Gartner

Research

Publication Date: 27 October 2009 ID Number: G00171637

Enterprise JavaBeans' Future: Getting Simpler, More Ubiquitous, but Not Dominant

Massimo Pezzini

Enterprise JavaBeans (EJB), part of the popular Java Platform, Enterprise Edition (Java EE) set of standards, is aimed at supporting implementations of server-side, transactional application logic. Despite being extensively used by enterprise developers, it often has been criticized for its complexity. This research is targeted to CIOs, CTOs, enterprise technical planners and architects, application development managers, project leaders and senior developers. They will find insights about the evolution of the EJB technology and its prospects for growth in adoption through simplification and integration into products other than the classic Java EE application servers.

Key Findings

- EJB technology is frequently used for transaction processing, especially high-end, applications; but it hasn't gained mainstream adoption because of its complexity.
- Alternative, grassroots, open-source technologies, like Spring and Hibernate, have achieved greater popularity because many Java programmers find their approaches to transactional application development friendlier than EJB.
- The most recent EJB versions (the established 3.0 and the upcoming 3.1, which includes the Java Persistence API [JPA] 2.0 specification) are easier to deal with, in part because they have absorbed many Spring and Hibernate characteristics.
- EJB 3.1/JPA 2.0 technology will be incorporated in non-Java-EE-compliant application servers and will grow in adoption for midlevel and high-end application projects. But Spring and Hibernate will remain popular and viable options.

Recommendations

- Java EE users should look at EJB 3.x/JPA as a significant productivity improvement over EJB 2.x. Focus on JPA to grab productivity "low hanging fruits." Experiment with EJB 3.1 when compliant products are available to check for additional productivity improvements.
- Users and vendors that adopted Spring and Hibernate can safely continue with these technologies, but should plan for continuing convergence of Spring/Hibernate with EJB/JPA technologies.

WHAT YOU NEED TO KNOW

EJB technology adoption will grow significantly during the next five years because of the greater ease of use of EJB 3.1/JPA 2.0, and because it will be extensively supported in non-fully Java-EE-compliant products. However, Spring and Hibernate will remain popular frameworks for transactional applications and viable alternatives to EJB and JPA. Users and independent software vendors that have concerns regarding the necessity of dealing with multiple, partially overlapping alternatives will increasingly find it easier to tackle these challenges because EJB and Spring/Hibernate will keep converging, driven by continuing cross-pollination, equal support from vendors, and extensive coexistence and integration in the same platform middleware products.

ANALYSIS

EJB is one of the key features of the Java EE set of standards driven by Sun Microsystems through the open Java Community Process (JCP). The EJB standard defines a component model to support the implementation of applications' server-side business logic. It specifies capabilities such as transaction and concurrency control, security, component deployment, integration with naming and directory services, object persistency and intra-application communication mechanisms.

How EJB Fits With Java EE

In the Java EE model, applications are split into a front end and a back end. The user-facing, front-end logic runs in the "Web container" (or "servlet container") supporting the JavaServer Pages (JSP) and servlet standards. The user-facing logic, in turn, calls data-centric, back-end logic implemented in the form of "enterprise beans." These are application components developed according to the EJB standard and running in the "EJB container," a platform middleware supporting the EJB application programming interfaces (APIs; plus other capabilities like J2EE Connector Architecture). The front end and the back end interoperate through Java EE-defined intra-application communication mechanisms, including local calls, Java Remote Method Invocation-Internet Inter-Orb Protocol (RMI-IIOP), Web services and Java Message Service (JMS) asynchronous messages.

Years of practical experience in Java development have shown that there is a wide range of applications — especially in user-centric, Web scenarios — for which an EJB container is not necessary. Therefore, some extremely popular Java application server products, such as Apache Tomcat or Jetty from Intalio/Webtide, only implement the Web container and other aspects of the Java EE standard (e.g., Java Database Connectivity and JMS), but not the full platform. The implementation of only a Web container is not sufficient to achieve formal Java EE certification. To be certified by Sun as "compliant," a product must implement *all* the Java EE specifications (including EJB) and must successfully execute a relevant test suite. Hence, any product certified as Java EE 5 (or earlier versions)-compliant fully implements — by definition — the EJB specifications. Therefore, there are many popular EJB container products available as part of open-source (e.g., JBoss and GlassFish) and closed-source (e.g., IBM's WebSphere Application Server or Oracle WebLogic Server) Java EE application servers.

Despite such strong support from the vendor community, EJB technology never fully won ample mainstream adoption because it was deemed too complex by most Java developers. Alternative server-side application component models — such as the Spring Framework (a sort of simplified version of EJB session beans; see "Spring: Past, Present and Future") and Hibernate (an alternative to EJB entity beans for managing object persistency) — gained notable popularity.

The successful and massive utilization of these technologies demonstrated the practical validity of some of the concepts they introduced. JCP took advantage of these experiences and introduced Spring/Hibernate-derived concepts, such as annotations and dependency injection, in the EJB 3.0 specification (part of Java EE 5 released in 2007), with the aim of making enterprise beans radically easier to develop and less cumbersome to deploy.

Java EE 5 and EJB 3.0

EJB 3.0 supports three approaches to developing server-side business logic:

Session Beans: These are callable application logic components that may have transactional properties. For example, the execution of a method in a session bean can be declared as atomic (all or nothing). EJB 3.0 session beans can be stateless and stateful:

- The state (that is, the value of its internal variables) of a *stateless session bean* is lost when a method execution ends and the reply is sent back to the caller. It is possible for multiple callers (clients) to share the same instance of a stateless session bean, as isolation is enforced by the container.
- The state of a *stateful session bean* instead is maintained. When a stateful session bean is called a second time, the client application will find the bean state as it was left at the end of the first execution, and so on. However, an instance of a given stateful session bean can be accessed by only one client that has exclusive ownership of it.

Message-Driven Beans (MDBs): MDBs are similar to session beans, but their execution is triggered by an asynchronous JMS message, rather than via a synchronous method call. An MDB doesn't expose a callable interface, but subscribes to a JMS queue or topic and executes code to process the incoming messages (one at a time).

JPA: This provides an object/relational mapping API, largely inspired by the Hibernate and TopLink technologies, which enable Java programmers to manipulate relational tables through their representation in the form of corresponding Java classes. JPA has replaced entity beans, types of EJBs defined by earlier versions (2.x) of the specification. For backward compatibility, entity beans are still supported, and products must implement them to obtain Java EE 5 certification. Although formally part of EJB 3.0, JPA can stand on its own, and JPA applications don't require a full EJB container or Java EE 5 server to run.

EJB 2.x defined an XML deployment descriptor file associated with any EJB and containing information about actions to take at deployment time. In EJB 3.0, deployment instructions are given to the EJB container in the form of Java annotations (sort of "comments" in the Java source code). For backward compatibility and other practical reasons, the deployment descriptor is still supported.

Java EE 6, EJB 3.1 and EJB Lite

The JCP is now working on Java EE 6, which is likely to be officially released by the end of 2009 (a Java EE 6-compliant version of the GlassFish Application Server, the open-source Java EE reference implementation, will be generally available at the same time as the new specification; other compliant products will be available within the following 12 to 24 months). The main new EJB (3.1) features that will be introduced in Java EE 6 are:

Singleton Beans: Similar to stateful session beans, but they can be shared across multiple, concurrent clients. There is only one instance of a singleton bean per application and per Java Virtual Machine (JVM) that responds to all method calls performed by all its clients. Concurrency can be managed by the EJB container or by the singleton bean.

Asynchronous Method Call: This enables a client to call a method of a session bean (stateless, stateful or singleton), regain control without waiting for the reply and then synchronize with the reply at a later time. (This is different from MDBs, which *never* provide a reply.) A given session bean can support a mix of synchronous and asynchronous methods. An asynchronous method call is an atomic transaction in its own right or it never participates in a transaction.

Global Java Naming and Directory Interface (JNDI) Names: Until EJB 3.1, the assignment of global JNDI names to EJBs was implemented differently by vendors: The same application deployed on different containers from different vendors might have different JNDI names for its session beans, which created portability problems. EJB 3.1 standardizes global JNDI names for session beans, so that they are the same across different EJB 3.1 containers.

Improved Timer Service: The timer service was introduced in EJB 2.1 so that methods can be invoked on occurrence of well-defined timer events (or "timers"). EJB 3.1 allows for the implementation of timers via annotations, in addition to the original programmatic mechanism. Moreover, EJB 3.1 timers are global across multiple JVM (in EJB 2.1, timers only worked on a single JVM) and their granularity is much finer and more flexible.

EJB Lite: This defines a subset of EJB 3.1 features: local stateless, stateful and singleton session beans, transactions, security and others. The main capabilities not supported by EJB Lite include remote methods calls, MDBs, timers and asynchronous method calls. EJB Lite provides a lightweight transactional application logic container for the Web profile (see below). It aims to make not only EJB programming more accessible for novices, but also the EJB technology more widely supported in open- and closed-source products by enabling the implementation of EJB containers less complex to develop by vendors than those needed to support the full EJB 3.1.

Embeddable EJB Container: This is an EJB container that can be instantiated on a minimal Java Platform, Standard Edition (Java SE) environment, without requiring the full Java EE capabilities. The standard requires at least EJB Lite support, but vendors can decide to implement the complete EJB 3.1 set in their embeddable EJB containers.

Simplified EJB Packaging: In Java EE, an application is packaged for deployment in an Enterprise ARchive (EAR) file. This includes a Web Application Archive (WAR) module (which keeps the code to be deployed in the Web container) and, if needed, an EJB Java Application Archive (JAR) module, including the application EJBs. In EJB 3.1, EJBs can be directly stored in the WAR file, although this is suggested for only simple applications using a small number of EJBs.

In addition to EJB 3.1, Java EE 6 adds improvements to JPA (2.0) and to Servlets (3.0), including an asynchronous method call mechanism similar to the one supported by EJB 3.1 session beans.

Java EE 6 also introduces profiles: functionally meaningful, and JCP approved, subsets of the whole Java EE 6. Products will be able to obtain a Java EE 6 certification (specific for a given profile), even if they don't support the full capabilities of the platform. At this time, Java EE 6 only specifies the Web Profile (Servlet 3.0, EJB 3.1 Lite, JPA 2.0, Java Transaction API [JTA] 1.1, JSP 2.2, Java Server Faces [JSF] 2.0, and other features) that extends the "classic" Web container with some EJB capabilities to support a wide class of transactional applications not requiring the power and complexity of the full EJB 3.1.

Adoption Will Grow, but EJB Will Not Dominate

Although EJB 3.0 was positively received by the Java developers community as a significant improvement over EJB 2.x, it was perceived by many experienced Java programmers just as an attempt to catch up. Nevertheless, JPA is rapidly gaining ground (even Hibernate supports it). JPA has eroded much of the Hibernate differentiation over entity beans; therefore, alternative open-source (like EclipseLink, DataNucleus, Open JPA) or closed-source (like Oracle TopLink) JPA-based products are being adopted more frequently, especially by developers that want to focus on a standard API to avoid vendor or product lock-in issues.

The Java developer community's reactions to EJB 3.1 drafts echo those to EJB 3.0: An improvement, but not big enough to leapfrog Spring. Nevertheless, we expect that when EJB 3.1 is available in the most popular application server products, adoption will be significant for several reasons:

- Developers (especially those implementing business-critical, heavy-duty back-end applications) that have not yet endorsed Spring, but have marginally used EJBs so far, will find EJB 3.1 more compelling than its predecessors because of its greater ease of development and deployment.
- By being an open "paper" standard, there will be many competing implementations (like there are for JPA), thus the lock-in risk will be perceived to be lower for EJB 3.1 than for Spring.
- Embeddable EJB containers will favor integration of EJB technology in extreme transaction processing/cloud-oriented Java (but not Java EE-compliant) application servers — such as GigaSpaces eXtreme Application Platform (XAP), Appistry CloudIQ Platform or Paremus Service Fabric. Thus, they will promote technology convergence between mainstream Java EE application servers and the most advanced generation of platform middleware, further favoring EJB 3.1 usage at the high end of the transactional application spectrum.
- EJB 3.1 use for relatively simple, opportunistically oriented transactional applications will be favored by the appearance in the market of new Web Profile-compliant products, an incremental evolution of classic Web containers and a natural "downsizing" of established full Java EE products.

However, EJB 3.1/JPA 2.0 adoption will not grow to the point of displacing Spring/Hibernate:

- Spring and Hibernate have been in the market for several years (much more than EJB 3.0), have huge and growing installed bases, can count on a very large and loyal community of developers, and are embedded or supported in a wide variety of products.
- Spring popularity will further grow because of its integration with the OSGi standard, a technology that is rapidly gaining momentum in the vendor community (see "OSGi: Enabling an 'SOA-Inside' Approach to Application Infrastructure Middleware").

VMware recently proposed acquisition of SpringSource (the company behind the Spring technology), which may create concerns about the future Spring "openness." If VMware fails to reassure the industry about Spring's future as an open and open-source technology, then EJB 3.1 may have a chance of being adopted more widely, given the large choice of open-source and closed-source implementations that will be available in the market. However, Java and Java EE (including EJB) may also face credibility challenges, particularly if Oracle fails to reassure the industry about the future of these technologies as soon as the pending acquisition of Sun is completed (see "Oracle's Acquisition of Sun Could Change Java's Course").

Recommendations

For Java EE Users: Focus on JPA, as it is the evolution of EJB that will have more impact in the industry. JPA is drastically easier to deal with than the old entity beans. As soon as practical, test whether EJB 3.1's anticipated productivity improvements can be obtained in your development organization. Look at these standards as the most vendor-neutral platform for transactional Java applications.

Users and Vendors Adopting Spring/Hibernate: Safely proceed with these technologies, but plan for their continuing convergence and coexistence with EJB and JPA.

RECOMMENDED READING

"Trends in Platform Middleware, 2009: Facing the Cloud Upheaval"

"Magic Quadrant for Enterprise Application Servers"

"Spring: Past, Present and Future"

"OSGi: Enabling an 'SOA-Inside' Approach to Application Infrastructure Middleware"

"Oracle's Acquisition of Sun Could Change Java's Course"

REGIONAL HEADQUARTERS