 Value Object class diagram.

As shown in this class diagram, the value object is constructed on demand by the enterprise bean and returned to the remote client. However, the Value Object pattern can adopt various strategies, depending on requirements. The “Strategies” section explains these approaches.

Participants and Responsibilities

Figure 1.2 contains the sequence diagram that shows the interactions for the Value Object pattern.

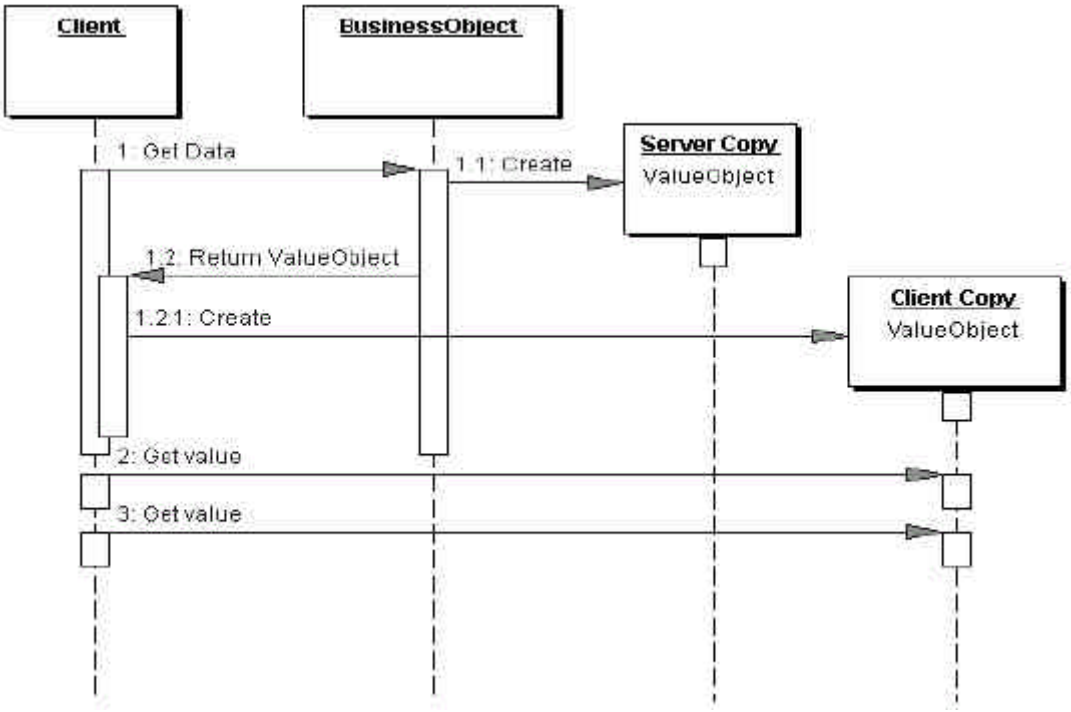

Figure 1.2 Value Object sequence diagram.

Client

This represents the client of the enterprise bean. The client can be an end-user application, as in the case of a rich client application that has been designed to directly access the enterprise beans. The client can be Business Delegates (see “Business Delegate” on page 248) or a different BusinessObject.

BusinessObject

The BusinessObject represents a role in this pattern that can be fulfilled by a session bean, an entity bean, or a Data Access Object (DAO). The BusinessObject is responsible for creating the value object and returning it to the client upon request. The BusinessObject may also receive data from the client in the form of a value object and use that data to perform an update.

ValueObject

The ValueObject is an arbitrary serializable Java object referred to as a value object. A value object class may provide a constructor that accepts all the required attributes to create the value object. The constructor may accept all entity bean attribute values that the value object is designed to hold. Typically, the members in the value object are defined as public, thus eliminating the need for get and set methods. If some protection is necessary, then the members could be defined as protected or private, and methods are provided to get the values. By offering no methods to set the values, a value object is protected from modification after its creation. If only a few members are allowed to be modified to facilitate updates, then methods to set the values can be provided. Thus, the value object creation varies depending on an application’s requirements.

It is a design choice as to whether the value object's attributes are private and accessed via getters and setters, or all the attributes are made public.