

openArchitectureWare 4.1 Model-To-Model with UML2 Example

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

Table of Contents

INTRODUCTION.....	3
WHY THIS EXAMPLE?.....	3
SETTING UP ECLIPSE.....	4
THE BUILDING BLOCKS OF THE TRANSFORMER.....	4
USING THE TRANSFORMER.....	5
TESTING AN M2M TRANSFORMATION.....	5

Introduction

This tutorial introduces various somewhat advanced topics of working with openArchitectureWare. We don't generate code, we transform models. Specifically, the tutorial covers the following:

- real model-to-model transformations
- how to test m2m transformations
- working with Eclipse UML2 (specifically, transforming away from it)
- using global variables
- implementing JAVA extensions

Note that this tutorial actually documents the *uml2ecore* utility that can be used to transform metamodels drawn with an UML2 tool into an Ecore instance. This tool is available for download as a plugin on the oAW download page.

Please make sure you first read the *u10_uml2ecore* reference. That document explains what the tool does – this is more or less a precondition to making sense *how* the tool does it (which is what this document explains).

Also note that many of the gory details are actually documented as code comments in the various oAW files of the example. So, make sure you have the code available as you read this document.

Why this example?

In general, we needed an example for real m2m transformations. And obviously, since we already had the *uml2ecore* tool, it makes sense to document this one. Also, including some information of working with UML2 is also important. However, there's one really good reason why looking at this example is important.

If you work with UML2 models, you'll soon realize that the UML2 metamodel is very intricate, non-intuitive and complicated. So it is very good practice to „transform away“ from the UML2 model as early as possible. Especially, if you want to enhance your model using model modifications (remember the emergency-stop feature in the statemachine example?) this approach is recommended. While read access to UML2 models is tedious, write access is almost impossible to get correct, and in some aspects also impossible with generic EMF tools such as oAW.

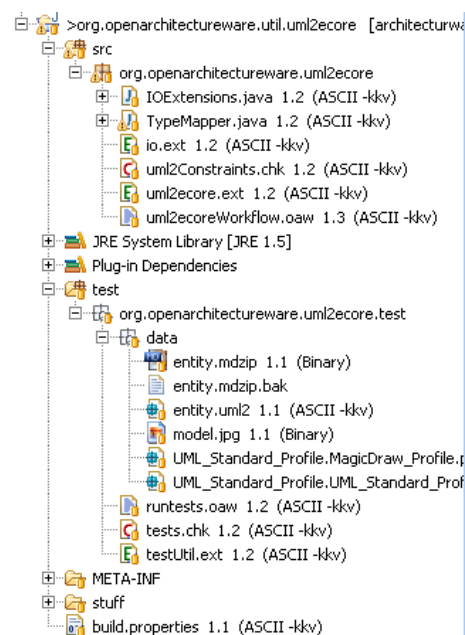
So: if you work with a UML2 model, always begin by transforming into your own domain-specific metamodel.

In this example, we transform into an instance of Ecore in order to get the meta model (since the uml2ecore tool is used for EMF meta modelling in UML tools). So in your application projects, the transformation would not create an instance of Ecore, but an instance of your domain-specific metamodel (which, in turn, might have been created with uml2ecore). The only difference to the example given here is that you'd use as a meta model not Ecore, but rather your own one. This has effects on the meta model declarations in the workflow file; everything else is similar.

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.

Then download the sample code for this tutorial and put the project into your workspace. Should look something like this, then.



The Building Blocks of the Transformer

The first thing you should look at (*after* reading the uml2ecore user guide!) is the workflow file. It reads the UML2 model, checks a couple of constraints against it and then invokes the transformation. We then write the result of the transformation out to an *.ecore* file.

::CODE:: src/org/openarchitectureware/uml2ecore/uml2ecoreWorkflow.oaw

You could now take a look at the constraints against the UML model. There are currently quite few of them in the code, but at least it shows how to write constraints against UML2 models

::CODE:: src/org/openarchitectureware/uml2ecore/uml2Constraints.chk

The transformation itself is located in *uml2ecore.ext*. While the transformation is not too complicated, it is also not trivial and illustrates many of the facets of the Xtend transformation language.

::CODE:: src/org/openarchitectureware/uml2ecore/uml2ecore.ext

To see how global variables are used, take a look at the `<globalvar .../>` declaration in the workflow file, as well as the `nsUri()` function in *uml2exore.ext*.

To see *JAVA* extensions in action, take a look at *io.ext* and *IOExtensions.java*

Using the transformer

Usage of the transformer can be seen from the *uml2ecore* user's guide mentioned a couple of times already. However, you can also take a look at the stuff in the testing section.

Testing an M2M transformation

M2M transformations can become quite intricate, especially if UML2 is involved. Therefore, testing such a transformation is essential. Note that you should resist the temptation to write tests in the form of generic constraints such as

for each Class in the UML2 model there has to be an EClass in the Ecore model

While this statement is true, it is also on the same level of genericity as the transformation itself. It's therefore not much less complex, and therefore as error prone as the transformation itself.

Instead what you should do is the following:

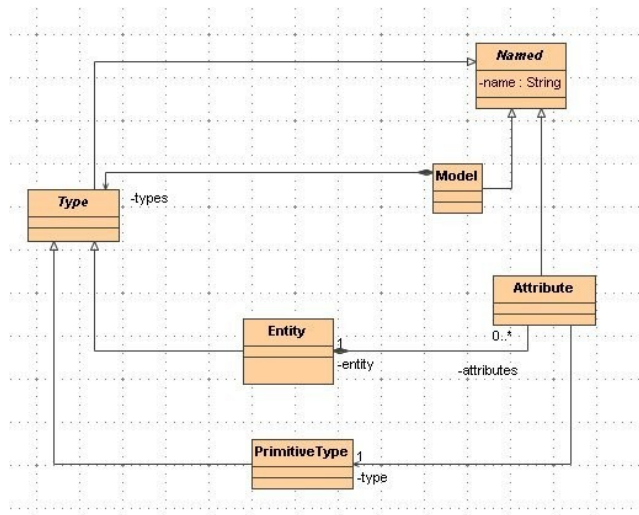
- Define a reference model. This model should contain all the possible alternatives of creating input models, i.e. should cover the complete variability of the meta model.
- Then run the transformation on that reference model
- Finally, write *model-specific* constraints that verify the particular result of the transformation.

So in our example, we have a small meta model modelled in UML using abstract and non-abstract classes, inheritance, attributes, 1:n and 1:1 references, bidirectional and

unidirectional. The model, in our case, is drawn with MagicDraw 11.5 and exported as an UML2 model. You find all the model contents in the following location.

::CODE:: test/org/openarchitectureware/uml2ecore/test/data

Here's a screenshot of the model:



You now run the transformation using a workflow file that looks as follows:

::CODE:: test/org/openarchitectureware/uml2ecore/test/runtests.oaw

This workflow basically just invokes the transformation just as any other user of the transformation. However, in addition, after the transformation has finished, we validate test-specific constraints. This is what the additional constraint checker component is good for in the above mentioned workflow.

Now comes the point: The test constraints are not generic! Instead they do have knowledge about the input model (which generic constraints don't, they just operate with knowledge about the *meta* model). So this file contains constraints such as

There has to be a class called Entity in the resulting model.

Or:

There must be an EReference owned by Attribute called type, pointing to the class called PrimitiveType.

Make sure that the constraints – again: coverage! – check all relevant aspects of the transformation.

::CODE:: test/org/openarchitectureware/uml2ecore/test/tests.chk