JavaBeans and Enterprise JavaBeans: What's the difference?

Mike Day
IBM Object Middleware Marketing Team
July 1999

Editor's Note: The following article is based on a roundtable discussion. Participants included Ken Burgett from the IBM Component Broker beta support team, and Liane Acker, Jim Knutson and David Morrill from the IBM Enterprise Java Beans department.

You're probably already using JavaBeans today and don't know it. There are no restrictions to having JavaBeans on your desktop if you have a Java-enabled browser. The Web pages you use may have beans as part of an applet. Soon, you will interact with JavaBeans as the visual portion of a browser, then those JavaBeans will interface to an EJB on the server. This capability also extends to both the Internet and intranets.

A JavaBean and Server Bean, more commonly known as an Enterprise JavaBean (EJB), have some basic similarities. They are objects or components created with a set of characteristics to do their own specific job. They also have the ability to take on other characteristics from the container on the server in which they currently reside. This enables a bean to behave differently, depending on the specific job and environment where it is placed.

This opens up a tremendous business possibility. Because JavaBeans are platform-independent, solution providers in the future will be able to readily market their client-side JavaBeans to a wide audience without having to create or maintain different versions. These JavaBeans can be paired with EJBs performing business functions such as ordering, credit-card processing, electronic transfers, inventory allocation, shipping, and so on. There is great potential here, and it is the kind of potential Component Broker (part of WebSphere Application Server, Enterprise Edition) is designed to deliver.

A JavaBean is a component that has interfaces in it or properties associated with it so it can be interrogated by and integrated with other beans developed by different people at different times. You can build a bean and tie it together later with other beans at construction time. This process provides a way to build something and use it again later; that's the notion of a component. This single application can be deployed as a stand-alone program, as an ActiveX component, or within a browser.

A JavaBean is different from a pure object because of its external interface, the properties interface. This interface allows a tool to read what the component is supposed to do, hook it up to other beans, and plug it into another environment. JavaBeans are intended to be local to a single process, and they are often visible at runtime. This visual component may be a button, list box, graphic or chart—but it is not required.

Executable components
Server beans or EJBs are executable components or business objects deployed on the server. They have a protocol that allows them to be accessed remotely and installed or deployed on a particular

server. They have a set of mechanisms that allow them to delegate major factors of service—security, transactional behavior, concurrency (the ability to be accessed by more than one client at a time) and persistence (how their state can be saved)—to the container in which they are placed on the EJB server. They get their behavior when installed in a container, which provides the different qualities of service, so selecting the right EJB server is critical. This is where IBM WebSphere Enterprise Edition excels.

An EJB is a nonvisual, remote object designed to run on a server and be invoked by clients. An EJB can be built out of multiple, nonvisual JavaBeans. They have a deployment descriptor that is intended for the same purpose as JavaBean properties: It's a description of the bean that can be read later by a tool. EJBs also are platform independent; once a bean is written, it can be used on any platform that supports Java—clients and servers, too.

Because they are generated by a toolset such as IBM VisualAge for Java, EJBs are server-based objects and are intended to be called remotely. They are installed on the EJB server and get a remote interface to call just as other CORBA remote objects are called.

An ActiveX object
A JavaBean can be deployed as an ActiveX object, but while a *proxy* to an EJB can do this, an EJB itself cannot be an ActiveX object, as ActiveX runs on the desktop. To do this on a platform-dependent, Windows-only platform, a developer could have the JavaBean rendered as an ActiveX component.

Benefits
The major benefit of an EJB is that the bean developer can stipulate, when the bean is built, what kind of behavior is needed, but not how it's done. Development is in two parts: The programmer develops the bean and verifies that it works with the build tools and includes a deployment descriptor that identifies the kinds of quality-of-service behaviors needed. In the next step, another programmer can take that bean and use a deployment tool that reads the EJB deployment descriptor and installs the bean into a container on an enterprise Java server. In this second step, the deployment tool takes some action—this may mean generating codes such as state saving codes, putting in transactional hooks or doing security checks. All this action is generated by the deployment tool; the bean developer and deployer can be different people.

Any platform-independent JavaBean can be adapted, through the use of a deployment tool, into a platform-specific EJB with the correct qualities of services available to meet the specific requirements of existing business systems and applications. This is why an EJB server is so important to integrating systems, networks and architectures.

EJBs and IBM WebSphere Enterprise Edition
As used in IBM WebSphere Enterprise Edition, EJBs can be configured as a managed business object. The container they delegate services to is the Component Broker container in which they are installed. The persistence part of an EJB is mapped in what we call the Data or State Object. The EJB server provides different qualities of service for the EJB, and choosing the right EJB server may be critical to meeting the complete business requirements. The Component Broker function is extremely robust, providing high level functions such as workload balance and support across multiple machines in a server group. It has system management capabilities that go well beyond what the Enterprise Java Server (EJS) specifications call for. Therefore, a JavaBean or EJB written to the basic standards can run on WebSphere Enterprise Edition using the Component Broker function and gain all that additional value.

There are unique features and service qualities provided by the EJB server, and they are not all the same. IBM Component Broker has some powerful features—for example, scalability lets the developer deploy EJBs across many types of servers, from small systems to large networks. Developers can start small, for example, in a single department, deploying first on a Java server on a LAN, knowing the JavaBeans and EJBs created there can be deployed on a global network whenever ready. Then, the developer can test it, get the feel of it, run a pilot, do a prototype, and so on. When satisfied with it, the developer can scale it up dramatically by moving it to a high-performance server. The JavaBeans and EJBs are not constrained by any computer architectural boundaries. They are written in Java and can run on anything with a Java virtual machine available, and any Enterprise Java Server (EJS) can be used to deploy the object. So developers can build on what's convenient today, deploy on what's convenient later, and it doesn't have to be the same machine or even the same kind of machine.

IBM WebSphere Enterprise Edition supports deploying a business object across multiple servers. EJBs are integrated into the Component Broker function as business objects and are handled as any other business object. Therefore, the EJBs can connect to the back-end system of choice to do whatever is necessary to meet their business needs. That becomes the persistence infrastructure that Component Broker provides for the EJB. Developers will be able to keep using current legacy systems and provide them with an e-business interface by utilizing Component Broker as an EJB server.

For an EJB to work in the WebSphere Component Broker environment, you can use the Component Broker deployment tool to install it on one or more servers, and then add it to the naming server so it can be found universally. Anyone with access to the common naming server can find it, can find the home and can execute a method on the home, creating an EJB. This is exactly what you do with Component Broker.

An example
Let's take the example of an electronic shopping cart seen on a catalog-shopping Web site. Your cart is a JavaBean. You fill it up with items from the catalog shelves and these items are JavaBeans themselves. It is all visual and user-oriented. When you check out, the items in your cart are sent to an EJB on the server that executes the business operations necessary for checking the credit card for authorization and available credit, producing a packing slip or special directions to the shipping department for which items to pull and where to send them—all the activities your business programs are already doing.

Summary
The whole thrust is not so much the capabilities of beans as it is the competitive potential they can provide your business. IT architects and application developers now can focus their attention strictly on business logic and leave the infrastructure—such as transactions, persistence and security—up to the server. WebSphere's Component Broker function provides all this—and back-end access—with the object-transaction manager.

About the Author
Mike Day is a member of IBM's Object Middleware Marketing Team. He conducts roundtables that focus on topics of general interest suggested by our readers. Roundtable guests are selected based on their relative background, experience or immediate activities in the topic area. To suggest a topic, write to us at compbrok@us.ibm.com

What do you think of this article?

○ Killer! ○ Good stuff ○ So-so; not bad ○ Needs work ○ Lame!

Comments?

Submit feedback

What do you think of this article?

○ Killer! ○ Good stuff ○ So-so; not bad ○ Needs work ○ Lame!

Comments?

Submit feedback