Composite Entity

Context

Entity beans are not intended to represent every persistent object in the object model. Entity beans are better suited for coarse-grained persistent business objects.

Problem

In a J2EE application, clients (applications, JSPs, servlets, JavaBeans) access entity beans via their remote interfaces. Thus, every client invocation potentially routes through network stubs and skeletons, even if the client and the enterprise bean are in the same JVM, OS, or machine. When entity beans are fine-grained objects, clients tend to invoke more individual entity bean methods, resulting in high network overhead.

Entity beans represent distributed persistent business objects. Whether developing or migrating an application to the J2EE platform, object granularity is very important when deciding what to implement as an entity bean. Entity beans should represent coarse-grained business objects, such as those that provide complex behavior beyond simply getting and setting field values. These coarse-grained objects typically have dependent objects. A dependent object is an object that has no real domain meaning when not associated with its coarse-grained parent.

A recurring problem is the direct mapping of the object model to an EJB model (specifically entity beans). This creates a relationship between the entity bean objects without consideration of coarse-grained versus fine-grained (or dependent) objects. Determining what to make coarse-grained versus fine-grained is typically difficult and can best be done via modeling relationships in Unified Modeling Language (UML) models.

There are a number of areas impacted by the fine-grained entity bean design approach:

- ***Entity Relationships***—Directly mapping an object model to an EJB model does not take into account the impact of relationships between the objects. The inter-object relationships are directly transformed into inter-entity bean relationships. As a result, an entity bean might contain or hold a remote reference to another entity bean. However, maintaining remote references to distributed objects involves different techniques and semantics than maintaining references to local objects. Besides increasing the complexity of the code, it reduces flexibility, because the entity bean must change if there are any changes in its relationships.

  Also, there is no guarantee as to the validity of the entity bean references to other entity beans over time. Such references are established dynamically using the entity's home object and the primary key for that entity bean instance. This implies a high maintenance overhead of reference validity checking for each such entity-bean-to-entity-bean reference.

- ***Manageability***—Implementing fine-grained objects as entity beans results in a large number of entity beans in the system. An entity bean is defined using several classes. For each entity bean component, the developer must provide classes for the home interface, the remote interface, the bean implementation, and the primary key.

  In addition, the container may generate classes to support the entity bean implementation. When the bean is created, these classes are realized as real objects in the container. In short, the container creates a number of objects to support each entity bean instance. Large numbers of entity beans result in more classes and code to maintain for the development team. It also results in a large number of objects in the container. This can negatively impact the application performance.

- ***Network Performance***—Fine-grained entity beans potentially have more inter-entity bean relationships. Entity beans are distributed objects. When one entity bean invokes a method on another entity bean, the call is potentially treated as a remote call by the container, even if both entity beans are in the same container or JVM. If the number of entity-bean-to-entity-bean relationships increases, then this decreases system scalability due to heavy network overhead.
- ***Database Schema Dependency***—When the entity beans are fine-grained, each entity bean instance usually represents a single row in a database. This is not a proper application of the entity bean design, since entity beans are more suitable for coarse-grained components. Fine-grained entity bean implementation typically is a direct representation of the underlying database schema in the entity bean design. When clients use these fine-grained entity beans, they are essentially operating at the row level in the database, since each entity bean is effectively a single row. Because the entity bean directly models a single database row, the clients become dependent on the database schema. When the schema changes, the entity bean definitions must change as well. Further, since the clients are operating at the same granularity, they must observe and react to this change. This schema dependency causes a loss of flexibility and increases the maintenance overhead whenever schema changes are required.
- ***Object Granularity (Coarse-Grained versus Fine-Grained)***—Object granularity impacts data transfer between the enterprise bean and the client. In most applications, clients typically need a larger chunk of data than one or two rows from a table. In such a case, implementing each of these fine-grained objects as an entity bean means that the client would have to manage the relationships between all these fine-grained objects. Depending on the data requirements, the client might have to perform many lookups of a number of entity beans to obtain the required information.

Forces

- Entity beans are best implemented as coarse-grained objects due to the high overhead associated with each entity bean. Each entity bean is implemented using several objects, such as EJB home object, remote object, bean implementation, and primary key, and each is managed by the container services.
- Applications that directly map relational database schema to entity beans (where each row in a table is represented by an entity bean instance) tend to have a large number of fine-grained entity beans. It is desirable to keep the entity beans coarse-grained and reduce the number of entity beans in the application.
- Direct mapping of object model to EJB model yields fine-grained entity beans. Fine-grained entity beans usually map to the database schema. This entity-to-database row mapping causes problems related to performance, manageability, security, and transaction handling. Relationships between tables are implemented as relationships between entity beans, which means that entity beans hold references to other entity beans to implement the fine-grained relationships. It is very expensive to manage inter-entity bean relationships, because these relationships must be established dynamically, using the entity home objects and the enterprise beans' primary keys.
- Clients do not need to know the implementation of the database schema to use and support the entity beans. With fine-grained entity beans, the mapping is usually done so that each entity bean instance maps to a single row in the database. This fine-grained mapping creates a dependency between the client and the underlying database schema, since the clients deal with the fine-grained beans and they are essentially a direct representation of the underlying schema. This results in tight coupling between the database schema and entity beans. A change to the schema causes a corresponding change to the entity bean, and in addition requires a corresponding change to the clients.

- There is an increase in chattiness of applications due to intercommunication among fine-grained entity beans. Excessive inter-entity bean communication often leads to a performance bottleneck. Every method call to the entity bean is made via the network layer, even if the caller is in the same address space as the called bean (that is, both the client, or caller entity bean, and the called entity bean are in the same container). While some container vendors optimize for this scenario, the developer cannot rely on this optimization in all containers.
- Additional chattiness can be observed between the client and the entity beans because the client may have to communicate with many fine-grained entity beans to fulfill a requirement. It is desirable to reduce the communication between or among entity beans and to reduce the chattiness between the client and the entity bean layer.

Solution

Use Composite Entity to model, represent, and manage a set of interrelated persistent objects rather than representing them as individual fine-grained entity beans. A Composite Entity bean represents a graph of objects.

In order to understand this solution, let us first define what is meant by persistent objects and discuss their relationships.

A persistent object is an object that is stored in some type of data store. Multiple clients usually share persistent objects. Persistent objects can be classified into two types: coarse-grained objects and dependent objects.

A coarse-grained object is self-sufficient. It has its own life cycle and manages its relationships to other objects. Each coarse-grained object may reference or contain one or more other objects. The coarse-grained object usually manages the lifestyles of these objects. Hence, these objects are called dependent objects. A dependent object can be a simple self-contained object or may in turn contain other dependent objects.

The life cycle of a dependent object is tightly coupled to the life cycle of the coarse-grained object. A client may only indirectly access a dependent object through the coarse-grained object. That is, dependent objects are not directly exposed to clients because their parent (coarse-grained) object manages them. Dependent objects cannot exist by themselves. Instead, they always need to have their coarse-grained (or parent) object to justify their existence.

Typically, you can view the relationship between a coarse-grained object and its dependent objects as a tree. The coarse-grained object is the root of the tree (the root node). Each dependent object can be a standalone dependent object (a leaf node) that is a child of the coarse-grained object. Or, the dependent object can have parent-child relationships with other dependent objects, in which case it is considered a branch node.

A Composite Entity bean can represent a coarse-grained object and all its related dependent objects. Aggregation combines interrelated persistent objects into a single entity bean, thus drastically reducing the number of entity beans required by the application. This leads to a highly coarse-grained entity bean that can better leverage the benefits of entity beans than can fine-grained entity beans.

Without the Composite Entity approach, there is a tendency to view each coarse-grained and dependent object as a separate entity bean, leading to a large number of entity beans.

Structure

While there are many strategies in implementing the Composite Entity pattern, the first one we discuss is represented by the class diagram in Figure 1.1. Here the Composite Entity contains the coarse-grained object, and the coarse-grained object contains dependent objects.

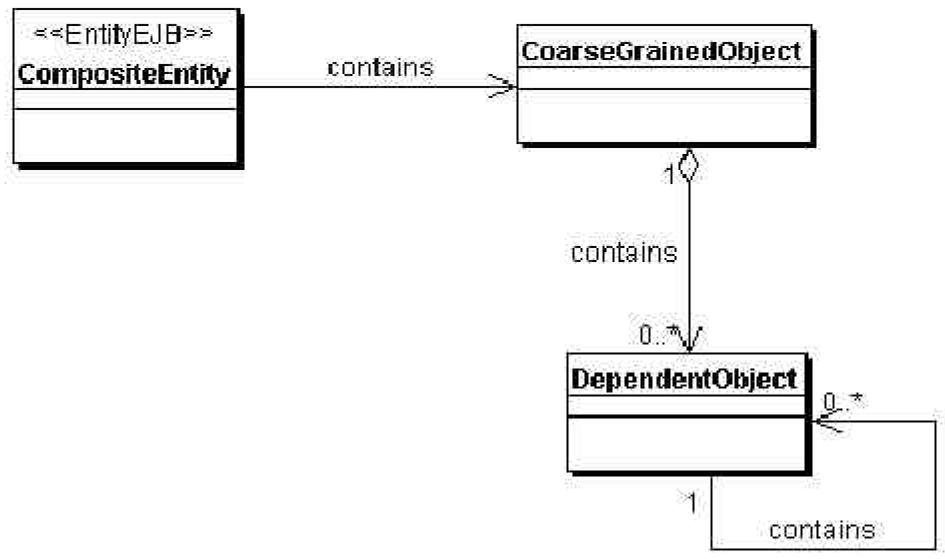

Figure 1.1 Composite Entity class diagram.

The sequence diagram in Figure 1.2 shows the interactions for this pattern.

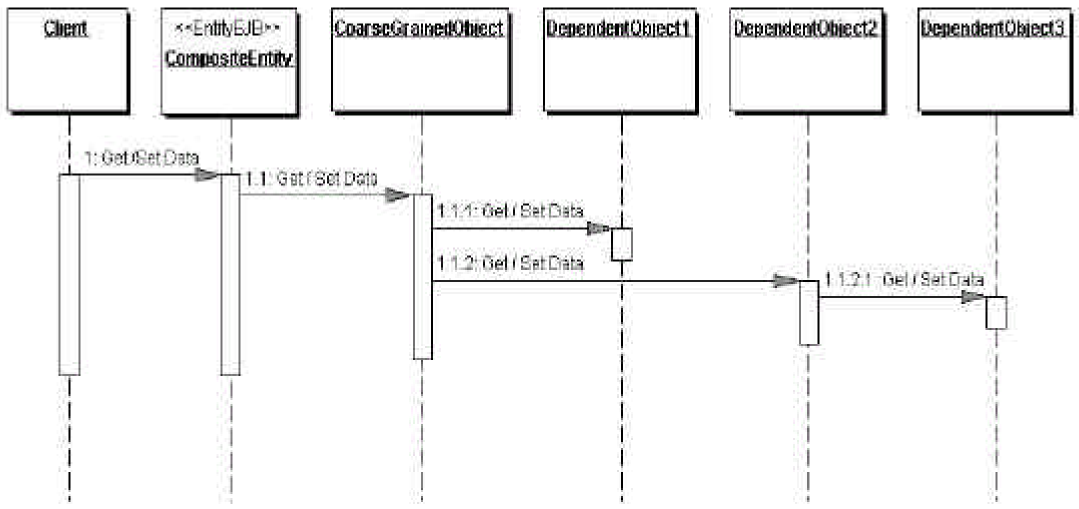

Figure 1.2 Composite Entity sequence diagram.

Participants and Responsibilities

CompositeEntity

CompositeEntity is the coarse-grained entity bean. The CompositeEntity may be the coarse-grained object, or it may hold a reference to the coarse-grained object.

Coarse-Grained Object

A coarse-grained object is an object that has its own life cycle and manages its own relationships to other objects. A coarse-grained object can be a Java object contained in the Composite Entity. Or, the Composite Entity itself can be the coarse-grained object that holds dependent objects. These strategies are explained in the “Strategies” section.

DependentObject1, DependentObject2, and DependentObject3

A dependent object is an object that depends on the coarse-grained object and has its life cycle managed by the coarse-grained object. A dependent object can contain other dependent objects; thus there may be a tree of objects within the Composite Entity.