

Aspects of Enterprise Java Beans

Gregory Blank, Gene Vayngrib
Information Builders, Inc.
Two Penn Plaza, New York, NY 10021-2898
email: (pgmgfb, pgmglv)@ibi.com
<http://www.ibi.com/>

May, 20, 1998

Abstract

Enterprise Java Beans (EJB), a specification for a Java component framework recently released by Sun Microsystems, immediately attracted attention of several major software vendors, including IBM, Oracle and Sybase. Analysts agree that EJB has a potential to replace CORBA as a standard for enterprise level applications. Aspect Oriented Programming fits naturally into EJB paradigm, but requires a few enhancements to be useful in this environment.

Enterprise Java Beans

EJB ties together other specs from Sun, including Java Transaction Service (JTS), Java Naming and Directory Interface (JNDI), Java Database Connectivity (JDBC) and Java Messaging Service (JMS). It can be seen as a modern version of CORBA, made more accessible through the use of a single language and modern component methodologies and patterns. A mapping to CORBA was released along with the very first version of EJB.

The main concepts are:

- EJB server
- EJB container
- EJ Beans
- EJB JAR files
- Deployment descriptors

Accordingly, the following roles are differentiated for developers:

- server provider
- container provider
- bean provider
- bean deployer and assembler

EJB server is a Java component/application server that houses containers and provides services to them: connection and thread management, etc. Containers house EJ beans and are able to take over transactions, persistence and security that the beans do not wish to

handle themselves. Containers can be specialized for beans with special needs. The spec differentiates EJ beans of two types: non-persistent session beans and persistent entity beans, corresponding to records in a database, CICS transactions or other remote sources. EJB JAR file is a zip Java archive that provides an export/import standard for the EJ beans. Besides classes themselves it contains various meta-information files, including deployment descriptors, transaction and security attributes in the form of serialized Java classes.

The spec describes the contracts between EJ Beans and containers, and promises to add container-server contracts.

A typical application bean would consist of:

- implementation classes
- a remote interface; required by the spec RMI-style interfaces, the only public “face” of the bean
- a home interface; also required, factory and finder methods
- a deployment descriptor(DD)

The responsibilities of container include generation of classes for access to the bean (EJB Object), for lifecycle operations (EJB Home), and possibly remote stubs.

The spec also mentions the possibility of modification of the application code by the container at deployment time.

AOP and EJB

There is a lot of conceptual affinity between the goals of EJB and those of AOP. Both technologies strive to separate the functional from non-functional issues, to disentangle the roles a typical programmer has to assume, and constantly switch between, during the code development. The recipes are seemingly different: EJB takes the road of dynamic composition, where container plays Big Brother to the component, controlling the traffic, imposing constraints, handling transactions on the component’s behalf. AOP, on the other hand, leaves the classes to communicate freely but *brainwashes* them first by modifying the code to perform the necessary calls and actions. It is clear that both approaches are complimentary and can be used together to enhance the result.

The integration of AOP into the EJB framework will solve some of the EJB’s problems:

- EJB Objects can be generated using much richer RIDL portals instead of RMI interfaces
- EJB persistence can be implemented transparently for the EJ Beans via *persistence aspects*, mapping instance variables to database fields, calling JDBC methods and keeping track of changes
- Coordination aspects (COOL) are a natural way to capture EJB container coordination policies. As a matter of fact, EJ beans are not allowed to create and delete threads

On the other hand,

- AOP, as essentially a framework of definition languages, should be independent of the actual implementation of transactions, distribution, persistence or security. Aspects would plug into EJB containers to obtain access to various services

Aspect code (distribution, transaction, coordination, persistence) will be supplied separately from the application code and then *weaved in* at the deployment or run- time. A serialized Java *aspect descriptor* containing the names of the aspect files, and all the information necessary for *weaving*, might look something like this:

```
public class AspectDescriptor extends DeploymentDescriptor {
    String AspectSource;
    Collection TargetFiles;

    public void setAspectSource( String name) {}
    public String getAspectSource() {}

    public void setAspectType( String type) {}
    public String getAspectType() {}

    public Enumeration getTargetFiles {}
    public void addTargetFile {}
    public String removeTargetFile {}
}
```

The Aspect Weaver itself will be a part of the container, or maybe a service provided by the EJB server.

Aspect code could also be a part of the container itself, rather than supplied with the application. This would enable easy and elegant enforcement of laws and policies. Or better yet, the JAR-based aspects may inherit from the container-based aspects.

This approach would obviously require from AOP a number of features that it does not currently have:

- Dynamic bytecode-level weaving, a natural extension to the current AOP technology at least as far as Java is concerned. (Is the name based weaving the only possibility?)
- Aspect genericity. The presence of actual names of classes and variables in the aspect code restricts the range of applicability of this code. An introduction of parameterized types, regular expressions or functions returning names would allow *template* aspects to be written and applied to a wider domain of application code. For example, a container resident aspect source may specify the logic to be added to every constructor or finalize method, or a well-defined subset of them.

Enterprise Component Broker

Enterprise Component Broker from IBI is one of a few currently available EJB implementations. In fact, with the spec still changing rapidly, no one can claim full compliance.

ECB is a Java component framework and application server designed to complement a component development tool, like Symantec's VisualCafe or IBM's VisualAge by providing an integrated environment for assembly, deployment, testing and execution of enterprise applications.

The main features include:

- CORBA compliance. ECB is a CORBA 2.0 compliant object request broker
- Web integration. ECB is either itself a Web server, or works alongside another Web server through HTTP servlets. The servlets are also EJ Beans, complete with properties, events, bean interconnections, locating through JNDI, security, transactions and persistence.
- Visual Configurator, a browser based GUI tool allows to deploy, test and control the applications on servers across the network
- Rapid prototyping. Simple applications can be developed locally and then immediately tested in a distributed environment
- Application partitioning. The ECB application space is partitioned into projects. A project is a named persistent entity (currently saved in LDAP via JNDI) that can be instantiated on an ECB server, which is itself a project. Each project has its own namespace, cache and access control. An instance of project determines the boundaries of distribution transparency: as long as the calls stay within the project limits the application need not be changed when some objects become remote. The project paradigm also facilitates versioning and live update.
- Dynamic code generation. Client and server stubs, EJB home and object interfaces, event listeners and sources, component attributes and policies – all the code is generated by the ECB either at the deployment or during the run time.
- Enterprise integration. The application beans and JDBC drivers bundled in the shipped version provide seamless access to CICS and IMS transactions as well as data in 70 databases on 35 platforms.

ECB is available for download from www.ibi.com

We believe that ECB provides an ideal test-bed for application of AOP to Enterprise Java Beans. We would be willing to commit the necessary resources to make this happen.

References

1. Enterprise Java Beans. Version 1.0, Sun Microsystems, March, 1998
java.sun.com/products/ejb
2. Lopes, C. D: a Language Framework for Distributed Programming. Ph.D. Thesis, Northeastern University, 1997