



Advertisement: Support JavaWorld, click here!

VoiceWing
broadband phone service**NATIONWIDE CALLING AS LOW AS \$19.9**
FREE SET-UP WHEN YOU ORDER ONLINE
JOIN THE REVOLUTION
LEARN MORE NOW

August 2006

[HOME](#)[FEATURED
TUTORIALS](#)[COLUMNS](#)[NEWS &
REVIEWS](#)[FORUM](#)[JW
RESOURCES](#)[ABOUT
JW](#)

Migrating EJB 2.x applications to EJB 3.0

Gradually and iteratively adopt the benefits of the newest EJB specification

Summary

Enterprise JavaBeans 3.0 is a substantial change from the earlier specifications in terms of both the change in enterprise bean implementation models and in the bean location and call paradigm. How can you migrate legacy EJB code to utilize improvements in the new specification? This article discusses the strategies, both from a design and implementation perspective, for migrating existing EJB applications to the new specification. (1,900 words; **August 14, 2006**)

By **Shashank Tiwari**

Enterprise JavaBeans 3.0 simplifies the enterprise bean architecture and provides enhanced and more powerful features. The new specification leverages the annotations metadata facility introduced in Java 5, persistence and object-relational mapping best practices from tools such as Hibernate and TopLink, and the Dependency Injection pattern made popular by lightweight Java frameworks such as Spring.

This article discusses possible migration strategies for moving applications written using EJB 2.1 or an earlier specification to an EJB 3.0-based architecture. The possible migration paths are evaluated from both the perspectives of design and implementation. This article does not intend to be exhaustive in illustrating the migration options. After reading this article, you should be able to choose the best option, within your own specific context, for migrating the legacy EJB code to the new specification.

This article assumes you are familiar with the enterprise bean, Java 5, and object-relational mapping features and concepts.

EJB 2.1 to EJB 3.0: What has changed?

To provide a context for this article's discussion of possible migration paths, I begin by discussing the changes in the new specification in the context of each of the different bean types and then at a generic level pertinent across multiple bean types.

Session bean

In EJB 2.1 and earlier specifications, two interfaces—the home and the local, or remote, business interfaces—and the bean implementation class were required for each session bean. The home interface was required to extend the `EJBHome` or the `EJBLocalHome` interface and declare the lifecycle method, such as `create()`. The local, or remote, business interface was required to extend the `EJBObject` or the `EJBLocalObject` interface and declare the business methods. The bean implementation class itself was an `EnterpriseBean` type and, in the case of session beans, extended the `SessionBean` sub-interface. Callback method implementations in the bean class had to be provided so that the container could trigger them on occurrence of the appropriate lifecycle events. In addition, critical elements of the bean, including its transaction and security definition, and whether it was stateful or stateless, were defined in the associated deployment descriptors.

Listing 1 illustrates an example of a stateful session bean defined using the EJB 2.1 specification.

Listing 1. An EJB 2.1-based banking service stateful session bean

```
public interface BankingService extends EJBObject {

    public void deposit(int accountId, float amount) throws
RemoteException;

    public void withdraw(int accountId, float amount) throws
RemoteException;

    public float getBalance(int accountId) throws RemoteException;

    public void doServiceLogout() throws RemoteException;
}

public interface BankingServiceHome extends EJBHome {
    public BankingService create() throws CreateException,
RemoteException;
}

public class BankingServiceEJB implements SessionBean {

    public void deposit(int accountId, float amount) throws
RemoteException {
        //Business logic to deposit the specified amount and update the
balance
    }

    public void withdraw(int accountId, float amount) throws
RemoteException {
        //Business logic to withdraw the desired amount and update the
balance
    }

    public float getBalance(int accountId) throws RemoteException {
```

```
//Business logic to get the current balance
}

public void doServiceLogout() throws RemoteException {
//Service completion and logout logic
}

public void ejbCreate(){}
public void ejbActivate(){}
public void ejbPassivate(){}
public void ejbRemove(){}
public void setSessionContext(SessionContext context){}
}
}
```

In the EJB 3.0 specification, a session bean needs to define only a business interface and a bean implementation class. The home interface has been removed. The business interface is a regular Java interface, also sometimes called *POJI*, or the plain-old Java interface. The business interface does not need to extend the `EJBObject` or the `EJBLocalObject` interface; instead, if required, it could be defined within the interface hierarchy that represents the business domain model.

The bean implementation class is a regular Java class, also sometimes called a *POJO*, or a plain-old Java object. It does not implement an `EnterpriseBean` type. The declaration and the configuration in the deployment descriptor can be defined within the Java code, using the annotations metadata facility. In addition, default values are provided for most configurations, thus minimizing the bean-specific configuration requirements. Under the new specification, one could deploy session beans without any `ejb-jar.xml` deployment descriptors, though they still exist and could be used if the developer prefers that to the annotations model.

In the case of EJB 3.0 session beans that implement a Web service, the methods exposed as Web service operations are annotated with the `WebMethod` descriptor. Session beans that serve as Web service endpoints are annotated as a `WebService`.

Listing 2 illustrates the earlier example (from Listing 1) of the stateful session bean using the EJB 3.0 specification.

Listing 2. An EJB 3.0-based banking service stateful session bean

```
@Remote
public interface BankingService {
    public void deposit(int accountId, float amount);
    public void withdraw(int accountId, float amount);
    public float getBalance(int accountId);
    public void doServiceLogout();
}

@Stateful
public class BankingServiceBean implements BankingService {
```

```
    public void deposit(int accountId, float amount) {
        //Business logic to deposit the specified amount and update the
balance
    }

    public void withdraw(int accountId, float amount) {
        //Business logic to withdraw the desired amount and update the
balance
    }

    public float getBalance(int accountId) {
        //Business logic to get the current balance
    }

    @Remove
    public void doServiceLogout () {
        //Service completion and logout logic
    }

}
```

Message-driven beans

In EJB 2.1, a message-driven bean class implemented the `MessageDrivenBean` interface and the message listener interface. The callback methods were implemented in the bean class and the container, on a particular event called the corresponding method. Message-driven beans never involved the concept of a home and a remote, or local, interface.

In EJB 3.0, the `MessageDriven` annotation is used to mark and specify a message-driven bean. The deployment descriptor can also be used to specify a bean as message-driven. Thus, it is not required for the bean class to implement the `MessageDrivenBean` interface. The business interface of a message-driven bean is the message listener interface that corresponds to the message type that the bean is a listener for. In the case of Java Message Service, `javax.jms.MessageListener` is the message listener interface or the business interface. The bean class needs to implement the message listener interface or annotate the message listener interface using the `MessageDriven` annotation.

The new specification supports callback methods (`PostConstruct` and `PreDestroy`), provides the Dependency Injection pattern for access to resources, and allows interceptor method definition for message-driven beans.

Entity beans

EJB 2.1 and earlier specifications required the definition of a home and remote interface, and the implementation of the bean class for an entity bean, similar to that of a session bean type. For a container-managed persistence (CMP) and container-managed relationship (CMR) scenario, the mappings and definitions were specified in the deployment descriptors.

The EJB 3.0 specification radically changes the earlier concepts and implementation of entity beans from heavyweight enterprise bean objects, with home and remote/local interfaces, to a lightweight persistent domain object in line with the object-relational mapping concepts. A later discussion on migration in this article discusses and details some of these changes.

More changes—relevant to multiple bean types

The EJB 3.0 specification introduces an important change in the way enterprise beans are accessed and called. The earlier service locator pattern that utilized JNDI (Java Naming and Directory Interface) lookups is now replaced with the Dependency Injection pattern. Thus, the complexity around JNDI API usage is removed from the bean developer's and client's viewpoint.

EJB 3.0 also introduces the concept of interceptors. Interceptor methods intercept a business method invocation or a lifecycle event callback. Interceptor methods could be defined either on the bean class or on the interceptor class associated with the bean or both.

Listing 3 illustrates the usage of `AroundInvoke`, an interceptor annotation.

Listing 3

```
@Stateful
public class BankingServiceBean implements BankingService {

    public void deposit(int accountId, float amount) {
        ....
    }

    ....
    ....
    ....
    ....

    @Remove
    public void doServiceLogout () {
        ....
    }

    @AroundInvoke
    public Object auditBankingService(InvocationContext inv) throws
Exception {

        try {

            Object nextEntry = inv.proceed();

            //Log the method name and parameters by using getMethod() on
the InvocationContext instance.
            //The getMethod() returns the method of the bean class for
which the interceptor was invoked.
            //The return type for getMethod() is a Method class, the
reference to which can be used to get additional
            //method-related information. Target and context data can
also be obtained using the InvocationContext instance.

            return nextEntry;

        } catch (Exception ex) {
```

```
        //Exception handling code goes here.  
    }  
}  
  
}
```

Migrating to EJB 3.0

The EJB 3.0 (Java Specification Request 220) expert group specifically identified easy migration as an objective and hence provided for both backward compatibility and interoperability between enterprise bean components written in the old and the new specifications. This also makes it possible for EJB 2.x and earlier beans to be gradually and incrementally modified to utilize the EJB 3.0 specification. Interestingly enough, an entire chapter titled "Compatibility and Migration" is included in the EJB 3.0 simplified API specification document.

Some of the possible migration strategies are discussed below.

New functionality in EJB 3.0 with existing code in the old specification

An existing EJB application can be deployed in a container that supports EJB 3.0. The old bean components could be deployed and used without any modification.

EJB 2.x components could call EJB 3.0 components and vice-versa. Some scenarios where the client and server parts of the bean components are of different specifications are discussed below.

EJB 2.x and earlier clients of EJB 3.0 components

Enterprise bean components, which need to work with EJB 2.1 and earlier specifications, don't need to be written using old specifications any more. EJB 3.0 beans can utilize metadata annotations to make them work with older clients that expect to access them using the home and the remote interface. The methods required as per older specifications are mapped to corresponding methods in the enterprise bean written using the EJB 3.0 specification. As an example, the `create()` method as desired in the old specification could now map to a method that initializes the bean. This example was presented by Linda DeMichiel, one of the specification leads, as part of her [Java One 2005 presentation on EJB 3.0](#).

EJB 3.0 clients of EJB 2.x components

EJB 3.0 beans can access EJB 2.1 and earlier beans. Dependency injection is utilized to inject the EJB 2.1 component references. JNDI lookup calls are avoided. Once a handle is gained to the injected EJB home interface, the `create()` method is called to instantiate and subsequently utilize the bean.

Listing 4 presents an example where an EJB 3.0 client accesses an EJB 2.1 component.

Listing 4

```
....  
....  
  
@EJB BankingServiceHome bsHome;  
BankingService bs = bsHome.create();  
....
```

```
    bs.deposit(....);  
    ....  
  
    bs.getBalance(....);  
    ....  
  
    bs.doServiceLogout();  
    ....  
....
```

Migrating EJB 2.1 session beans to EJB 3.0

The EJB 2.1 session beans that implement the `SessionBean` interface become POJOs under EJB 3.0. The business interface that extended `EJBObject` or `EJBLocalObject` can now extend a business interface that makes sense in the particular context. The home interface is not required any more and hence can be ignored or removed. The unused lifecycle methods can be removed.

A session bean could be specified as stateless or stateful using annotations (`Stateless` and `Stateful`) or by the previous means of specifying the state in the deployment descriptors. For stateful session beans, the removal methods are marked with the `Remove` annotation.

EJB 3.0 clients access the other EJB components with the help of dependency injection. The `EJB` annotation can be used to specify the enterprise bean component.

Migrating EJB 2.1 entity beans to EJB 3.0

EJB 2.1 entity beans that implement the `EntityBean` interface become POJOs under EJB 3.0. A CMP and CMR entity bean in EJB 2.1 has getter/setter methods in line with the JavaBeans paradigm. In EJB 3.0, the Persistence API defines metadata annotations to define persistence and relationship criteria on the lines of object-relational mapping concepts. The home and local interfaces are not required for entity beans in EJB 3.0.

In the new specification, an entity bean or an entity object is simply a POJO with the `Entity` annotation. `NamedQuery` and `NamedQueries` annotations are used to annotate the finder methods. The `Column` annotation maps persistent entity bean properties to the corresponding database table columns, and the identifier or primary key for a persistent entity is defined using the `Id` annotation. As in the case of object-relational mapping tools, the property name is assumed to be the same as the column name if the `Column` annotation is not explicitly defined. The table name and schema is specified using the `Table` annotation. Each property value has getter/setter methods defined for it. Properties that are not persistent are annotated as `Transient`.

CMR is specified using the `OneToMany`, `OneToOne`, `ManyToMany`, and `ManyToOne` annotations, depending on the multiplicity and the directionality of the relationships.

In EJB 3.0, the Persistence API defines an `EntityManager` class in line with similar classes in object-relational mapping tools such as Hibernate. The lifecycle management (creation and removal) and search of a persistent entity is done using the `persist()`, `remove()`, and `find()` method calls on the `EntityManager` class.

A phased approach


The migration from an earlier EJB specification to the EJB 3.0 specification can be planned in a phased manner to minimize risk and any possible adverse impact on the existing applications that interact with these beans. The discussions above clearly illustrate that multiple strategies can be

utilized for a piecemeal implementation.

Automating the migration

The migration to EJB 3.0 involves removal of the redundant home interface definitions, removal of the empty callback implementations, replacement of the JNDI lookups with metadata annotations to facilitate dependency injection, and modification of the business interface and the bean class implementation. It also involves conversion of deployment descriptor configurations to annotations within the code. Many of these tasks can be automated. Third-party companies and IDE vendors already have offerings to automate part of the migration effort.

Conclusion

Migration to EJB 3.0 is a fairly uncomplicated task and can be carried out in a phased and piecemeal manner. Some of the migration tasks can be automated, and tools and IDEs can be leveraged to ease the process. With the current emphasis on backward compatibility and ease of migration, now may be the best time to move EJB applications to EJB 3.0. As the EJB specification continues to evolve, moving legacy EJB code (from version 2.1 and earlier versions) to the newer specifications may grow more difficult. 

About the author

[Shashank Tiwari](#) (a.k.a. Shanky) is a software architect at [Saven Technologies](#), an information technology company based in Illinois. He has, for more than seven years, been involved with architecting high-performance distributed applications using J2EE. He advises investment banking and financial service companies in building robust quantitative, data intensive, and scalable software applications. He is also an ardent supporter of open source software. He is one of the maintainers of the Python GUI framework called anygui. He lives with his wife and two sons in New York. More information about him can be accessed at www.shanky.org.

Resources

- JSR 220, the Enterprise Java Beans 3.0 specification—there are three documents in the specification (EJB 3.0 Simplified API, Java Persistence API, and EJB Core Contracts and Requirements):
<http://www.jcp.org/en/jsr/detail?id=220>
- "Enterprise JavaBeans 3.0," 2005 JavaOne Session TS-7969, Linda DeMichiel, EJB 3.0 Specification Lead, Sun Microsystems:
<http://developers.sun.com/learning/javaoneonline/2005/coreenterprise/TS-7969.pdf>
- "Migration to EJB 3.0: Getting There Is Easier Than You Think," 2005 JavaOne Session TS-3795, Mike Keith and Merrick Schincariol, Oracle:
<http://developers.sun.com/learning/javaoneonline/2005/coreenterprise/TS-3795.pdf>
- "The New EJB 3.0 Persistence API," 2005 JavaOne Session TS-7949, Linda DeMichiel (Sun Microsystems), Gavin King (JBoss), Mike Keith (Oracle), and Patrick Linskey (SolarMetric):
<http://developers.sun.com/learning/javaoneonline/2005/coreenterprise/TS-7949.pdf>
- For an introduction to EJB 3.0, read "Simplify Enterprise Java Development with EJB 3.0," Michael Yuan (*JavaWorld*):
 - Part 1: Use annotations to develop POJO services (August 2005):
<http://www.javaworld.com/javaworld/jw-08-2005/jw-0815-ejb3.html>
 - Part 2: POJO persistence made easy (September 2005):
<http://www.javaworld.com/javaworld/jw-09-2005/jw-0912-ejb.html>
- For more articles on EJB, browse the **Enterprise JavaBeans** section of *JavaWorld's* Topical Index:
http://www.javaworld.com/channel_content/jw-ejbs-index.shtml?

Advertisement: Support JavaWorld, click here!



VoiceWing
broadband phone service

NATIONWIDE CALLING AS LOW AS \$19
FREE SET-UP WHEN YOU ORDER ONLINE
JOIN THE REVOLUTION
LEARN MORE NOW [▶](#)

[HOME](#) | [FEATURED TUTORIALS](#) | [COLUMNS](#) | [NEWS & REVIEWS](#) | [FORUM](#) | [JW RESOURCES](#) | [ABOUT JW](#) | [FEEDBACK](#)

Copyright © 2006 JavaWorld.com, an IDG company