Automated EJB2 to EJB3 Migration

Isha Mittal,
AMITY University
Noida, India

J.S. Sodhi, PhD
AMITY University
Noida, India

Abstract

Conversion from some old technology to new technology is requirement for today's world, our IT industry was started growing up in late 90's. At that time there were lots of development comes in different industry, banking system had changed drastically, it's come online with online solutions, ATM takes place for withdrawing money from the bank, similarly manufacturing industry grows with IT etc.

We had used advance technology of that time for providing solution to the respective industry. As the time changes there are some advancement in each field so IT has also comes with lots of new technology & configurations, so that we can solve the real world problem more accurately with cheap prize.

We are providing a automate migration tool that will migrate EJB2.1 (old) to EJB3.0 (new)[1], EJB2.1 have several disadvantages like it's do not have annotation where annotations are server independent configurations.

Keywords

EJB2, EJB3, migration, business layer, Ejb2 to Ejb3 migration, time saving,

1 INTRODUCTION

Develop a tool to automate the generation of EJJB3.0 entity (with JPA) and corresponding DAO objects using available EJB2.1 artifacts such as XDoclet bean and the code (EJB2.1 files, Weblogic AS specific deployment descriptors, Data Transfer Objects (DTOs)) generated using it.[2]

The artifacts generated by the tool should follow the listed standards.

- EJB3.0 Entity

o Should list the files that were used to generate the entity in class level comments.

o Should contain standard JPA annotations like "@Entity", "@Table", "@NamedQueries", "@TransactionAttribute", "@Column", "@Id", "@SequenceGenerator", "@GeneratedValue", and "@Lob" wherever applicable.

o Should list the EJB2.1 deployment descriptor annotations that are not processed by the tool as "*TODO*" in class level comments

o Should extend CLBaseEntity<PK Type> and implement "*getCompareables()*" and "*getPrimaryKey()*" methods.

o In case of composite primary key, a separate primary key class should be generated annotated with "@Embeddable" annotation.

o In case of composite primary key, entity should have an instance variable of the primary key class with "*@EmbeddedId*" annotation.

- DAO Interface

o Should extend CLBaseDao<EJB3.0-Entity, PK Type>

o Should have declarations of all finder methods.

o Should have "*TODO*" comment that list all the custom methods (not including getters/setters of entity fields and "*getData()*" methods) with parameters.

- DAO Implementation Class

o Should extend CLBaseDaoImpl< EJB3.0-Entity, PK Type> and implement <EJB3.0-Entity>Dao

o Should have implementations of all finder methods.

o Should have "*TODO*" comment that list all the custom methods (not include getters/setters of entity fields and "*getData()*" methods) with parameters

2.1 Analysis

2.1.1 Identification of Data

In this analysis all the data that would represent an entity attributes was recognized. The following is the list of such attributes.

- Entity name and fields
- Table name and columns
- Finder methods and queries
- Entity relationships

2.1.2 Identification of Artefacts

In this analysis all the artifacts that would be used to aggregate entity attributes was recognized. The following is the list of such artifacts.

- ejb-jar.xml
- weblogic-cmp-rdbms-jar.xml
- weblogic-ejb-jar.xml
- Local interface corresponding to each entity bean
- Data Transfer Objects corresponding to each entity bean

- Patterns recognized

o In-case of deployment descriptors each module contains these xml files in <path>\target\classes\META-INF\

o The corresponding jar file contain the DTOs and local interfaces of the entity beans are accessible from <path>\target\

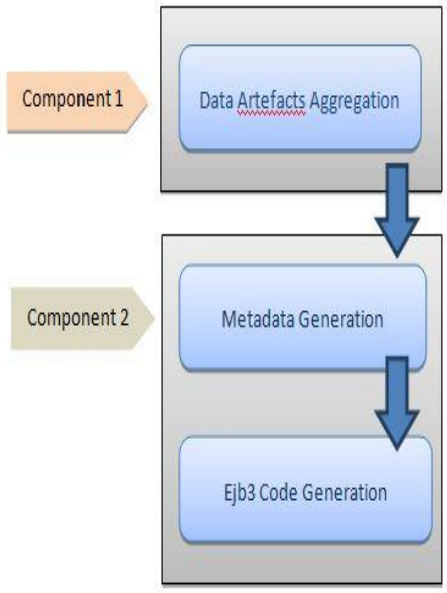
2.1.3 Identification of Development Methodologies

In this analysis development methodologies were identified that would be used to generate EJB3.0 and DAO code. The following is the list of such artifacts.

- XML Transformation using JAXP
- Saxon Processor (XSLT 2.0 compatible)
- Logging using log4j
- Directory scanning based on pattern

2.2 Design

The tool consists of 2 mains components as depicted in Figure1



2.2.1 Data Artefacts Aggregation

This component searches a particular directory for certain set of files by making use of Regular Expression and generates 2 files. Figure 2 depicts this process.

The file patterns used to scan the required directory are

- Pattern used to filter 3 types of files – "(^weblogic-cmp-rdbms-jar.xml)|(^ejb-jar.xml)|(^ weblogic-ejb-jar.xm)"
- Pattern used to make sure that files come from certain type of directory only – ".*classes\\\\META-INF.*"
- Pattern used to filter jar file that contains DTOs and local interfaces – "\\D.*-SNAPSHOT.jar$"

The file crawler generates two files after scanning the depths of the required directory.

- An xml file (*filteredDDPaths.xml*) that contains the absolute paths of the deployment descriptor files for each module.

```
<project>

<module>

<ejb-dd-path>../work/../classes/META-INF/ejb-jar.xml</ejb-dd-path>

<web-rdbms-dd-path>../work/../classes/META-INF/weblogic-cmp-rdbms-jar.xml</web-rdbms-dd-path>

<web-ejb-dd-path>../work/../classes/META-INF/weblogic-ejb-jar.xml</web-ejb-dd-path>

</module>
```

- A file (*jarPaths.txt*) that lists the absolute paths of the jar files that contain DTOs and local interfaces for each module. This list would be used to set Java "classpath" and is a pre-requisite for "Data Transformation"

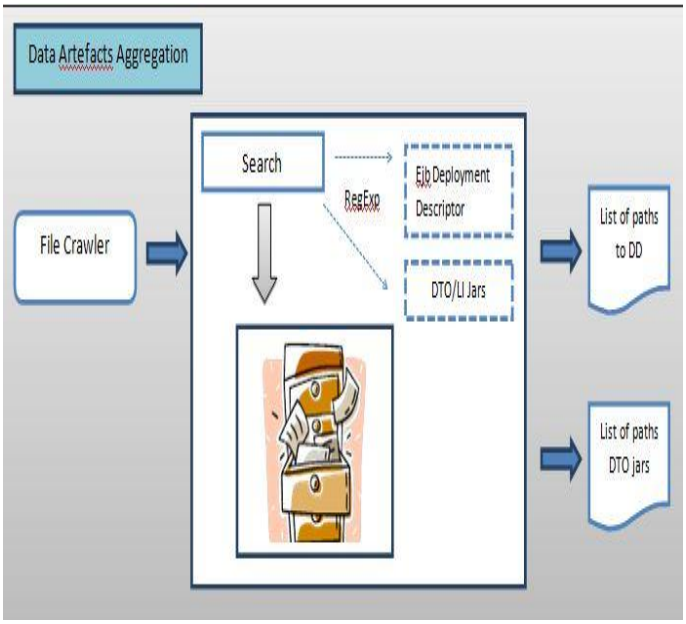


Figure 2–Data Artifacts Aggregation

2.2.2 Data Transform

The data transformation component provides the functionality of meta-data generation and finally EJB3.0/DAO code generation. The following Figure 3 depicts this process.

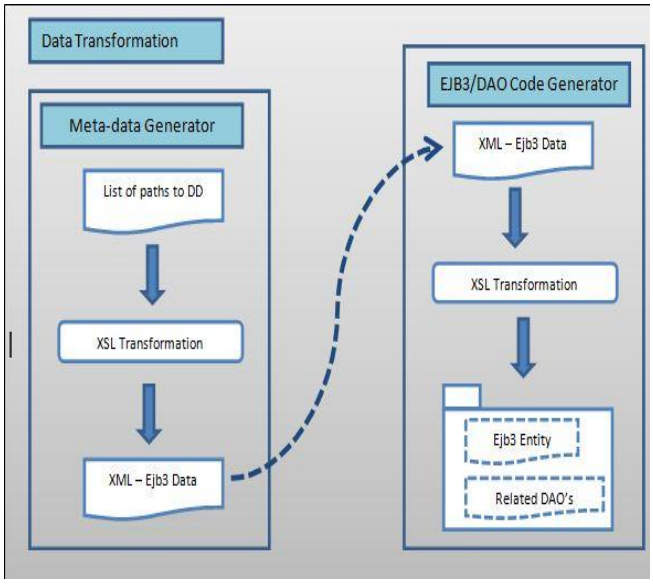


Figure 3–Data Transformation

2.3.1 Meta-data Generation

This component gathers all the information segregated in different files for each entity and place it one xml file (*entity-data.xml*). XSL transformation would be incorporated to work with different xml files and access DTOs present in jar files to aggregate the data. For each "module" element in "*filteredDDPaths.xml"*, a separate "*entity-data.xml*" file would be generated.

2.3.2 EJB3.0/DAO Code Generation

This component uses the meta-data file "*entity-data.xml*" and performs an XSL transformation to generate the Java code. The XSL transformation will generate EJB3.0 entity and it corresponding DAO objects referencing the generic DAO.

3 IMPLEMENTATION

3.1 Development Modules

Eclipse IDE is used for tool development.

3.1.1 3rdParty

- Apache

○ **log4j-1.2.16.jar** – Application logging

- Saxon-9.1B

○ **saxon9-dom.jar** – XML document parsing

○ **saxon9.jar** – XSLT processor

- J2EE

○ **j2ee-1.4.jar** – Required because the tool loads EJB2.1 files (local interfaces) using reflection for scanning

3.1.2 Common

3.1.2.1 Utilities

- **ConfigReader.java** – This class is used to load base configuration file.
- **ToolResourceBundle.java** – This class maintains resource bundle for given file.
- **Constants.java** – This interface holds application constants values.
- **ClassAnalyzer.java** – This component loads a class dynamically and retrieves its properties.
- **DataInterpreter.java** – This class is to analyse the data for blank or empty values.
- **FileUtility.java** – This class provides various file I/O functionality.

3.1.2.2 Resources

- **ejb2-to-ejb3-config.properties** – Tool configuration file
- **log4j.properties** – Tool log4j configuration file
- **CatalogManager.properties** – This file tells the resolver where to look for catalog files and sets configuration options.
- **Xml-catalog.xml** – A catalog in XML provides a mapping from generic addresses to specific local directories on a given machine. A catalog can be used to locate the DTD, system entity files, and stylesheet files during processing.
- **ejb-jar_2_0.dtd** – DTD used by "ejb-jar.xml"
- **gcode-constants.xml** – This XML file holds constants that used by XSLT files.

3.1.3 EJB2 to EJB3

3.1.3.1 Data Artefacts Aggregation

- **FileCrawler.java** – This component searches a particular directory for certain set of files by making use of Regular Expression and generates 2 files. The first is an XML file that contains the absolute paths of the deployment descriptor files for each module. The second is a text file that lists the absolute paths of the jar files that contain DTOs and local interfaces for each module.

3.1.3.2 Data Transformation

- Meta-data Generator

○ **MetadataGenerator.java** – This class gathers all the information segregated in different files for each entity and place it one xml file (*entity-data-i.xml*). XSL transformation would be incorporated to work with different xml files and access DTOs present in jar files to aggregate the data. For each "*module*" element in "*filteredDDPaths.xml*", a separate "*entity-data-i.xml*" file would be generated.

○ **meta-data-builder.xsl** – XSLT file used for meta-data generation

- EJB3.0/DAO Code Generator

○ **CodeGenerator.java** – This class uses the meta-data file "*entity-data-i.xml*" and performs an XSL

transformation to generate the Java code. The XSL transformation will generate EJB3.0 entity and it corresponding DAO objects referencing the Generic DAO.

○ **code-generator.xsl** – XSLT file used for EJB3.0/DAO generation.

3.1.4 Build

3.1.4.1 Ant Build Process

The build process will package the common and EJB2 to EJB3 java code in separate jar files. The resources would not be packaged within the jar file. This would provide flexibility to the user to modify configuration settings without worrying about executing the build again. \

- ejb2-to-ejb3-tool-build.xml
- ejb2-to-ejb3-tool-build.properties

3.2 Execution Process

The EJB2 to EJB3 migration tool would be executed using a batch process (*ejb2_to_ejb3_tool.bat*). The following would be the various stages.

- Data artefacts aggregation
- Set java classpath using the jar file paths set in "jarPaths.txt
- Data transformation

4 THINGS TO REMEMBER

The following should be considered while integrating the generated code.

1. I has been noticed that in the EJB2.1 code, named queries use reference to the entity attribute name that do not match the actual declared variable. Wherever such instances are encountered, rectify the named query.

2. If a "TODO" related to "*The following table column is mapped to an entity field as well as to an entity relationship attribute*" exists; however the related attributes are defined in separate classes (e.g. Entity and PK class), ignore and remove this TODO.

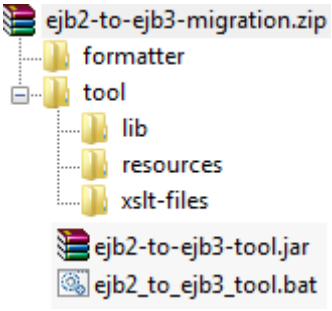
3. After applying fixes for a TODO, it should be removed from the code.

4. If an entity exists more than once in the EJB jar, it should be distinguished by custom entity name by setting the "name" attribute of "@Entity" annotation.

4.1 Tool

4.2 Structure

The tool deliverable is in form of a zip archive file "*ejb2-to-ejb3-migration.zip*". Its structure is as follows.



Directory/File	Description
formatter	Contains Eclipse IDE code formatting preferences file.
Tool	Base folder for migration tool.
Lib	Contains 3rd party and common project jar files.
resources	Contains configuration files.
xslt-files	Contains XSL transformation file.
ejb2_to_ejb3_tool.bat	Batch file to execute the tool.
ejb2-to-ejb3-tool.jar	Migration tool jar.

4.3 Configuration

The tool can be configured by using "ejb2_to_ejb3\Utility\resources\ejb2-to-ejb3-config.properties" file. It lists different keys that can be set as per project requirements.

S. No.	Key	Purpose
1.	data.lookup.dir.path	Directory path for searching the artifacts. e.g. "//ip.address/c$/work/flood"
2.	data.artifacts.dir.name	Directory for listing the filtered artifacts. e.g. "//ip.address/c$/ejb2_to_ejb3/artifacts-list-

		files”
3.	data.artifacts.dd.paths	Name of the file that lists the path to filtered files. e.g. “filteredDDPaths.xml”
4.	data.artifacts.jar.paths	Name of the file that lists the jar file names that are to be set in Java classpath. These jar files contain DTO's and Local interfaces that would be interpreted for EJB3 entity properties. e.g. “jarPaths.txt”
5.	data.meta.dir.name	Directory path of the generated meta-data files e.g. “//ip.address/c$/ejb2_to_ejb3/meta-data-files”
6.	meta.data.file.name	Name of the meta-data file e.g. “entity-data-”
7.	meta.data.file.extension	File extension of meta-data file e.g. “.xml”
8.	filtered.data.file.root	Artifacts xml file root node name e.g. “project”
9.	filtered.data.file.element.module	Artifacts xml file module node name e.g. “module”
10.	filtered.file.element.ejb.dd.path	Artifacts "ejb-jar.xml" xml file module child node name e.g. “ejb-dd-path”
11.	filtered.file.element.web.rdbms.dd.path	Artifacts “weblogic-cmp-rdbms-jar.xml” xml file module child node name e.g. “web-rdbms-dd-path”
12.	filtered.file.element.web.ejb.dd.path	Artifacts “weblogic-ejb-jar.xml” xml file module child node name e.g. “web-ejb-dd-path”
13.	data.generated.code.dir.option	Flag to switch between custom output directory location. Valid values are “CUSTOM” and “LOOK_UP_DIR” e.g. “CUSTOM”
14.	data.generated.code.custom.dir.name	Directory location where the code is generated. Applicable if the directory option is "CUSTOM" e.g. “//ip.address/c$/ejb2_to_ejb3/output”
15.	data.generated.code.sub.src.main	Sub-directory name where the EJB3 code generated by the tool would be placed e.g. “src/main/java/”
16.	data.generated.code.sub.src.test	Sub-directory name where the EJB3 test code generated by the tool would be placed e.g. “src/test/java/”
16.	dd.file.name.pattern	Pattern of the files to be filtered by the file crawler e.g. “(^weblogic-cmp-rdbms-jar.xml)|(^ejb-jar.xml)|(^weblogic-

		ejb-jar.xml)"
17.	dd.file.absolute.path.pattern	Filtered deployment descriptor files absolute path pattern e.g. ".*classes\\\\META-INF.*"
18.	jar.file.path.filter	Filter for jar file location e.g. "classes"
	src.dir.filter	Filter for source directory location path e.g. "target/classes"
19.	entity.dao.package.filter.val	Filter for establishing the package for entities and daos e.g. "interfaces."
20.	jar.file.name.pattern	Pattern of the jar file to be filtered e.g. "\\D.*-SNAPSHOT.jar$"
21.	meta.data.builder	Location of the transformation file for entity meta-data builder e.g. "//ip.address/c$/ejb2_to_ejb3/utility/xslt-files/meta-data-builder.xsl"
22.	code.generator	Location of the transformation file for entity code generator e.g. "//ip.address/c$/ejb2_to_ejb3/utility/xslt-files/code-generator.xsl"
23.	code.constants.file	Location of the xml file that holds constants used in code generation e.g. "//ip.address/c$/ejb2-to-ejb3-migration/tool/resources/xsl-constants.xml"
24.	xml.transform.factory	XML Transform Factory e.g. "javax.xml.transform.TransformerFactory"
25.	saxon.xml.transform.factory	XML Transform Factory Implementation e.g. "net.sf.saxon.TransformerFactoryImpl"

4.4 Execution

The tool can the executed by running the "*ejb2_to_ejb3_tool.bat*". This batch process involves 5 different stages that are listed below.

S.No.	Section
1.	Data artifacts aggregation
2.	Set Java classpath
3.	Meta-data generation
4.	Code generation
5.	Generated code indentation using Eclipse IDE. Note: The user would need to modify the script for section 5 of the batch file as per the local machine.

6 CONCLUSION

This paper gives an intelligent solution for migration of old ejb version ejb 2.1 to new ejb version 3.0. It can be used in the IT industry, especially for those companies which are working on migration domain. Ejb contains the business logic of our code that needs more security & flexibility from other code; hence we can say it's a useful tool for migration your code to new code.

7 REFERENCES

[1] Oracle Ejb Migration 2005

[2] **EJB** components **Migration** Service and Automatic Deployment – HA