Enterprise Java Beans

Mikhail Chalabine
mikch@ida.liu.se

EJB Intro

- An EJB is a distributed server-side non-visual component
  - Multiple address spaces
  - Distributed objects
  - Transactional access to remote objects
- EJB is a part of the J2EE standard
  - javax.ejb package
  - Component specification
  - Programmer implements a set of interfaces from the EJB API
- Implementation by independent vendors
  - Tools and Containers
    - Proprietary: IBM (WebSphere), BEA (WebLogic), Sun and Netscape (iPlanet), Oracle, Borland
    - Open source: JBoss (www.jboss.org)

EJB Intro cont.

- Separate business logic from middleware services:
  - networking
  - transactions
  - persistence
  - logging
  - resource pooling
- EJB Container / Application server
  - Manages beans
  - Provides middleware services
- Clients: JSPs, Servlets, Java applications, and other EJBs

EJB Yesterday and Today

- 2.0
- 3.0
- The goal of Enterprise JavaBeans (EJB) 3.0 is to simplify development of Java applications and standardize the persistence API for the Java platform.
- EJB 3.0 is a part of the next major revision of the J2EE platform, J2EE 5.0.
2.0 vs. 3.0

- **Simplified EJB**
  - EJB 3.0 eliminates the need for *home* and *component interfaces* and the requirement for bean classes for implementing `javax.ejb.EnterpriseBean` interfaces. The EJB bean class can be a pure Java class often referred as *POJO* and the interface can be a simple business interface. The bean class can implement the business interface.

- **Interceptors**
  - An interceptor is a method that intercepts a *business method* invocation.
  - An interceptor method may be defined in a Stateless Session Bean, Stateful Session Bean or an interceptor class may also be used instead of defining the interceptor method in the bean class.

- **Simple JNDI lookup of EJB**
  - Lookup of EJB has been simplified and clients do not have to create a bean instance by invoking `create` method on EJB and can directly invoke a method on the EJB.

- **Use of Annotations Instead of Deployment Descriptors**
  - Metadata annotation is being used as an alternative to deployment descriptors.
  - Annotations can be used to specify bean types, different attributes such as transaction or security settings, O-R mapping and injection of environment or resource references.
  - Deployment descriptor can be used to override metadata annotations.
**Middleware**

- **Explicit middleware** (e.g. CORBA):
  - Write to API
  - Difficult to write, maintain and support
- **Implicit middleware** (e.g. EJB)
  - Write isolated business logic
  - Declarative middleware service specifications
  - Middleware services generated automatically
  - Tool support

**Distributed Objects**

Client

Network

Remote interface

Distributed object
Distributed Objects

- Client
- Stub
- Distributed object
- Skeleton
- Remote interface
- Network

Distributed Objects using EJBs

- Client
- Stub
- Skeleton
- Distributed object
- EJB Object
- Bean
- Remote interface
- Network

Execution flow

- Client calls a method on the EJB object
- EJB object delegates the call to a bean
- EJB receives the result
- EJB passes the result to the caller

Mikhail Chalabine mikch@ida.liu.se
Enterprise Java Beans (EJB)
EJB 2.0

- **Home interface**
  - Defines the life cycle methods of the bean
    - Create
    - Destroy
- **Remote interface**
  - Defines the business methods of the bean
- **Bean class**
  - Business logic

**EJB Object (Remote Interface)**

- Extends javax.ejb.EJBObject
- Defines business methods clients call
- Acts as a proxy

```java
package ejbExample.interfaces;

/* This is a remote interface for HelloBean */
public interface Hello extends javax.ejb.EJBObject {
    public String Hello() throws java.rmi.RemoteException;
}
```
EJB Home Object (Home Interface)

- Client
- Stub
- Network
- EJB Object
-.Bean
- Skeleton
- EJB Home
- EJB Container

EJB Home Object Characteristics

- Extends javax.ejb.EJBHome
- Acts as a factory to create EJB instances
- Allows clients to create/remove/find EJBs

```java
package ejbExample.interfaces;

/**<HelloBean's home interface*/
public interface HelloHome extends javax.ejb.EJBHome {
    Hello create() throws java.rmi.RemoteException,
                        javax.ejb.CreateException;
}
```

Summary: EJB Architecture

- Enterprise Bean class
- Supporting classes
- EJB Object
- Remote interface
- Home object
- Deployment descriptor (XML)
- Vendor-specific files
- (Local interface)

Summary: an EJB consist of

- EJB-jar file
Deployment

- EJB deployment descriptor (XML)
- ejb-jar.xml
- Attributes of the beans specified declaratively
- Deployment descriptor language is a composition language
- EJB-jar file is verified by container
- Container generates stubs and skeletons

How clients find the Home object

- Java Naming and Directory Interface (JNDI)
  - Similar to CORBA naming service
  - Mapping between resource names and physical locations
- No machine address to home object hard coded
  - Address to JNDI server is needed
  - Kept in the initial context
  - Use initial context factory to acquire an initial context (is the JNDI driver)
  - Vendor specific, bound to J2EE server implementation

EJB Architecture

Types of Beans

- Session beans
  - Stateless
  - Stateful
- Entity beans
- Message-Driven beans

```java
Context ctx = new InitialContext();
HelloWorld home = (HelloWorldHome) PortableRemoteObject.narrow(ctx.lookup("HelloWorld"), HelloWorldHome.class);
```
So, what does the container do?

- Generates stubs and skeletons
- Creates EJB instances as needed.
- Persists entity beans.
- Handles security and transactions

How can container vendors compete?

- Caching strategies
- Development tool integration
- Database access optimization
- Performance

XDoclet

- Deployment descriptor
- Generate from declarative specification
  - Remote interface
  - home interface
  - local interface
  - local home interface
  - primary key class
- Specification as comments in the Bean class

Demonstration

CUGS
Mikhail Chalabine
mikch@ida.liu.se
Local interfaces

- When beans call beans locally
- Optimization
- Call by value/reference problem

Entity Beans

- Represent business data stored in database
- Database types converted to Java types
- Change of values in the Entity Bean is propagated to the DB

Entity Beans: Persistence

- Bean managed persistence (BMP)
- Container managed persistence (CMP):
  - Object to relational database mapping (common)
  - Object databases (uncommon)
  - Container generates persistence as subclass
  - EJB-QL, query language
- An entity bean is a view into a data source, e.g., a database

Entity Beans: Façade design pattern
**Entity Beans: Security**

- Authentication - JAAS
- Authorization
- Deployment descriptor
  - Roles
  - Roles and methods
- No instance level based security

**Message-Driven Beans (MDB)**

- Don't have home, remote or local interfaces
- Have a single business method:
  - onMessage
- No static type check
- No return values
- No exceptions
- Stateless

**Why Message-Driven Beans?**

- Performance
- Reliability
- Support for multiple senders and receivers
- “Easy” integration to legacy systems

**Demonstration**

CUGS
Mikhail Chalabine
mikch@ida.liu.se
Final thoughts

- Is it object-oriented?
  - Separation of data and operations (entity beans and session beans)
  - No inheritance between beans in 2.0!
    - 3.0 Standard: beans are POJOs
- Suitable for which tasks?
  - One architecture. Anomalies if trying to do anything else
- Component marketplace?
  - Not today!

Possible problem sources

- Remotability
  - Pramatics: stay away from large distributed systems
- Security
- Persistence
- Caching
- Scalability
- Messaging
- Transactions

Resources

- Szyperski, chapter 14
- Sun EJB tutorial
- Ed Roman: Mastering EJB
  - http://www.theserverside.com/books/wiley/masteringEJB/index.jsp
- JBoss, Open source EJB Container
  - http://www.jboss.org

What we have to do

- Create a project
- Create an EJB
- Generate the EJB-related files
- Create a servlet and a web application
- Generate the servlet-related files
- Create a J2EE Application
- Package the application (jar, war)
- Configure JBoss servlet container and launch
- Deploy the application
Create a New Project

- File > New > Project > JBoss-IDE > J2EE
- Projects > J2EE Project
  - Name: CUGS-EJB
  - Create folder src
  - Set default output to /EJB-CUGS/bin

EJB

- File > New > Other > JBoss-IDE > EJB
- Components > Session Bean
- Set package to cugs-ejb.ejb and the class to MyBean.
- Make sure ejbCreate() is selected
- Click create, note all the method stubs are generated with the default ejbCreate() method

EJB Add business method

- Right click on the MyBean class under the MyBean Java file
- J2EE > Add Business Method
- Enter getName as the method name and String for the return type
- Add the implementation to the method

```java
/**
 * Business method
 * @ejb.interface-method view-type = "remote"
 */

public String getName(String input_str) {
    // TODO Auto-generated method stub
    return input_str;
}
```

Generate EJB related files (1)

- Project properties (right click on the project)
- Select the XDoclet configurations
- Enable XDoclet
- EJB Configuration: right-click in the upper area to pop-up a menu and choose Add.
  - Type EJB
  - Click Ok
Generate EJB related files (2)

- EJBDoclet Configuration
  - Select EJB configuration
  - Right-click in the lower-left area. Choose Add Doclet
  - Choose ejbdoclet and click Ok.
    - Lower-right area
      - Set destDir to src
      - Set ejbSpec to 2.0

- This creates an ejbdoclet that will produce files in src folder under the EJB 2.0 specification.

Generate EJB related files (3)

- Fileset configuration
  - Right-click ejbdoclet and choose Add
  - Choose fileset and click Ok
  - Lower-right area
    - Set dir to src
    - Uncheck excludes
    - Set includes to **/Bean.java

- This will define a fileset that contains the src directory and all files under it that end in Bean.java (i.e., including our MyBean.java)

Generate EJB related files (4)

- Deployment Descriptor
  - Add a new deploymentdescriptor subtask to the ejbdoclet
  - Set the destDir to src/META-INF

- All of the standard EJB deployment descriptors will now be placed in the src/META-INF directory

Generate EJB related files (5)

- Container Configuration (JBoss)
  - Add a new jboss subtask to ejbdoclet
  - Set destDir to src/META-INF
  - Set Version to 3.0

- All of the JBoss-specific deployment descriptors will now be placed in the src/META-INF directory.
Generate EJB related files (6)

- Package Substitution Configuration
  - Add a new `packageSubstitution` subtask to the `ejbdoclet`
  - Set `packages` property to `ejb`
  - Set `substituteWith` property to `interfaces`

- This will place our generated EJB interfaces in the `cugs-ejb.interfaces` java package.

Generate EJB related files (7)

- Interface Configuration
  - Add a new `remotINTERFACE` subtask to the `ejbdoclet`
  - Add a new `homeINTERFACE` subtask to the `ejbdoclet`

- These will generate the EJB home and remote interfaces.

- Click OK
- Right-click on the EJB-CUGS and select Run XDoclet

The Servlet and the Web-App (1)

- Create a new HTTP Servlet
  - File > New > Other > JBoss-IDE > Web Components > HTTP Servlet

- Set `Package` to `cugs-ejb.web`
- Set class `Name` to `MyServlet`
- Under which stubs..... > `init()`
- Under which service method stubs > `doPost()`

The Servlet and the Web-App (2)

- Add a home member
  ```java
  private MyHome home;
  ```

- Complete the init method
  ```java
  public void init(ServletConfig config) throws ServletException {
      try {
          Context context = new InitialContext();
          Object ref = context.lookup("java:/comp/env/ejb/My");
          home = (MyHome) PortableRemoteObject.narrow(ref, MyHome.class);
      } catch (Exception e) {
          throw new ServletException("Lookup of java:/comp/env/ejb/My failed");
      }
  }
  ```
**The Servlet and the Web-App (3)**

- Complete the `doPost()` method
  ```java
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
      // TODO Auto-generated method stub
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html><head><title>Getting the name</title></head>
      <body>
      <h1>Name Service</h1>
      </body>
      </html>");
      My bean = home.create();
      String result = bean.getName();
      out.print("<p>The name is: ");
      out.println(result);
      out.println("</p>");
      } finally {
      out.println("</body></html>");
      out.close();
      }
  }
  ```

- JBoss configuration
  ```
  /**
   * Servlet Class
   * @WebServlet
   *    name="MyServlet"
   *    url-pattern="/MyServlet"
   *    description="Servlet that returns a name"
   *    display-name="My servlet"
   *    @jboss.ejb-ref
   *    name="ejbref"
   *    ref-name="ejbref"
   *    jndi-name="ejbref"
   */
  ```

**Generate Servlet-related files (1)**

- Select Project properties > XDoclet Configuration > Add > Type Web
- Select the Web configuration
- Right-click lower-right area and choose Add Doclet
- Choose `webdoclet` and click Ok.
- Set `destDir` to `src/WEB-INF`
- Our configuration now contains a `webdoclet` that will produce files in the `src/WEB-INF` folder.

**Generate Servlet-related files (2)**

- Fileset Configuration
  - Right-click on the `webdoclet` and choose `Add`.
  - Choose `filesset` and click `Ok`
  - Set properties in the lower-right area
    - Set `dir` to `src`
    - Uncheck `excludes`
    - Set `includes` to `**/*Servlet.java`
  - Our configuration now contains a webdoclet with a fileset that contains the `src` directory, and all files under it that end in `Servlet.java`
Generate Servlet-related files (3)

- Deployment Descriptor
  - Add a new deploymentdescriptor subtask to the webdoclet
  - Set Servletspec to 2.3.

- All of the standard Web deployment descriptors will now be placed in the src/WEB-INF

Generate Servlet-related files (4)

- JBoss Configuration
  - Add a new jbosswebxml subtask to the web-doclet
  - Set version to 3.0

- All of the JBoss-specific Web deployment descriptors will now be placed in the src/WEB-INF directory

- Click on the XDoclet and save

- Right-click on the Project and select Run XDoclet

Generate Servlet-related files (5)

- Create the HTML page
  - Create a docroot folder under the root of the project
  - Create an index.html under the docroot folder.

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Name Request</title>
</head>
<body>
<h1>Name Request Form</h1>
<form action="MyServlet" method="POST">
<table cellspacing="2" cellpadding="2" border="0">
<tr><td><input type="submit" name="Compute" value="Compute"></td>
<td><input type="Reset"></td></tr>
</table>
</form>
</body>
</html>
```

We have done

- Created a project
- Created an EJB
- Generated the EJB-related files
- Created a servlet and a web application
- Generated the servlet-related files
Remains to be done

- Create a J2EE Application
- Package the application (jar, war)
- Configure JBoss servlet container and launch
- Deploy the application

J2EE Application (1)

- Create the `application.xml`
  - Right-click on the `src/META-INF` and choose `New > Other...`
  - Choose `JBoss-IDE > Descriptors > EAR 1.3 Deployment Descriptor` and click `Next`
  - Make sure `application.xml` is the name of the file and click `Finish`

J2EE Application (2)

- Add the following to the `application.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN" "http://java.sun.com/dtd/application_1_3.dtd">
<application>
    <display-name>Get Name Application</display-name>
    <modules>
        <ejb>MyEJB.jar</ejb>
    </modules>
    <web>
        <web-uri>MyWeb.war</web-uri>
        <context-root>/My</context-root>
    </web>
</application>
```

Packaging (1)

- The EJB JAR. It will contain the EJB classes and interfaces, as well as the `ejb-jar.xml` and `jboss.xml` deployment descriptors
- The EJB Client Jar. It will contain the EJB interfaces.
- The Web Application WAR. It will contain the Servlet class, the EJB client Jar, as well as the `web.xml` deployment descriptors
- The J2EE Application EAR. It will contain the EJB Jar and the Web Application War, as well as the `application.xml` deployment descriptor.
Packaging (2)

- Enable Packaging
  - Right-click project properties
  - Select Packaging Configurations
  - Enable Packaging

Packaging (3): MyEJB.jar

- Create EJB JAR
  - Right-click in the area to pop-up the menu and choose Add Archive. Type MyEJB.jar

- We want to add the EJB classes and interfaces.
  - Select the MyEJB.jar and right-click in the area to pop-up the menu and select Add Folder.
  - A folder chooser dialog appears
    - Click Project Folder
    - Select /CUGS-EJB/bin. Click Ok.
  - Set includes to: cugs-ejb.ejb/*.class,cugs-ejb/interfaces/*.class and click Ok.

Packaging (4): MyEJB.jar

- Add the standard EJB deployment descriptor
  - Select the MyEJB.jar
  - Right-click in the area and select Add File from the pop-up menu

  - The dialog allows to select which file to include in the package and to set a prefix which will be appended when building the package
    - Click Project File
    - Select CUGS-EJB/src/META-INF/jboss.xml and click Ok.
    - Set the Prefix to META-INF. The ejb-jar.xml should be located in the META-INF directory.

Packaging (5): MyEJB.jar

- Add the specific EJB deployment descriptor (JBoss)
  - Select the MyEJB.jar
  - Right-click in the area and select Add File from the pop-up menu

    - Click Project File
    - Choose CUGS-EJB/src/META-INF/jboss.xml
    - Set Prefix to META-INF. The jboss.xml should be located in the META-INF directory.

- The package configuration for MyEJB.jar is complete now
Packaging (6): MyEJB-client.jar

- MyEJB-client.jar
  - No need if you run JBoss 4.0 or above

Packaging (7): MyWeb.jar

- Create a WEB WAR
  - Click the Add button in the Packaging Configuration and type MyWeb.war
- Add the web classes
  - Select the MyWeb.war, right-click in the area and select Add Folder from the pop-up menu.
  - Click Project Folder in the folder chooser dialog and select CUGS-EJB/bin. Click Ok.
  - Set Includes to cugs-ejb/web/*.class as we only want to include web-related class files.
  - Web classes should be located in the WEB-INF/classes folder according to the container specification. Set the Prefix to WEB-INF/classes

Packaging (8): MyWeb.jar

- Add the standard web deployment descriptor
  - Select the MyWeb.war, right-click in the area and select Add File from the pop-up menu.
  - Click Project File in the folder chooser dialog and select CUGS-EJB/src/WEB-INF/web.xml. Click Ok.
  - The web.xml should be located in the WEB-INF folder according to the container specification. Set the Prefix to WEB-INF

Packaging (9): MyWeb.jar

- Add the specific web deployment descriptor (JBoss)
  - Select the MyWeb.war, right-click in the area and select Add File from the pop-up menu.
  - Click Project File in the folder chooser dialog and select CUGS-EJB/src/WEB-INF/jboss-web.xml. Click Ok.
  - The web.xml should be located in the WEB-INF folder according to the container specification. Set the Prefix to WEB-INF
Packaging (10): MyWeb.jar

- Add HTML
  - Select the MyWeb.war, right-click in the area and select Add Folder from the pop-up menu.
  - Click Project Folder in the folder chooser dialog and select CUGS-EJB/docroot. Click Ok.

- The package configuration for MyWeb.jar is now complete.

Packaging (11): MyApp.ear

- Create an APP EAR
  - Click the Add button in the Packaging Configuration and type MyApp.ear. You have created a packaging configuration that will produce MyApp.ear.

- Add the application deployment descriptor
  - Select the MyApp.ear, right-click in the area and select Add File from the pop-up menu.
  - Click Project File in the folder chooser dialog and select CUGS-EJB/src/WEB-INF/application.xml. Click Ok.
  - The application.xml should be located in the WEB-INF folder according to the container specification. Set the Prefix to WEB-INF

- Add the EJB module to the application
  - Select the MyApp.ear, right-click in the area and select Add File from the pop-up menu.
  - Click Project File in the folder chooser dialog
    - The file to be selected is CUGS-EJB/MyEJB.jar but it does not exist yet, so set File to CUGS-EJB/MyEJB.jar manually (type it in)
  - Click Ok.

- The packaging configuration for MyApp.ear is now complete.

Packaging (12): MyApp.ear

- Add the WEB module to the application
  - Select the MyApp.ear, right-click in the area and select Add File from the pop-up menu.
  - Click Project File in the folder chooser dialog
    - The file to be selected is CUGS-EJB/MyWeb.jar but it does not exist yet, so set File to CUGS-EJB/MyWeb.jar manually (type it in)
  - Click Ok.

- The packaging configuration for MyApp.ear is now complete.

Packaging (13): MyApp.ear

- Add the WEB module to the application
  - Select the MyApp.ear, right-click in the area and select Add File from the pop-up menu.
  - Click Project File in the folder chooser dialog
    - The file to be selected is CUGS-EJB/MyWeb.jar but it does not exist yet, so set File to CUGS-EJB/MyWeb.jar manually (type it in)
  - Click Ok.

- The packaging configuration for MyApp.ear is now complete.

- Right-click on the project and > Run Packaging
Remains

- JBoss Configuration and launch
- Application deployment

- See JBoss documentation www.jboss.org