How to build O/R mapping meta data files

by Thomas Mahler, Armin Waibel

1. How to build O/R mapping files

Writing the repository.xml file for only a few classes can easily be done manually with the text or xml editor of your choice.

But keeping the repository in sync with the java codebase and the database gets more difficult if several hundred classes and large developer teams are involved.

This page contains tips how to integrate mapping tools and code-generators into your build process.

2. classification of O/R related transformations

Let's start with a classification of the source transformation problems that developers have to face in an O/R environment.

Typical development environments contain some or all of the following artefacts:

- A UML model containing at least class diagrams of the persistent classes. All modern UML tools can export to the XMI standard format.
- Other tools, such as Torque, also use a model based approach but use different model file formats (typically XML based)
- Java source code for the persistent classes. The Java source code can possibly be enhanced with xdoclet tags.
- The OJB repository.xml file. This file contains all the class-descriptors for the persistent classes.
- The database. This could be an online DB or a DDL script representing the database tables. The database contains all tables used to store instances of the persistent classes.

The technique you will use depends a lot on the problem you have to solve, on the methodology and the tool chain you have in use, which of transformations between those artefacts fit to your development process.

1. **Forward engineering from XMI:** A UML model in XMI format with class diagrams of your persistent classes exists and is used as the master source (model driven approach). Java code, repository.xml and DDL for the database tables have to be generated from this model.
2. **Forward engineering from Torque:** A model of the persistent entity classes exists in form of a Torque.XML file. Java code, repository.xml and DDL for the database tables have to be generated from this model.
3. **Forward engineering from the repository.xml:** The OJB repository.xml file is used a model format. Java code and DDL for the database tables have to be generated from this model.
4. **Xdoclet transformation from Java code:** Java code for the persistent classes exists and contains special comment tags in the Xdoclet ojb-module format. Repository.xml and DDL for the database tables have to be generated from the java files via Xdoclet transformation.
5. **Reverse engineering from database:** There is a database with existing tables or a DDL script. Java code and repository.xml have to be generated from the database.

These transformations are depicted in the following graphics. The numbers close to the arrows correspond to the numbers in the above enumeration. All related transformations have the same colour.

In the following sections we will have a closer look at each of these transformations an discuss tools that provide support each approach.

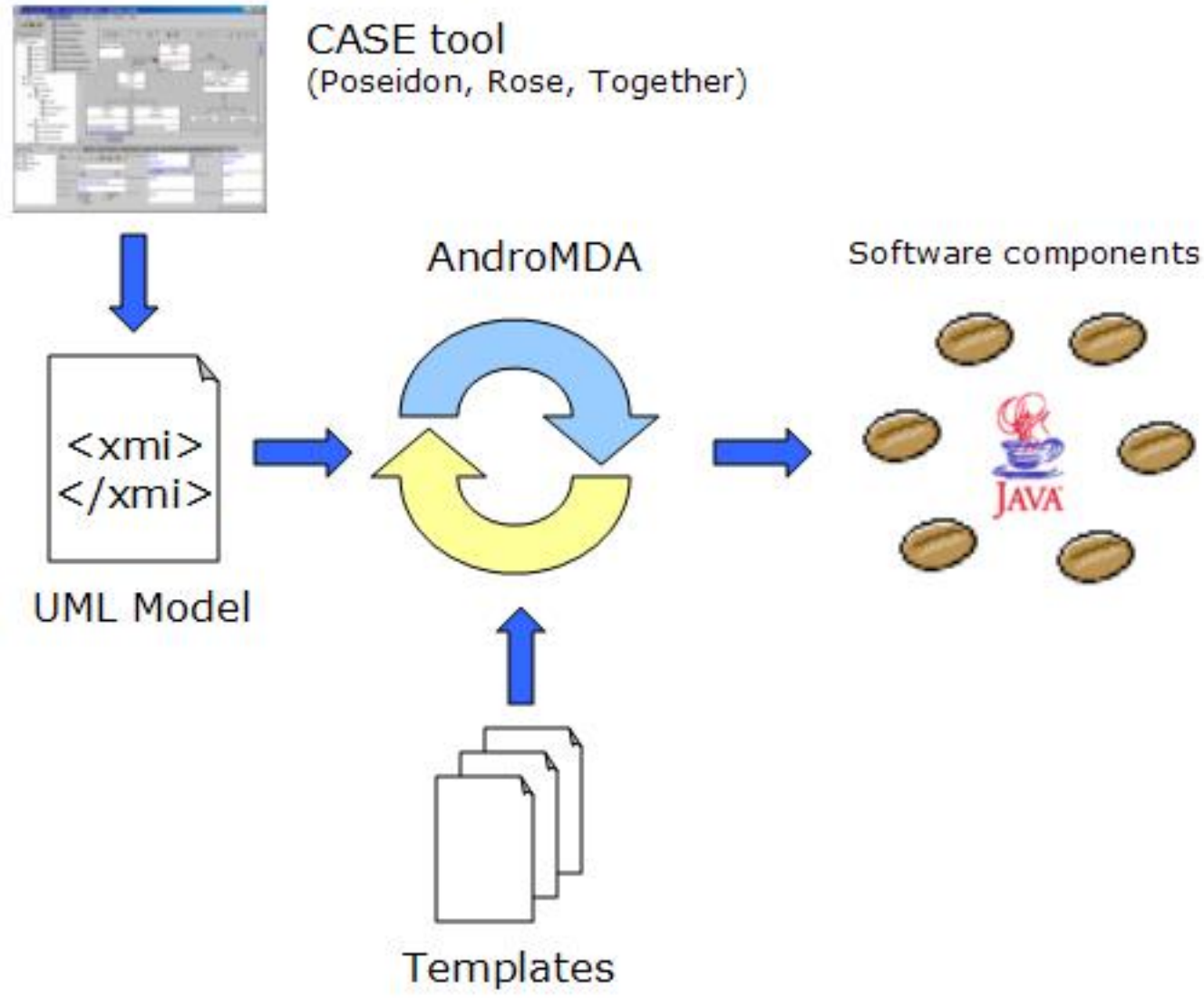
3. Forward engineering from XMI

This approach is recommended if you start from scratch with a new project and have to deal with a large number of persistent classes. This approach works best when there are no restrictions regarding the database, like integration of legacy tables.

Forward engineering from XMI fits perfectly into a model driven architecture (MDA) software development process.

Tool support

- **AXGen**
  AXgen is a code generator using XMI as input and Velocity templates for transformation.
  The power of AXgen is in its simplicity. You don't have to understand complicated structure of your XMI file to write an XSLT stylsheet for transformation. AXgen uses nsuml to deal with the xmi file, which gives access to the Metamodel in an objectoriented way.
  Further AXgen makes use of Jakartas Velocity. Velocity is a very sophisticated Java-based template engine. This means that inside your templates you can call Java methods. Feel free to write templates that generate anything you want.
  Our motive for AXgen is to generate Java Classes for use in a O/R Mapping tool that allows transparent persistence for Java Objects against relational databases. Therefore AXgen comes with a bundle of ready to use templates for generating ObJectRelationalBridge (OJB) specific stuff like:
  - Entity Classes
  - XML Repository
  - SQL script to build the DB scheme
- **AndroMDA**
  AndroMDA is a code generator framework - it takes a Unified Modeling Language (UML) model from a CASE-tool in XMI format and generates custom components. It comes with a set of sample templates that generate classes attributed with XDoclet tags. One build step later, the XDoclet tool generates full-blown components that can readily be deployed in the JBoss application server (and the other servers that XDoclet can feed).



andromeda image

Currently AndroMDA provides no special support for OJB. But by tagging classes with tags of the XDoclet OJB module it is possible to use it as a full forward engineering engine.

- Searching the Sourceforge project list for "XMI" returns a long list of projects dealing with code generation. It may be a good idea to check if you find a tool that matches your requirements.

4. Forward engineering from Torque

Torque

Torque is a persistence layer. Torque includes a generator to generate all the database resources required by your application and includes a runtime environment to run the generated classes.

Torque was developed as part of the Turbine Framework. It is now decoupled and can be used by itself. Starting with version 2.2 Turbine uses the decoupled Torque.

Torque uses a single XML database schema to generate the SQL for your target database and Torque's Peer-based object relation model representing your XML database schema.

You can use devaki-nextobjects to create the model for your application.

OJB uses Torque's generator engine to setup the testbed database and feed it with initial data.

Besides the SQL generation facilities Torque also provides special support for OJB related transformations. It provides the following two ant targets:

- **ojb-model**
  generates a simple object model for ojb
- **ojb-repository**
  generates the repository for ojb

A complete list of all availableTorque targets can be found at the Torque Generator site.

5. Forward engineering from repository.xml

There is currently no tool available that directly supports this model. It is not difficult to implement an XSLT stylesheet that transforms the OJB repository.xml directly into DDL Statements.

An even simpler approach could be to transform the repository.xml file into a Torque xml file. DDL can then be generated by the Torque engine.
If you write such an XSLT file please tell us about it!

6. XDoclet transformation from Java code

XDoclet
XDoclet is a code generation engine. It enables Attribute-Oriented Programming for java. In short, this means that you can add more significance to your code by adding meta data (attributes) to your java sources. This is done in special JavaDoc tags.

OJB was shipped with its own **xdoclet-module**.

XDoclet will parse your source files and generate many artifacts such as XML descriptors and/or source code from it. These files are generated from templates that use the information provided in the source code and its JavaDoc tags.

XDoclet lets you apply Continuous Integration in component-oriented development. Developers should concentrate their editing work on only one Java source file per component.

XDoclet originated as a tool for creating EJBs, it has evolved into a general-purpose code generation engine. XDoclet consists of a core and a constantly growing number of modules.

7. Reverse engineering from database

- **Druid**
  Druid is a tool that allows users to create databases in a graphical way. The user can add or import tables, fields, folders to group tables and can modify most of the database options that follow the SQL-92 standard. In addition to sql options, the user can document each table and each field with HTML information. It is distributed with modules for generating Java classes, OJB metadata, and JDO metadata.
- **Impart Eclipse Plugin for OJB**
  The Impart Eclipse plugin is based on the OJB ReverseDB Tool and provides the same functionality (and also some additional goodies). It ships as a plugin to the Eclipse IDE. It provides a very convenient GUI that integrates smoothly into the Eclipse platform.
- **RDBS2J**
  RDBS2J is a GUI based mapping tool from relational database scheme to persistent java classes which use JDO as persistence mechanism. The mapping can be modified by the GUI.
  The current version is designed to create code for OJB.
  The ODMG and the JDO interface are supported. RDBS2J creates the *.jdo files and the repository_user.xml, which are needed by OJB.
- **The OJB ReverseDB tool**
  OJB ships with a simple reverse engineering tool that allows to connect to a RDBMS via JDBC and to take the tables from

the database catalog as input.
This tool provides a nice GUI to generate Java classes and the matching repository.xml file.
You can invoke the ReverseDB tool with the ANT target `reverse-db`.

Note:

The ReverseDB tool is not up to date - any help is welcome.