Using Java Objects and Services for Database Business Applications

Author: **Dănuț - Octavian Simion**, Lumina – The University of South-East Europe, Bucharest, Romania, danut.simion@lumina.org

The paper presents the facilities advantages of using Enterprise Java Objects in Business Applications and emphases aspects like simplicity, application portability, component reusability, ability to build complex applications, separation of business logic from presentation logic, easy development of Web services, deployment in many operating environments, distributed deployment, application interoperability, integration with non-Java systems and development tools. Enterprise JavaBeans - EJB is architecture for developing, deploying, and managing reliable enterprise applications in different environments. The Java Platform, Enterprise Edition and the EJB architectures provide superior support for Web-based enterprise database applications and makes easy to build custom interfaces for the end user.

Keywords: *Enterprise JavaBeans; Business objects; Java objects integration; Java Platform; transactions; Web services; database objects*

Introduction

Java objects represented by the Enterprise JavaBeans component architecture is the most important part of the J2EE platform. The core of a J2EE application is comprised of one or several enterprise beans that perform the application's business operations and encapsulate the business logic of an application. Other parts of the J2EE platform, such as the JSP, complement the EJB architecture to provide such functions as presentation logic and client interaction control logic. EJB is a standard for building server-side components in Java. An enterprise bean can compose one or more Java objects because a component may be more than just a simple object [4], [5]. These required methods allow the EJB container to manage beans uniformly, map the database objects regardless of which container beans are running in. In other words, Enterprise beans are components that are used as parts of distributed enterprise database applications.

The main features of Java objects

Java objects represented by enterprise beans are not entirely remote objects. If a client wants to use an instance of an enterprise bean class, the client do not invokes the method directly on an actual bean instance. The invocation is intercepted by the EJB container and then delegated to the bean instance. By intercepting requests, the EJB container automatically performs implicit middleware [1], [4]. Some of the services that are available at the point of interception include:

- Distributed transaction management. Transactions enable you to perform robust, deterministic operations in a distributed environment by setting attributes on your enterprise beans. The transaction service is exposed through the Java Transaction API - JTA. The JTA is a high-level interface that is used to control transactions;

- Persistence is a natural requirement of any deployment that requires permanent storage. EJB offers assistance here by automatically saving persistent object data to an underlying storage and retrieving that data at a later time;

The server side has different kinds of needs than GUI clients do. Server-side components need to run in a highly available, fault-tolerant, transactional, and multiuser secure environment. The application server provides this high-end server-side environment for the enterprise beans that map the database objects, and it provides the runtime containment necessary to manage enterprise beans. Specifically, EJB is used to help write logic that solves business problems. Typically, EJB components, enterprise beans can perform any of the following tasks:

- Perform business logic. Examples include applications like shopping cart, ensuring that the manager has authority to approve the purchase order, or sending an order confirmation e-mail using the Java-Mail API;
- Access a database. Enterprise beans can achieve database access using the Java Database Connectivity-JDBC API;

EJB components are not presentation tier components and they sit behind the presentation tier components or clients and do all the most of the business logic. Examples of the clients that can connect to enterprise beans include the following:

- Dynamically generated Web pages. Web sites who are transactional and personalized in nature need their Web pages generated specifically for each request. Core technologies such as Java servlets and JavaServer Pages (JSP) are used to dynamically generate such specific pages. Both servlets and JSPs live within a Web server and can connect to EJB components that map the database objects, generating pages differently based upon the values returned from the EJB layer;
- Web Service clients. Some business applications require no user interface at all. They exist to interconnect with other business partners' applications that may provide their own user interface.

The bean provider supplies business components, or enterprise beans. Enterprise beans are not complete applications, but rather are deployable components that can be assembled into complete solutions. The bean provider could be an internal department providing components to other departments.

Frequently the application assembler who is usually a developer or systems analyst is not familiar with these issues. Those who will deploy the EJB are aware of specific operational requirements and perform the tasks above. The developer that will deploy the EJB has the freedom to adapt the beans, as well as the server, to the environment in which the beans are to be deployed [3], [4].

EJB defines three different kinds of enterprise beans:

- Session beans. Session beans model business processes. They are like verbs because they perform actions. The action could be anything, such as adding numbers, accessing a database, calling a legacy system, or calling other enterprise beans. Examples include a workflow engine, a catalog engine, a credit card authorizer, or a stock trading engine;
- Entity beans. Entity beans model business data from database objects. They are like nouns because they are data objects and are Java objects that cache database information. Examples include a product, an order, an employee, a credit card, or a stock. Session beans typically harness entity beans to achieve business goals, such as a stock-trading engine - session bean that deals with stocks - entity beans;

The utilities of Java objects in Business components for database applications

The usage of Enterprise JavaBeans that map the database objects can be detailed in an example of Entity Beans and Details Object which models

a component in a production system and which supports the remote client view. This component of system has the attributes comp number, comp description, and price of the component. The following source code shows the home interface of the Enterprise Bean Comp:

```
package example_obj1;
import javax.ejb.*;
public abstract class CompObj implements EntityBean {
   private EntityContext theContext;
   private CompDetails theDetails;

   public CompObj () {}
   //The create method of the home interface

   public String ejbCreate(String compNumber)
      throws CreateException
   {
      setCompNumber(compNumber);
      theDetails = new CompDetails();
      theDetails.compNumber = compNumber;
      return null;
   }

   public void ejbPostCreate(String compNumber)
      throws CreateException
   {}
   public abstract void setCompNumber(String num);
   public abstract String getCompNumber();
   public abstract void setCompDescription(String desc);
   public abstract String getCompDescription();
   public abstract void setPrice(float p);
   public abstract float getPrice();
```

```
//The method of the remote interface
public void setCompDetails(CompDetails cd) {
   setCompDescription(cd.getCompDescription());
   setSupplierName(cd.getSupplierName());
   setPrice(cd.getPrice());
   theDetails = cd;
}
public CompDetails getCompDetails() {
    return theDetails;
}
//The methods of the javax.ejb.EntityBean interface
public void setEntityContext(EntityContext ctx) {
   theContext = ctx;
}
public void unsetEntityContext() {
   theContext = null;
}
public void ejbRemove()
   throws RemoveException
{}
public void ejbActivate() {
}
public void ejbPassivate() {
}
public void ejbLoad() {
   if(theDetails == null) {
      theDetails = new CompDetails();
   }
   theDetails.setCompNumber = this.getCompNumber();
   theDetails.setCompDescription =
                  this.getCompDescription();
   theDetails.setSupplierName = this.getSupplierName();
   theDetails.setPrice = this.getPrice();
```

```
  }

  public void ejbStore() {
  }}
```

The persistence manager ensures that after the method call the time stamp and other data of the Enterprise Bean are made persistent. The method ejbCreate initializes the time stamp, while the method ejbLoad sets the time stamp according to the persistent state. The time stamp is not public data of the Comp bean. The following source code shows the deployment descriptor of the Comp bean:

```
<?xml version="1.0" ?>
<ejb-jar                                    version="2.1"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">
  <enterprise-beans>
    <entity>
      <ejb-name>CompObj</ejb-name>
      <home>ejb.comp.CompHomeEx</home>
      <remote>ejb.comp.CompEx</remote>
      <ejb-class>ejb.comp.CompBeanEx</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.String</prim-key-
class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>CompBeanEx</abstract-
schema-name>
      <cmp-field>
        <description>comp number</description>
```

```
        <field-name>compNumber</field-name>
      </cmp-field>
      <cmp-field>
        <description>comp description</description>
        <field-name>compDescription</field-name>
      </cmp-field>
      <cmp-field>
        <description>comp price</description>
        <field-name>price</field-name>
      </cmp-field>
      <primkey-field>compNumber</primkey-field>
    </entity>
  </enterprise-beans>
  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>CompObj</ejb-name>
        <method-name>setCompDetails</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

This solution prevents clients from overwriting each other's changes. If the EJB container uses several instances of an Enterprise Bean identity for parallel processing of a method call in various transactions, it synchronizes the various instances through a call to the methods ejbLoad and ejbStore. A client will have to implement a relatively complex error handling mechanism for the case of an exception of type OutOfDateException [1], [6].

Conclusions

Java objects represented by Enterprise JavaBeans are the most important components of J2EE business applications and these include the business logic and can be used through interfaces in the Presentation Tier. These types of beans are specific for the Business logic and provide the java objects that represent the results from the clients requests that are also processed according with the business requirements of the database applications [2], [5]. In business applications the EJB work together on a Java platform through classes, interfaces, services and data types definition that map the database objects, so the design and implementation of those are very important for the interface designers, web designers, java programmers, the business consultants and the integration software architects and also allows dissociation between different modules of these database applications. The clients work with one exposed component interface and their requests are managed by the Java container, which can contain parts of enterprise database applications and access system resources such as database objects or other enterprise beans. Java objects are easy to work with, platform independent and can be used in different parts of business database applications.

References

[1] Dale Nilsson, *Accessing SAP Objects with Visual Age for Java*, eye-on-objects.com, 2009
[2] Mark Johnson, *Turn Java classes into JavaBeans*, www.javaworld.com, 2010
[3] Mark Johnson, *Bean Markup Language*, www.javaworld.com, 2010
[4] Robert Englander, *Developing Java beans*, O'Reilly, 2011
[5] Sandor Spruit, *Reflection on Java, Beans and relational databases,* www.javaworld.com, 2012

[6] Sun Microsystems, Inc., getting Java Business Integration Vision, Sun Microsystems, Inc., 2010

URI: http://java.sun.com/javase/technologies/desktop/javabeans/
URI: http://java.sun.com/docs/books/tutorial/javabeans/
URI: http://www.sun.com/software/javaenterprisesystem/
URI: http://www.javarules.org/
URI: http://java.sun.com