

openArchitectureWare 4.1.2 uml2Ecore Reference

Markus Voelter, voelter@acm.org, www.voelter.de

Table of Contents

WHAT'S THIS?.....	3
SETTING UP ECLIPSE.....	3
UML TOOL SUPPORT.....	3
SETTING UP YOUR META MODEL PROJECT.....	3
INVOKING THE GENERATOR.....	5
THE GENERATED MODEL.....	6
NAMING.....	7

What's this?

Building meta models with EMF's internal tools is tedious. Using the tree view based editors doesn't scale. Once you're at more than, say, 30 metaclasses, things become hard to work with. The same is true for GMF's Ecore editor. Since you cannot easily factor a large meta model into several diagrams, the diagram get cluttered and layouting becomes almost impossible.

One way to solve this problem is to use UML tools to draw the meta model and then transform the UML model into an Ecore instance.

The oAW `uml2ecore` utility transforms a suitably structured Eclipse UML2 model (which can be created using various tools) into an ecore file.

Note that this tool also serves as a tutorial for writing model-to-model transformations. This aspect, however, is documented elsewhere. This document only shows how to use the `uml2ecore` tool.

Setting up Eclipse

You need an installation of oAW 4.1 including the UML2 support. Run the UML2 example (available for download on the oAW download page) to verify that you have all the UML2 stuff installed.

The only additional thing required is that you install the `uml2ecore` plugin into your Eclipse installation. The plugin is part of the oAW 4.1.2 distribution.

UML Tool Support

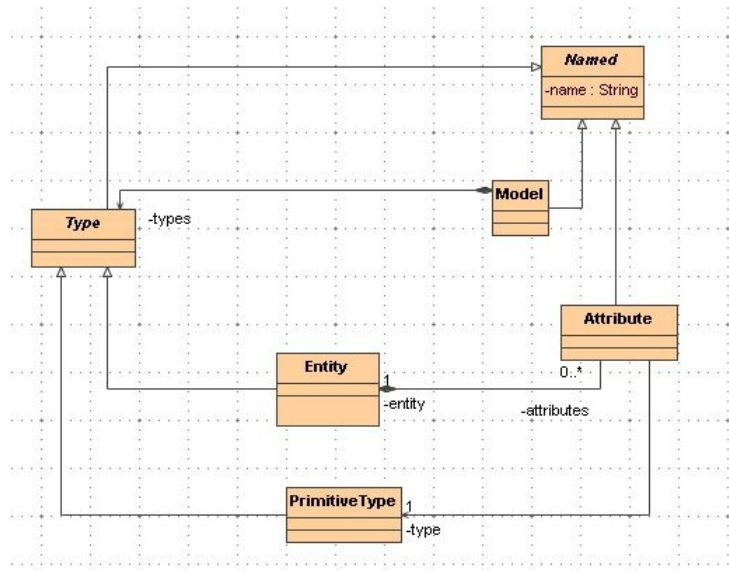
Of course you could use UML2's supplied tree editors for drawing the UML2 model that should be transformed into Ecore. However, this is useless, since then you're back to square one: tree editors. So you need to use an UML tool that is able to export the model in the **Eclipse UML2 2.0** format. For example, MagicDraw 11.5 can do that; this tool is also the one we tested the `uml2ecore` utility with.

Note that we do not use any profiles in the `uml2ecore` utility. Although using profiles might make the metamodel a bit more expressive, we decided not to use a specific profile, to reduce the potential for compatibility problems (with the various tools).

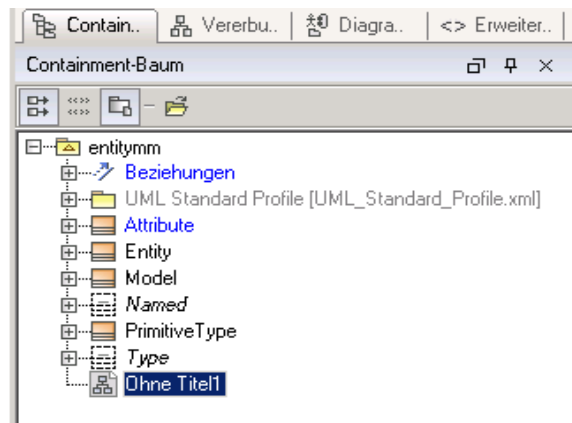
Setting up your meta model project

You should first create a new Generator project (select `File->New->Other->openArchitectureWare/Generator Project`).

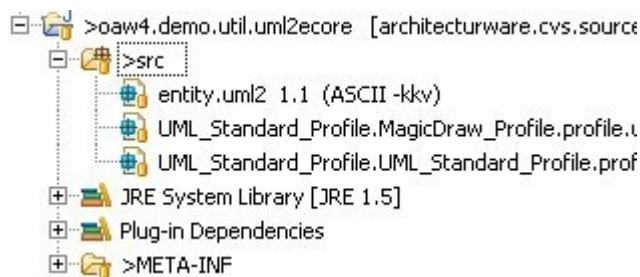
Then open your UML2 tool of choice (we'll use MagicDraw here) and draw a class diagram that resembles your metamodel. Here's the one we've drawn as an example for this document:



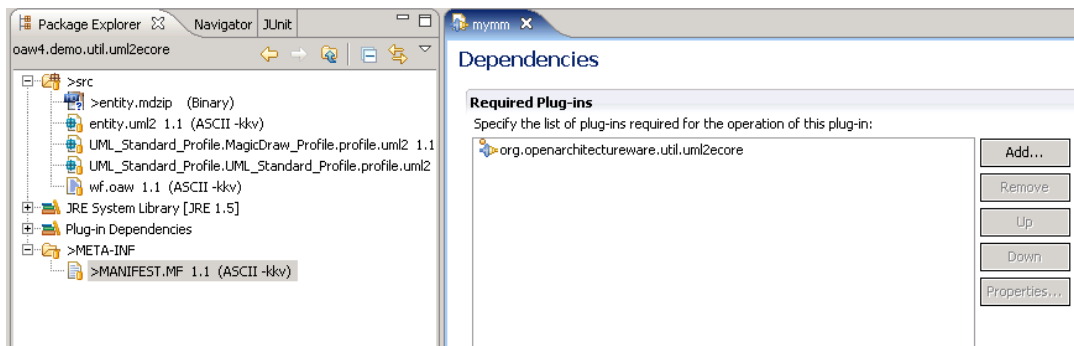
This is the usual meta model for entities and such. Nothing special. Also, the name of the model defines the name of the ecore metamodel; here's the MagicDraw tree view to illustrate this:



You now have to save/export this model in EclipseUML2 format. In MagicDraw, you do this by selecting *File->Export->Eclipse UML2*. The exported files have to be in the **root of the src folder** in the metamodel project created above! This is how the project looks like after this step, assuming you'd called your model *entity.uml2*:



Before you can actually run the generator, you have to make sure that your project has a plugin dependency to the *uml2ecore* plugin. Double-Click on the plugin's manifest file, select the *Dependencies* tab and click *add*. Select the *org.openarchitectureware.uml2ecore* plugin. The result looks like this:



In the current version, you also need a dependency to the *org.openarchitectureware.util.stdlib* project. Later versions of the *uml2ecore* plugin will reexport that dependency, so that you will not need to add it you your projects manually.

Invoking the Generator

As usual, you have to write a workflow file. It also has to reside in your project's source folder. Here's how it looks:

```
<cartridge file="org/openarchitectureware/uml2ecore/uml2ecoreWorkflow.oaw"

    uml2ModelFile="org/openarchitectureware/uml2ecore/test/data/entity.uml2"
    outputPath="out/"
    nsUriPrefix="http://www.voelter.de"
    includedPackages="Data"
    resourcePerToplevelPackage="false"
```

```
addNameAttribute="true"/>
```

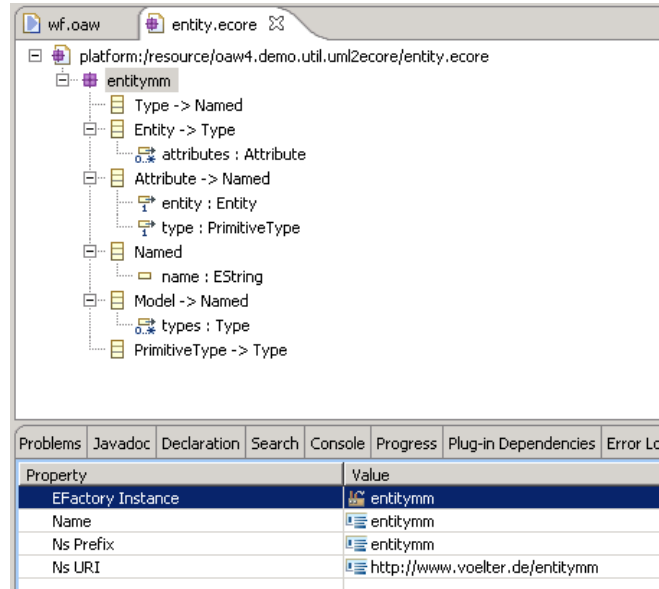
As you might expect, it simply calls a cartridge supplied by *uml2ecore* plugin. You have to define the following properties:

<i>uml2ModelFile</i>	the is the name of your UML2 file that contains the model; as usual, the file is looked for in the classpath (that's why you had to move it into the source folder)
<i>addNameAttribute</i>	<i>true/false</i> , the determines whether automatic namespace management and naming is turned on (see the end of this document). For the simple example, please set the value to be false.
<i>nsUriPrefix</i>	the <i>nsUriPrefix</i> is used to assemble the namespace URI. The name of the meta model, as well as the <i>nsPrefix</i> , will be derived from the UML model name; so in our example, the <i>nsPrefix</i> and the <i>name</i> of the generated <i>EPackage</i> will be <i>enttymm</i> . The complete namespace URL also required by Ecore is created by concatenating the <i>nsUriPrefix</i> given here, and the <i>name</i> . The resulting namespace URL in the example will thus be http://www.voelter.de/entityymm
<i>includedPackages</i>	determines which packages the transformer should consider when transforming the model; note that the contents of all the packages will be put into the root <i>EPackage</i> .
<i>outputPath</i>	determines where the resulting files are written to
<i>resourcePerToplevelPackage</i>	It set to true, the generator will write a separate ecore file for each of the top level packages in you UML model. Useful for modularizing meta models (see the end of this document).

You can now run this workflow by selecting *Run As -> oAW Workflow*. The name of the generated Ecore file will correspond to the name of the root Model element in the UML model.

The generated Model

Here's a screenshot of the generated model:



The generator also creates a constraints file (called *entitymmmConstraints.chk*) which contains a number of constraints; currently these are specifically the checks for the minimum multiplicity in references (an error is reported if the minimum multiplicity is one, but the reference is null or empty, respectively). You can integrate the generated constraints file into your workflow using the usual approach.

Naming

It is often the case the basically all model elements in a model should have a name. It might not always be necessary from a domain perspective, but it's really useful for debugging. Of course, you can add a superclass called *Named* with a single attribute *name* to all you classes. However, if you set the *addNameAttribute* parameter to be *true* when calling the *uml2ecore* cartridge, every class which does not inherit from another class gets an additional *name* attribute. Also, constraints that make sure that names within a namespace (i.e. a containment reference) are unique.

uml2ecore also comes with a utility extension called *org::openarchitectureware::util::stdlib::naming* that can calculate the *namespace()* and the *qualifiedName()* for every model element that has a name.

Modularizing the meta model file

You can modularize the meta model. If you specify the *resourcePerToplevelPackage="true"* parameter to the cartridge call, you'll get a separate ecore file for each top level package, as well as a separate constraint check file.