



UML Export

Example for Configuring a Post Process

1	Introduction.....	3
2	Used Software.....	3
3	Used Folder Structure.....	3
4	The Example.....	4
5	Configuration in Composum.....	5
6	Process Execution.....	7
7	Appendix - Used Files.....	8
7.1	build.bat.....	8
7.2	application.xml.....	9
7.3	build.properties.....	9
7.4	build.xml.....	10
7.5	uml2ejb.xml.....	10

1 Introduction

This project is intended to demonstrate the inclusion of a post-process into **Composum**. By an example suitable for practice a scenario is to develop for inclusion and application of a process which immediately after exporting a model from Composum to XML is processing this export result. As process to include **UML2EJB** will be used (<http://uml2ejb.sourceforge.org>). This tool generates from UML object models which are present in XML format complete Enterprise Java Beans. That means: the according Java code is generated, compiled, Java archives are build and the EAR archive for the EJBs. The result may be deployed, for instance, in JBoss application server. The advantage of this tool is that it not only for a single class but also for complete UML models is able to generate EJBs including the relations among each other.

2 Used Software

In the following there are listed the software packages including their version numbers, which are applied in the present example. The co-operation of this components for the UML export in Composum is tested.

- uml2ejb_131
- jakarta-ant-1.5.1
- jaxb-1.0-beta
- jboss-3.0.0
- jdk1.4.1_01

3 Used Folder Structure



Folder Structure

xmi - contains the Composum example

export - contains the executing command, make-files, properties

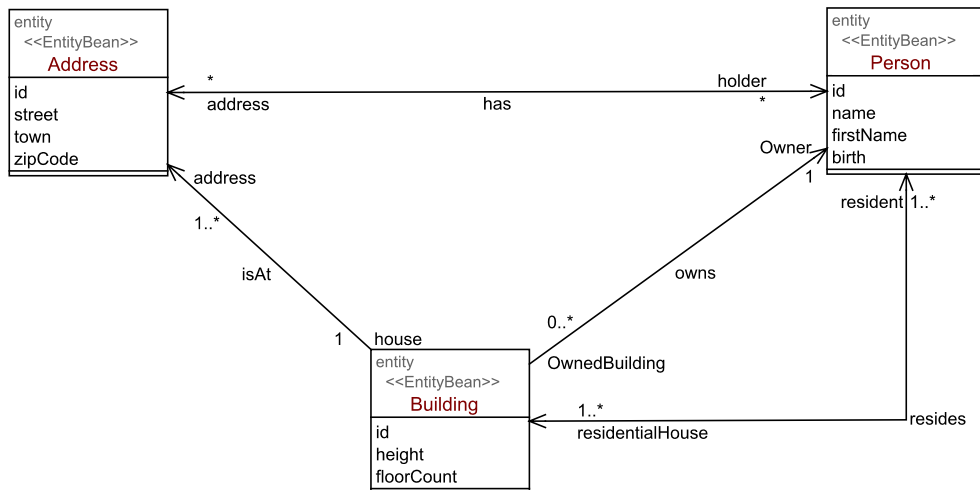
model/xmi - here the generated XMI file is put

model/xml - here a intermediate representation of the model in XML format is put

src/app/config - contains a file `application.xml`

Further folders are to configure for the process and are created automatically during process execution if they are not existing yet. They contain the results of the generation process.

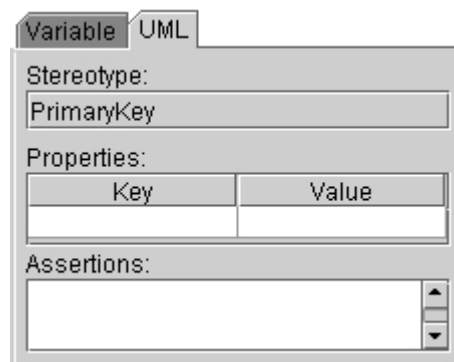
4 The Example



Example for the UML Export

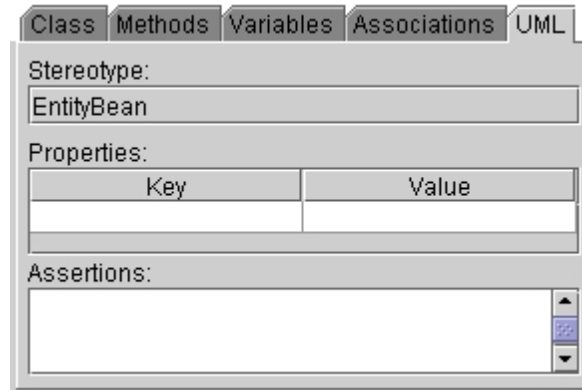
The example represents the relations between persons, addresses and buildings in a class diagram. To design the model UML2EJB conform you must comply the following design rules:

- All associations are to be named completely (names, roles). This can be done in the tab Association of the property editor of the particular association.
- Any Entity Bean must have an attribute with the stereotype *PrimaryKey*. This can be defined in the tab *UML* of the property editor of variables of a class; it is reached by clicking the arrow button at the end of the line in the variables table of that class.



Stereotype of Attributes

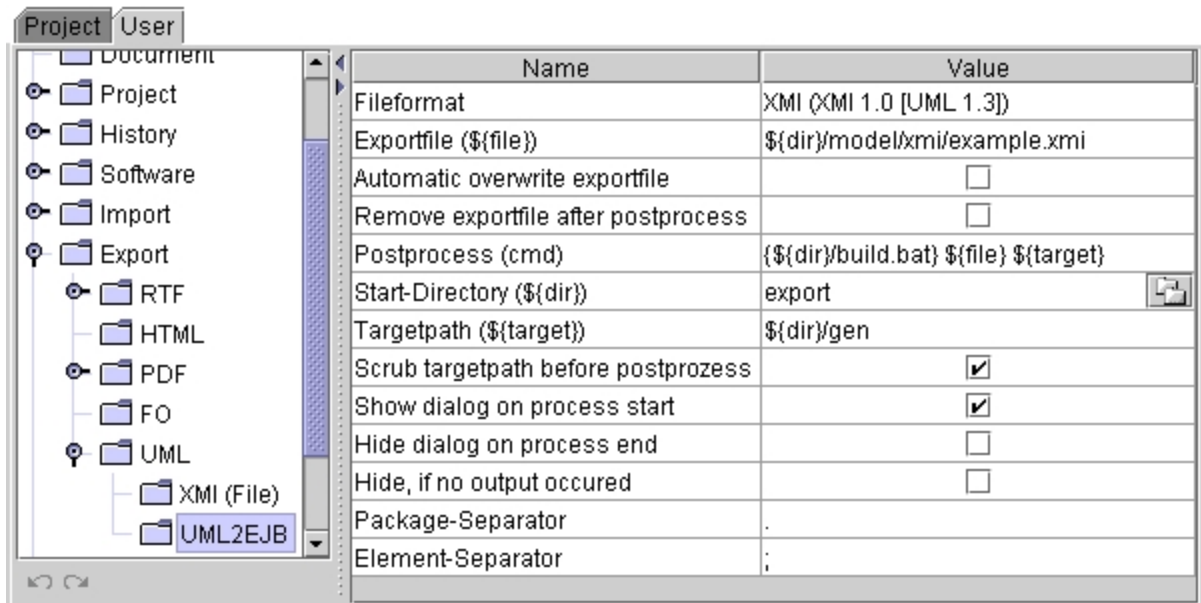
- Any EntityBean must have the stereotype *EntityBean*, to define in the tab *UML* of the property editor of the class.



Stereotype EntityBean

5 Configuration in Composum

The export process and the immediately following processing can be configured in **Composum** as a new menu item for the file menu and the context menus of class diagrams, packages or single classes. This menu item is to start the the XMI export from Composum and then to execute the generating of the EJBs using UML2EJB. To do this the files described above can be applied. This must be configured. In the user configuration of Composum, below *Export - UML* a new entry is to create using the context menu. In the example it is named *UML2EJB*. After selecting it on the right side of the configuration dialog a table to configure the process is shown.



Configuration of the Export Process

File format

The file format is XML. According to the UML specification for the versions 1.3 and 1.3 (www.omg.org) is XML intended as an export format for the UML metamodel, where here only the two combinations *XMI 1.0/UML 1.3* or *XMI 1.1/UML 1.4* are allowed. This field is already pre-defined. By default, the combination *XMI 1.0/UML 1.3* is set. For changes the specified syntax must be complied.

Start Directory

This is the directory from which the process is started. In the example this is the folder which

contains the file `build.bat`. The value can be absolutely or relatively to the directory in which the project is located. You should be carefully when specifying it relatively. It is a user specific configuration setting. It is not at other users' disposal which are working on this project. On the other side it is at all projects' disposal which are in work by this user, independent on the location of the particular project. The button at the end of the row opens the file chooser dialog to easily select a directory.

The value defined here can be used as a variable `${dir}` in the further configuration and also in the used files.

Exportfile

Here path and name of the result file of the XMI export is to specify. In our example the file must have the extension `.xmi`. For a path specification the already defined path for the execution (start) directory as variable `${dir}` can be used, if it is part of the path. The file itself can be used as variable `${file}` in the further configuration.

It is recommended to put the file into the directory `model/xmi` of the folder structure recommended in the chapter 3 Used Folder Structure.

Automatic overwrite exportfile

In the example this option is not set. A warning dialog is shown and the action can be cancelled if needed.

Remove exportfile after postprocess

This option is also not set in the example.

Postprocess

Here the executing process is defined. In the example this is the file `build.bat` in the folder `${dir}`. In the most simple case the specification `${dir}/build.bat` sufficient. In the example this process are overgiven two variables. These are the XMI file `${file}` which is to create and the target directory `${target}`. The parentheses enclosing the call `${dir}/build.bat` are to guarantee the operation system specific interpreting of path separators.

Targetpath

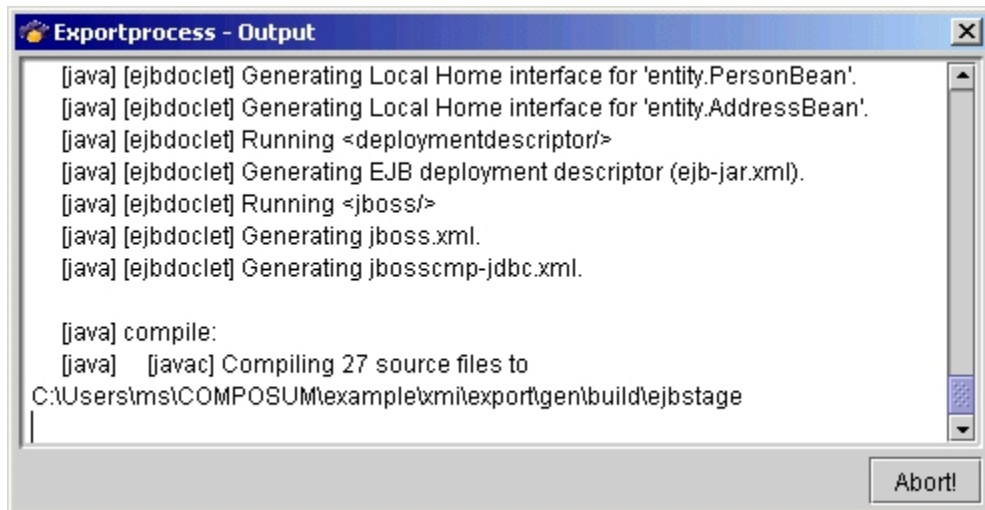
This value is not always important. If for the code generation required, it can be specified as a parameter in the command line (see postprocess). This value can be used as a variable `${target}`. The specified directory is created during process execution automatically if it is not existing yet.

Scrub target path before postprocess

This option is set in the example. It is intended to clear garbage, i. e. old sources that are not needed. So it can be ensured that only the object model from Comosum is included into the code generating.

Dialog Options

Informations about the process run that normally are shown by the executing process in a console can be followed from Comosum in a special dialog. At least at the beginning it is recommended to let show the dialog because it gives informations about success or non-success of the export. Additionally, you can cancel the process during its run.



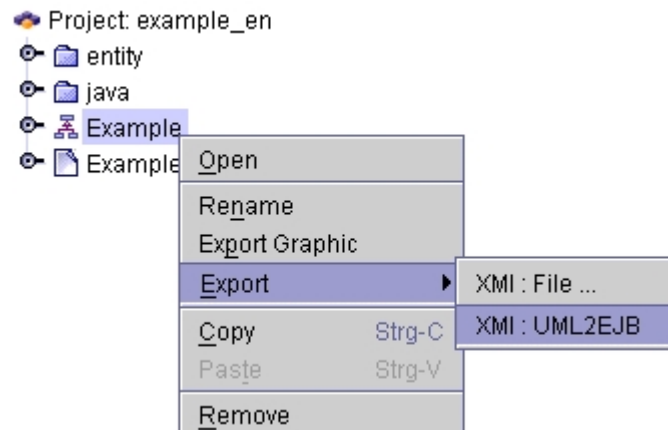
Shown Dialog during Process Run

Package and Element Separators

This settings are not needed in the example. If this is important for other processes you will find informations in the particular documentations.

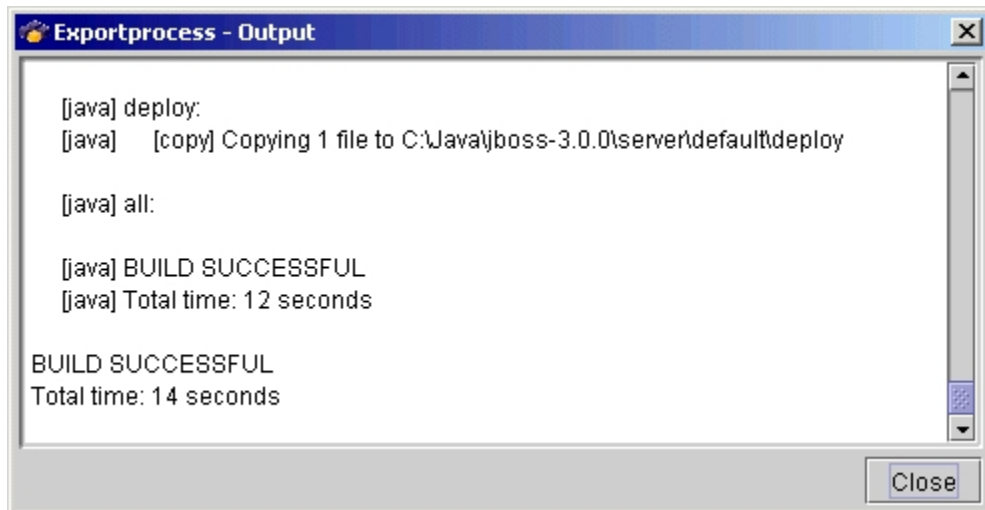
6 Process Execution

After finishing the configuration of the post-process it can be started from the file menu of Composum or using the context menu of a class diagram, a package or a single class.



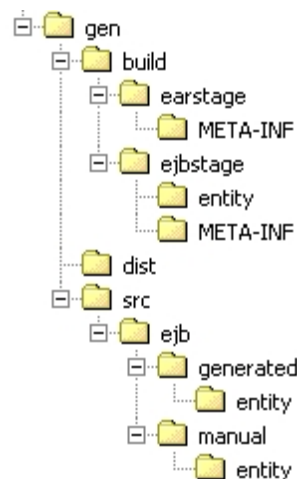
Prozess Start for a Class Diagram

The export is always executed over the structure of the origin of starting it. I. e. when calling it in the file menu the whole project is exported, activation from a package exports its whole content including its sub-packages. A class can also be single exported. The export of a class diagram exports all classes contained in it and all associations between them. That's why it is recommended to export the modelling of a class diagram.



Export Process Finished

After successfully finishing the process you can find the following folder structure according to that specified in the file `uml2ejb.xml`:



Verzeichnisse nach dem Export

The Java sources are put into the folder `src`. In the folder `manual/ejb` there are that sources which can be worked on by the developer manually (the implementation). This, for instance, was defined in the file `uml2ejb.xml` as `uml2ejb.ejb.impl.dir`. In the folder `build` there are contained the compiled classes (`ejbstage`) and the JAR archives. The folder `dist` contains the EAR archive.

In the folder `model/xmi` the file `example.xmi` is found which is specified in the configuration (`${file}`).

7 Appendix - Used Files

7.1 build.bat

```
@echo off
```

```
REM JDK 1.4.x contains a TRAX-Compatible XSLT-Processor (i.e. Apache XALAN)
```

```
REM If JDK 1.3.x is used, a TRAX-Compatible XSLT-Processor needs to
```

```
be installed and
REM put on the CLASSPATH

SET JAVA_HOME=C:\Java\jdk1.4.1_01
SET ANT_HOME=C:\Java\jakarta-ant-1.5.1

SET ANT_OPTS="-Dxmi.file=%1" "-Dgen.dir=%2"

call %ANT_HOME%\bin\ant.bat
```

In this file first needed paths are to be declared, as far as they are not already defined as environment variables in the system. In particular, these are the `JAVA_HOME` and the `ANT_HOME` folder. If not the JDK 1.4 is used it is additionally important to include a TRAX compatible XSLT processor (for instance *Xalan of Apache*). It must be taken into the classpath.

7.2 application.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE
Application 1.2//EN"
'http://java.sun.com/j2ee/dtds/application_1_2.dtd'>

<application>
  <display-name>Test Application</display-name>
  <module>
    <ejb>test-application-ejb.jar</ejb>
  </module>
</application>
```

This is a configuration file. Especially the tag `<module>` is important. It describes the table of contents for the EAR archive. It contains the name of the jar file(s). This must already exist and corresponds to the EJB Specification.

7.3 build.properties

```
uml2ejb.home=C:/Java/uml2ejb_131/
myant.home=C:/Java/jakarta-ant-1.5.1/
jaxb.home=C:/Java/jaxb-1.0-beta/
jboss.home=C:/Java/jboss-3.0.0/
j2ee.home=C:/Java/j2sdkee1.2/

uml2ejb.libs=${uml2ejb.home}uml2ejb.jar:${uml2ejb.home}jaxbfix
junit.libs=${uml2ejb.home}lib/junit.jar
struts.libs=${uml2ejb.home}lib/struts.jar
velocity.libs=${uml2ejb.home}lib/velocity-dep-1.2.jar
xdoclet.libs=${uml2ejb.home}lib/xdoclet-apache-module.jar:${uml2ejb.
home}lib/xdoclet-bea-module.jar:${uml2ejb.home}lib/xdoclet-caucho-mo
dule.jar:${uml2ejb.home}lib/xdoclet-ejb-module.jar:${uml2ejb.home}li
b/xdoclet-exolab-module.jar:${uml2ejb.home}lib/xdoclet-hp-module.jar
:${uml2ejb.home}lib/xdoclet-ibm-module.jar:${uml2ejb.home}lib/xdocle
t-java-module.jar:${uml2ejb.home}lib/xdoclet-jboss-module.jar:${uml2
ejb.home}lib/xdoclet-jdo-module.jar:${uml2ejb.home}lib/xdoclet-jmx-m
odule.jar:${uml2ejb.home}lib/xdoclet-macromedia-module.jar:${uml2ejb
.home}lib/xdoclet-mvcsoft-module.jar:${uml2ejb.home}lib/xdoclet-mx4j
-module.jar:${uml2ejb.home}lib/xdoclet-objectweb-module.jar:${uml2ej
b.home}lib/xdoclet-orion-module.jar:${uml2ejb.home}lib/xdoclet-prama
ti-module.jar:${uml2ejb.home}lib/xdoclet-sybase-module.jar:${uml2ejb
.home}lib/xdoclet-web-module.jar:${uml2ejb.home}lib/xdoclet-webwork-
module.jar:${uml2ejb.home}lib/xdoclet-xdoclet-module.jar:${uml2ejb.h
ome}lib/xdoclet.jar:${uml2ejb.home}lib/xjavadoc.jar
logger.libs=${uml2ejb.home}lib/commons-logging-api.jar:${uml2ejb.hom
e}lib/commons-logging.jar:${uml2ejb.home}lib/log4j.jar
```

```
ant.libs=${myant.home}lib/ant.jar:${myant.home}ext/optional.jar

jaxb.libs=${jaxb.home}lib/jaxb-api.jar:${jaxb.home}lib/jaxb-libs.jar
:${jaxb.home}lib/jaxb-ri.jar:${jaxb.home}lib/jaxb-xjc.jar:${jaxb.home}lib/jax-xml.jar

j2ee.libs=${jboss.home}server/default/lib/jboss-j2ee.jar:${j2ee.home}lib/j2ee.jar

build.classpath=${uml2ejb.libs}:${junit.libs}:${struts.libs}:${velocity.libs}:${xdoclet.libs}:${logger.libs}:${ant.libs}:${j2ee.libs}:${jaxb.libs}
```

This file corresponds in its contents to the file `build.bat`. It defines classpath and home folders. To fit it to the example here the JBoss folder is added.

7.4 build.xml

```
<!-- Calls ant with uml2ejb.xml as buildfile with JAXB on the systemclasspath.
      This is necessary due to a bug in the current JAXB-Release (beta) -->
<project name="launch" default="launch">

  <property file="build.properties" />

  <path id="launch.class.path">
    <pathelement path="${build.classpath}" />
    <pathelement path="${java.class.path}" />
  </path>

  <target name="launch">
    <java classname="org.apache.tools.ant.Main" fork="true">
      <classpath refid="launch.class.path" />
      <jvmarg value="-Dant.home=${ant.home}" />
      <jvmarg value="-Dxmi.file=${xmi.file}" />
      <jvmarg value="-Dgen.dir=${gen.dir}" />
      <arg value="-f"/>
      <arg value="uml2ejb.xml" />
    </java>
  </target>
</project>
```

Prepares the Ant build process and includes the file `uml2ejb.xml` as the make file.

7.5 uml2ejb.xml

```
<!-- This Ant-buildfile describes a sample scenario where an XMI-File
      is transformed into a J2EE-Application and deployed into the
      JBOSS (3.0.0)
      applicationserver using the UML2EJB-Codegenerator (1.31) -->
<project name="test" default="all" basedir=".">

  <property file="build.properties" />

  <property name="src.app.dir" value="src/app" />
  <property name="src.common.dir" value="${gen.dir}/src/common" />
  <property name="src.ejb.dir" value="${gen.dir}/src/ejb" />
```

```
<property name="src.appconfig.dir" value="\${src.app.dir}/config"
/>

<property name="uml2ejb.ejb.bean.dir"
value="\${src.ejb.dir}/generated" />
<property name="uml2ejb.ejb.impl.dir"
value="\${src.ejb.dir}/manual">
<property name="ejbdoclet.output.dir"
value="\${src.ejb.dir}/generated" />

<path id="build.class.path">
<pathelement path="\${build.classpath}"/>
</path>

<target name="all" depends="deploy"/>

<!-- create necessary directory structure -->
<target name="init">
<mkdir dir="\${gen.dir}/dist"/>

<mkdir dir="\${gen.dir}/build"/>
<mkdir dir="\${gen.dir}/build/earstage"/>
<mkdir dir="\${gen.dir}/build/ejbstage"/>

<mkdir dir="\${src.ejb.dir}/generated"/>
</target>

<!-- convert the XMI-File to Java-Classes (including XDoclet Com-
ments)-->
<target name="genjava">
<taskdef name="uml2ejb"
class-
name="de.mbohlen.tools.uml2ejb.anttasks.Uml2EjbGenTask"
classpathref="build.class.path"/>

<!-- transform the XMI-File into a simpler XML-Notation -->
<style in="\${xmi.file}"
out="model/xml/model.xml"
extension=".xml"
includes="*.xmi"
processor="trax"
style="\${uml2ejb.home}/xmi-to-simple00.xsl"/>

<!-- transform the simple XML-Notation to Java-Classes -->
<uml2ejb basedir="model/xml"
ejbDestdir="\${uml2ejb.ejb.bean.dir}"
implDestdir="\${uml2ejb.ejb.impl.dir}"
includes="*.xml"
lastModifiedCheck="true"
templatePath="\${uml2ejb.home}"
useDefaultTemplateConfig="true"
typeMappings="\${uml2ejb.home}/TypeMapping.xml">
</uml2ejb>
</target>

<!-- generate the EJB-Artefacts from XDoclet-Comments -->
<target name="genejb" depends="genjava">
<taskdef name="ejbdoclet"
classname="xdoclet.modules.ejb.EjbDocletTask"
classpathref="build.class.path"/>

<ejbdoclet destdir="\${ejbdoclet.output.dir}"
ejbspec="2.0"
force="false">
<fileset dir="\${uml2ejb.ejb.bean.dir}">
```

```
    <include name="**/*Bean.java" />
  </fileset>

  <dataobject/>
  <utilobject/>
  <remoteinterface/>
  <homeinterface/>
  <localinterface/>
  <localhomeinterface/>
  <deploymentdescriptor dest-
dir="${gen.dir}/build/ejbstage/META-INF/" />

    <jboss version="3.0"
      xmlencoding="UTF-8"
      typemapping="Hypersonic SQL"
      datasource="java:/DefaultDS"
      destdir="${gen.dir}/build/ejbstage/META-INF/" />
  </ejbdoclet>

  <replace file="${gen.dir}/build/ejbstage/META-INF/ejb-jar.xml" >
    <replacetoken><![CDATA[ServiceBean</ejb-class>]]></replacetoken>
  /
  <replacevalue><![CDATA[ServiceBeanImpl</ejb-class>]]></replacevalue>
  </replace>
  <replace file="${gen.dir}/build/ejbstage/META-INF/ejb-jar.xml" >
    <replacetoken><![CDATA[Bean</ejb-class>]]></replacetoken>
    <replacevalue><![CDATA[BeanCMP</ejb-class>]]></replacevalue>
  </replace>
</target>

<!-- compile all files -->
<target name="compile" depends="init,genejb">

  <!-- compile EJB classes -->
  <javac destdir="${gen.dir}/build/ejbstage" includes="**/*.java"
    includeAntRuntime="false">
    <src path="${uml2ejb.ejb.bean.dir}" />
    <src path="${uml2ejb.ejb.impl.dir}" />
    <src path="${ejbdoclet.output.dir}" />
    <classpath refid="build.class.path" />
  </javac>
</target>

<!-- create the EJB JAR and the EAR files-->
<target name="jars" depends="init,compile">
  <jar
jarfile="${gen.dir}/build/earstage/test-application-ejb.jar">
    <fileset dir="${gen.dir}/build/ejbstage/" includes="**/*" />
  </jar>
  <copy todir="${gen.dir}/build/earstage/META-INF/">
    <fileset dir="${src.appconfig.dir}" in-
cludes="application.xml" />
  </copy>
  <jar jarfile="${gen.dir}/dist/test-application.ear">
    <fileset dir="${gen.dir}/build/earstage/" includes="**/*" />
  </jar>
</target>

<!-- deploy the EAR in JBoss -->
<target name="deploy" depends="jars"
  description="Deploys generated ear in JBoss">
  <copy todir="${jboss.home}/server/default/deploy" >
    <fileset dir="${gen.dir}/dist/" includes="*.ear" />
  </copy>
</target>
```

</project>

This is the make file for execution of the Ant build process. This file can be fitted according to the intended tasks. In the example it generates from the XMI file from Composum a XML representation which serves as a base for generating Java classes according to the EJB specification. Then the result is compiled the JAR archives and the EAR archive are built. If needed, also the generation of a javadoc documentation or the including of own code is possible. It is recommended to read the according documentations. Example for execution of Ant tasks are there already delivered and are to fit to your requirements. In the file you can see the definition and the creation of the different folders for the export results (`property`, `mkdir`).