Best option for flexibility and performance

The Aggregate Entity Pattern Makes a Case for Using BMP

BY KEN MITSUMOTO

Entity beans have been much maligned lately, largely for their inability to scale and difficulty to develop and maintain. However, there remain some compelling reasons to consider using entity beans to persist data in a J2EE application. For example, survivability of entity beans, where state can be recovered following a hard crash, lends to increased reliability when the server is running.

The serialization of state to an alternate store allows entity beans to utilize limited resources more efficiently. That is, activating and passivating beans can allow a server to complete a transaction even when its headroom has been exceeded. A developer can mimic such facilities in session beans, but why reinvent the wheel? Of course, the most significant difference between entity beans and session beans is conceptual, not concrete. Whereas a single session bean instance serves only one client, an entity bean, by virtue of a primary key, serves many clients. Entity beans are more conducive to an architecture where the persistence layer of the application is distributed.

Entity bean evangelism is not the point of this article. The intention here is to promote the use of coarse-grained entity beans when entity beans make sense. A fine-grained bean creates a bijection, or one-to-one relationship, between a bean and a single row in a single table. An architecture built on fine-grained beans requires as many beans as there are tables – not exactly efficient. A coarse-grained bean, on the other hard, draws data from multiple tables. The implication is that a coarse-grained entity bean can persist an entire object graph. Effectively used, the number of entity beans required to model a complex schema can be reduced to a handful. Designing entity beans in this fashion follows the Aggregate Entity pattern, part of the Sun Java Center J2EE Patterns. The benefits of this pattern are explained well on the Sun Java Center site (http:// developer.java.sun.com/ developer/ technicalArticles/J2EE/patterns). Briefly:

- Reduced network traffic between all tiers
- More efficient utilization of resources
- Simplified client interface to data
- Improved manageability

As you would expect, these benefits do not come for free. Those familiar with the two paradigms of entity beans, container-managed persistence (CMP), and bean-managed persistence (BMP), know that the Aggregate Entity pattern cannot be implemented with CMP. For many, this is a deal-breaker. It doesn't have to be.

The release of WebSphere 5.0 has many developers looking forward to using CMP 2.0. The promise of robust persistence and relationship management, coupled with superior performance seem to obviate the need for BMP. However, in terms of flexibility and performance, BMP remains the best option. The server-implemented persistence in CMP is still limited. Relationship management lacks granularity. The performance improvements are only the result of application server vendors implementing a few tricks, such as optimizing data retrieval for finder operations. There is no reason why such optimizations cannot, and in fact, should not, be implemented in a BMP bean. Beyond these shortcomings, CMP's Achilles heel is the inability to draw data from more than one table. The current CMP specification precludes coarse-grained entities. As I have briefly discussed, in a fine-grained architecture, each bean is tied to a single table, rigidly binding the application to a schema. The client interface to the data becomes a mess of remote interfaces and disparate value objects. Any operation that draws data across more than one bean issues a cascade of ejbLoad() and ejbStore() calls, each running roundtrip to the database. The container must allot precious resources to all the bean implementation classes, their remote objects, and primary key classes. The Aggregate Entity pattern resolves these issues by providing one value object, one remote interface, one bean.

ABOUT THE AUTHOR
Ken Mitsumoto is a senior sales engineer for THOUGHT, Inc. (www.thoughtinc.com). He has been working with J2EE technologies since their inception. Ken has a background in mathematics and likes very fast motorcycles.

E-MAIL
ken.mitsumoto@thoughtinc.com

If the Aggregate Entity pattern has a flaw, it is that it is difficult to implement. The difficulty lies in managing an object graph efficiently within the bean. There are two primary hurdles: state management, and relationship management. With proper state management, the bean is always aware of the state of the graph. Because the bean provides an object view of data to its client, the bean is responsible for translating changes to the graph into SQL. The algorithm that resolves these changes should be intelligent enough to issue SQL only for objects that have changed. Efficiency mandates the removal of all unnecessary trips to the database. Moreover, how these selective operations are executed is important. Locking strategies, and graceful exception handling help maintain data validity, another primary directive. Equally challenging, relationship management addresses the issues of how the objects in the graph interact. Granular control over cascade behavior, such as lazy loading, is critical to performance, especially with large datasets. There is no reason to carry around more data than you need. Supporting patterns such as inheritance and polymorphism is conducive to good object-oriented design. The inability to represent these patterns in the persistence layer should not be a constraint on the object model.

I will not discuss the deeper specifics of implementing the Aggregate Entity pattern in this short article. The reasoning is that even with instructions spelled out, use of this pattern would still be too risky for most managers. Entity beans have been branded with two common complaints: "They don't perform," and "They're too hard." The developer's choice is often reduced to one or the other. CMP is easy, but its performance is poor. BMP using the Aggregate Entity pattern performs well, but is difficult. With the current EJB specification, CMP is not a good performance option. The logical alternative is to consider ways to make the Aggregate Pattern easier to implement.

There is a way. Developers are always asking, "What good are Object to Relational mapping tools?" Object graph management in entity beans would be a great answer. Some of these tools even allow developers to generate entire entity beans that implement the Aggregate Entity pattern with the click of the mouse. Moreover, these tools decouple the application from the schema, allowing for new possibilities in design, implementation, and management. If entity beans seem like the right solution for your project, one of these tools can be a low-cost, quick time-to-market solution to implement Aggregate Entity pattern BMPs with the ease of CMP.

AD