

Using ATL with oAW and EMF

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



Table of Contents

CONFIGURING ECLIPSE AND OAW.....	3
WHAT YOU NEED TO DO BEFORE... ..	3
INSTALLING THE PRE-BUILT TUTORIAL.....	3
CREATING THE PROJECT.....	3
DEFINING THE TRANSFORMATION.....	4
EXAMPLE DATA.....	5
THE WORKFLOW FILE.....	5
RUNNING THE TRANSFORMATION.....	7

Configuring Eclipse and oAW

Before any of the following can work, you need to install the ATL transformation framework. ATL is actually another subproject of Eclipse GMT. You can get all the software from <http://eclipse.org/gmt/atl/>. Specifically, there is an installation guide that you should read at [http://eclipse.org/gmt/atl/doc/ATL_Installation_Guide\[v0.1\].pdf](http://eclipse.org/gmt/atl/doc/ATL_Installation_Guide[v0.1].pdf). And of course you need EMF.

In addition to installing ATL, you also need the oAW ATL adapter, which can be found in the *adapter.emf.atl* project. Make sure this one is installed in your workspace; see the *Installation* docs for details.

What you need to do before...

In order to familiarize yourself with ATL, you should run through the ATL starter's guide. You can find it at http://eclipse.org/gmt/atl/doc/ATL_Starter_Guide.pdf

Installing the pre-built tutorial

Instead of building the tutorial manually, you can also download the *oaw-samples-emf-4.x.x* package and install the contained Eclipse projects into your workspace (you might want to delete the *ocl* demo project for this first example). To make the projects compile and run, you have to define the following two Eclipse environment variables:

Variable	... points to
OAW_CORE	oaw4/oaw-core-4.x.x
OAW_LIB	oaw4/oaw-core-4.x.x/lib

Creating the Project

We start by creating the example project. Here it's called *oaw4.demo.emf.datamodel.atl*. Make sure it's a Java project. Also, for the ATL compiler to work you need to have the following two entries in the *.project* file:

```
<buildSpec>
...
  <buildCommand>
    <name>org.atl.eclipse.adt.builder.atlBuilder</name>
    <arguments>
    </arguments>
  </buildCommand>
...

```

```
</buildSpec>
<natures>
  ...
  <nature>org.atl.eclipse.adt.builder.atlNature</nature>
  ...
</natures>
```

You also have to add a number of dependencies:

- Project Dependency to: *adapter.emf.atl*
- Project Dependency to: *oaw4.demo.emf.datamodel* (defines the metamodel we will work with in the example)
- as usual, all the jars from *oaw4/oaw-core-4.x.x*, also those in it's *lib* subdir.

Defining the transformation

The following ATL file defines the transformation we want to use. It is the identity transformation, i.e. it does not do anything really useful. We just append a *new_* to the attribute names. Here is the *DataMod.atl* file.

```
module DataMod;
create OUT : data from IN : data;

rule Entity {
  from s: data!Entity
  to t: data!Entity (
    name <- s.name,
    attribute <- attrs
  ),
  attrs : distinct data!Attribute foreach (a in s.attribute) (
    name <- 'new_'+a.name,
    type <- a.type
  )
}
```

After saving this file in the *src* folder, there should be a *DataMod.asm* file right next to it. If you have set up ATL and the project builders/natures correctly, the ATL IDE should have created the asm file automatically. ASM files are a kind of “transformation assembler code”; the transformer actually executes this one, and not directly the atl file.

Note: you should use the ATL IDE interactively to verify that the transformation works before you embed it into an oAW workflow!

Example Data

We need some example data to transform. We use the usual trivial entity, defined in the *src/example.data* file.

```
<?xml version="1.0" encoding="UTF-8"?>
<data:DataModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:data="http://www.openarchitectureware.org/oaw4.demo.emf.datamodel"
el">
  <entity name="Person">
    <attribute name="name" type="String"/>
    <reference target="//@entity.1" name="autos"/>
  </entity>
  <entity name="Vehicle">
    <attribute name="plate" type="String"/>
  </entity>
</data:DataModel>
```

The Workflow File

Here it becomes interesting, since this is the place where the oAW/ATL integration actually takes place. The following is the workflow file, called *src/workflow.oaw*.

```
<?xml version="1.0" encoding="windows-1252"?>
<workflow>
  <property file="workflow.properties"/>
```

So, first we load the example model file into a *slot* called *model*. This is identical to the first step in the oAW code generation example.

```
<component id="xmiParser"
class="org.openarchitectureware.emf.XmiReader">
  <modelFile value="{modelFile}"/>
  <metaModelPackage value="data.DataPackage"/>
  <outputSlot value="model"/>
</component>
```

Next is the invocation of the ATL transformation itself. We include the *SimpleATLTransformerComponent* into the workflow. It needs a number of parameters.

```
<component id="trafo"
class="org.openarchitectureware.adapter.emf.atl.SimpleATLTransformerComponent">
```

The first one is the working directory. Since ATL is file based, we create a number of intermediate XMI file from which ATL works. We have to specify the directory where these should be put.

```
<workDir
value="{workspaceRoot}/oaw4.demo.emf.datamodel.atl/temp"/>
```

We then make known the (one or more) metamodels. Note how we use the *value="key -> value"* syntax to actually specify key/value pairs as parameters to the oAW workflow. So here we map the metamodel called *data* (which we use under that name in the ATL transformation) to the respective ecore file.

```
<metamodelPath value="data ->
${workspaceRoot}/oaw4.demo.emf.datamodel/metamodel/data.ecore"/>
```

Here we specify which metamodel should be used for each of the input/output models.

```
<inputModelToMetamodelMap value="IN -> data"/>
<outputModelToMetamodelMap value="OUT -> data"/>
```

Then we map the model(s). We use the logical names used in the transformation and map them either to files or (as shown here) to slots. We have to use the *slot* keyword to denote that the models should be taken from the slots¹.

```
<modelPath value="IN -> slot:model"/>
<modelPath value="OUT -> slot:newModel"/>
```

Finally, we specify the transformation. Note that you have to specify the asm file here (we decided to use this approach instead of specifying the atl file to "enforce" that you actually have the IDE/build set up so that the asm file is produced...)

```
<asmFile
value="${workspaceRoot}/oaw4.demo.emf.datamodel.atl/src/DataMod.asm"/
>
</component>
```

That's all for the transformation. To see what we actually did, we subsequently write the content of the *newModel* slot out to a file.

```
<component id="xmiWriter"
class="org.openarchitectureware.emf.XmiWriter">
<inputSlot value="newModel"/>
<modelFile value="${newModelFile}"/>
</component>
</workflow>
```

To make the workflow work, we also have to define the properties:

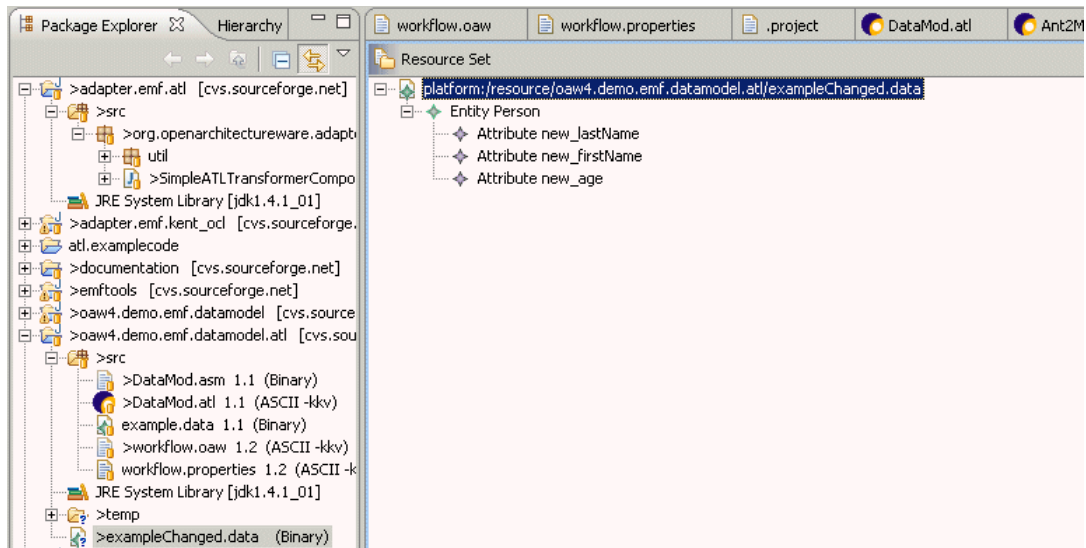
```
modelFile=example.data
newModelFile=exampleChanged.data
workspaceRoot=L:/workspace-oaw4-doku
```

Make sure you adapt the workspace root accordingly...!

¹ What really happens here is that the models are read from the slot into a temp file in the previously specified working directory. The transformation works from that file. Later, the transformed model is read from the file and put back into the slot.

Running the transformation

You run the transformation in the usual way – by running the *.oaw* file. As a result, you should get a *exampleChanged.data* file with the contents displayed in the next illustration.



About our Sponsors

itemis GmbH & Co. KG is an independent IT service company with an emphasis on consulting, coaching, and software development. Every single itemis expert provides many years of project experience and widespread knowledge about all object oriented and component based software development issues - especially in the field of model driven software development.

b+m is the founder of the openArchitectureWare project. The software was originally developed within the scope of many successful projects. b+m opened the software to the community in late 2003. All of the paradigms of Model-Driven Software Development including Product Line Engineering and not only the generator framework have become a key concept for product and customer specific development at b+m. b+m customers can make use of long time experience and substantial know-how in that field. Located at the company headquarters in Melsdorf/Kiel and at its subsidiaries in Berlin, Cottbus, Hamburg, Hanover and Kiel the b+m staff of 205 provides practical solutions for customized business applications, business process optimization and comprehensive architecture, project and quality management.

oose Innovative Informatik GmbH offers coaching, consulting and training in all themes about software engineering. The main focus of their activities are software architecture, requirements engineering and project-management. oose have first-hand information and experience, because our staff take actively part with others in actual trends, standards and innovations. Our staff support this and pass their know-how regularly on by writing and publishing books or being speaker at conferences, etc. Within the OMG oose collaborate actively on the specifications of the UML and also the SysML.

MID Enterprise Software Solutions GmbH is a leading supplier of optimized tool environments for standardsbased and model-centric software development as well as business process modeling. This includes professional tool consulting and tool components to build a complete tool environment using the best techniques and tool modules available - Architectural and Operational Excellence. With innovatorAOX, MID provides a holistic standard tool environment for object- and function-oriented software development as well as business process and data modeling to help its customers establish highly efficient processes and tool environments for software production, The unique and seamless integration of business process modeling into the development process ensures an unprecedented level of convergence of business requirements and implemented IT systems. Project members from all departments speak the same language and all requirements are clearly described.