

openArchitectureWare 4.1 UML2 Example

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

PRIMARY SPONSORS



SECONDARY SPONSOR

Table of Contents

| | |
|------------------------------------|----------|
| SETTING UP ECLIPSE..... | 3 |
| SETTING UP THE PROJECT..... | 3 |
| CREATING A UML2 MODEL..... | 4 |
| MODELLING THE CONTENT..... | 5 |
| CODE GENERATION..... | 5 |
| DEFINING THE TEMPLATES..... | 5 |
| DEFINING THE WORKFLOW..... | 6 |
| PROFILE SUPPORT..... | 7 |
| DEFINING A PROFILE..... | 7 |
| APPLYING THE PROFILE..... | 8 |
| GENERATING CODE..... | 9 |

Setting up Eclipse

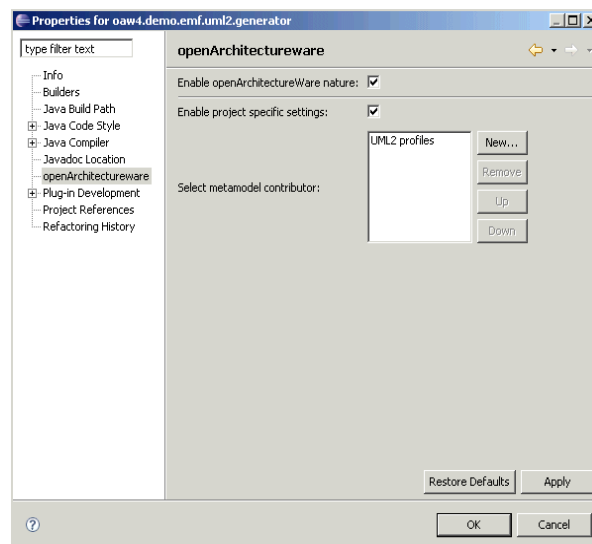
Before you can use oAW with Eclipse UML2, you first have to install the UML2 plugins into your Eclipse installation.

Setting up the project

Create a new Plugin project. You have to add the following dependencies to the manifest file:

- all the openArchitectureWare plugins, especially the UML2 adapter plugins
- And, **importantly**, you have to add a couple of the UML2 plugins.
 - org.eclipse.uml2
 - org.eclipse.uml2.common
 - org.eclipse.uml2.uml
 - org.eclipse.uml2.uml.resources
 - org.eclipse.uml2.ecore.importer
 - org.eclipse.uml2.ecore.exporter

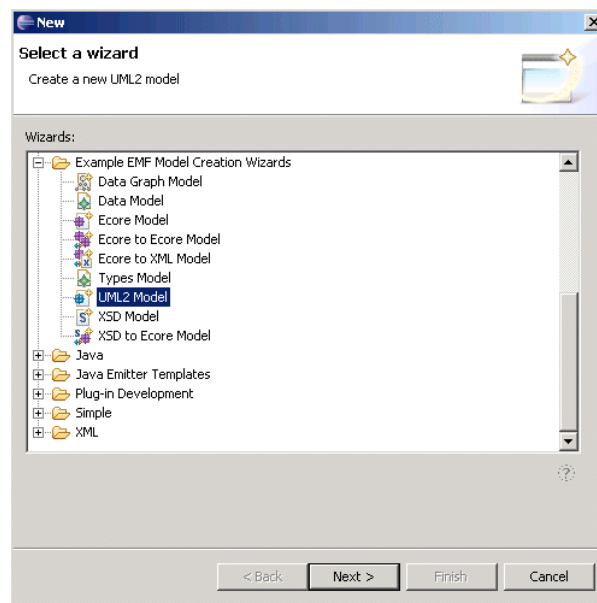
To tell the oAW Eclipse plugins that this project is a UML2 specific one, you need to specify that in the oAW preferences. Open the project properties, select the openArchitectureWare tab and select the *UML2 profiles* metamodel.



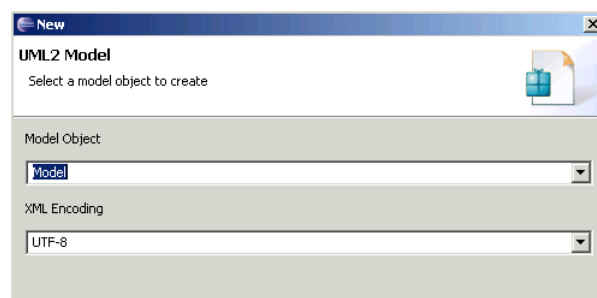
Note, that if you want to transform an UML2 model into a normal EMF model, you need to add the UML2 metamodel *and* the EMF metamodels. The order is important! The UML2 profiles entry must be first in the list.

Creating a UML2 Model

You start by defining a uml2 model, i.e. an instance of the UML2 metamodel. In the new Java project, in the source folder, you create a UML2 model that you should call *example.uml*.

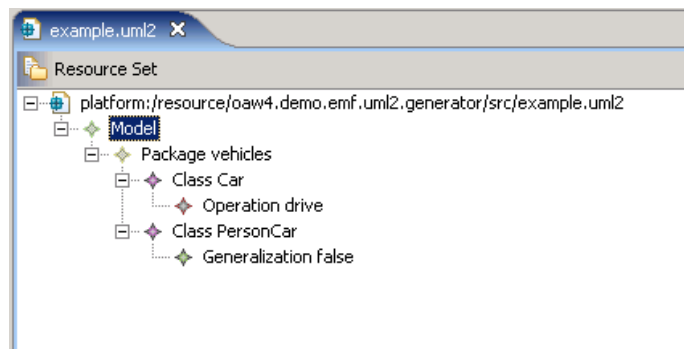


You then have to select the model object. Make sure its a *Model*, not a *Profile*.



Modelling the content

You should then build a model that looks somewhat like this:



By the way, if you rename the *.uml* file to *.ecore*, you can edit the model using the ecore editors. To inspect the model, they provide a somewhat better view, so you might try!

Code generation

Defining the templates

Inside the source folder of our project, create a *templates* package. Inside that package folder, create a template file *Root.xpt* that has the following content. First, we define the entry template that is called *Root*. Since we expect a UML model element to be the top element to the model, we define it for *uml::Model*. Note the use of the *uml* Namespace prefix, as defined in the UML2 metamodel. Inside that template, we iterate over all owned elements of the model and expand a template for the packages defined in it.

```

«DEFINE Root FOR uml::Model»
    «EXPAND PackageRoot FOREACH ownedElement»
«ENDDEFINE»

```

In the package template, we again iterate over all owned elements and call a template that handles classes. At this point we expect that only classes are in that package.

```

«DEFINE PackageRoot FOR uml::Package»
    «EXPAND ClassRoot FOREACH ownedType»
«ENDDEFINE»

```

This template handles classes. It opens a file that has the same name as the class, suffixed by *.java*. Into that file, we generate an empty class body.

```

«DEFINE ClassRoot FOR uml::Class»
    «FILE name+".java"»
        public class «name» {}
«ENDFILE»

```

```
«ENDDFINE»
```

Defining the workflow

In order to generate code, we need a workflow definition. Here is the workflow file; you should put it into the source folder. The file should be generally understandable if you read the emfExample docs.

```
<?xml version="1.0" encoding="windows-1252"?>
<workflow>
```

You need to setup the UML2 stuff (registering URI maps, Factories, etc.). This can be done declaring a bean in before of the *XmiReader* component:

```
<bean class="oaw.uml2.Setup" standardUML2Setup="true"/>

<component class="oaw.emf.XmiReader">
  <modelFile value="example.uml"/>
  <outputSlot value="model"/>
</component>
```

The *XmiReader* reads the model and stores the content (a list containing the model element) in a slot named 'model'. As usual, you might want to clean the target directory.

```
<component id="dirCleaner"
  class="oaw.workflow.common.DirectoryCleaner"
  directories="src-gen"/>
```

and in the generator we also configure the EMF meta model for oAW, together with the UML2 metamodel package, because the UML2 metamodel refers to it.

```
<component id="generator" class="oaw.xpand2.Generator"
  skipOnError="true">
  <metaModel class="oaw.type.emf.EmfMetaModel">
    <metaModelPackage value="org.eclipse.emf.ecore.EcorePackage"/>
  </metaModel>
  <metaModel class="oaw.uml2.UML2MetaModel"/>
  <expand value="templates::Root::Root FOR model"/>
  <outlet path="src-gen/">
    <postprocessor class="oaw.xpand2.output.JavaBeautifier"/>
  </outlet>
</component>
```

If you run the workflow (by right clicking on the .oaw file and select *Run As -> oAW Workflow* the two Java classes should be generated.

Profile Support

OAW4 is shipped with a special UML2Profiles metamodel implementation. The implementation maps Stereotypes to Types and Tagged Values to simple properties. It also supports Enumerations defined in the profile and Stereotype hierarchies.

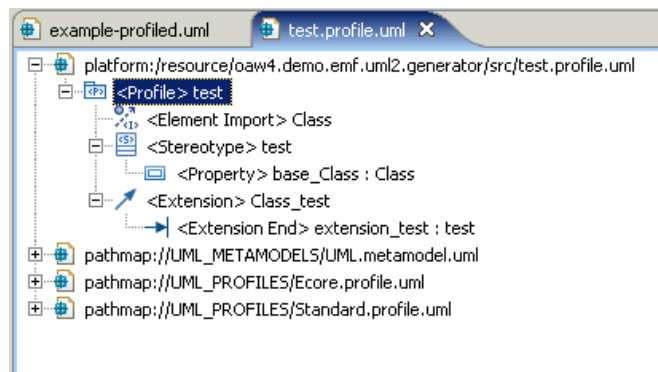
Defining a Profile

To define a profile, you can use a variety of UML2-based modelling tools. Assuming they do actually correctly create profile definitions (which is not always the case, as we had to learn painfully), creating a profile and exporting it correctly is straight forward.

In this section, we explain the „manual way“, which is good for explaining what happens, but completely useless for practical use. You don't want to build realistically-sized models using the mechanisms explained below.

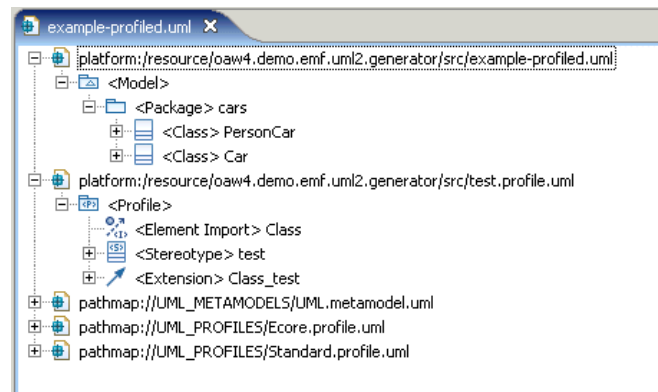
You start by creating a new UML2 file (as shown above). In the example we'll call it *test.profile.uml*. The root element, however, will be a *Profile*, not a *Package*. Don't forget to actually assign a name to the profile! It should be *test*, too.

As a child of that *Profile*, you then create a *Packaged Element Stereotype* (you'll have to scroll a bit in the *Add Child* menu....). For the sake of example, we'll call it *test*, too. In our case, we want to make the stereotype be applicable to UML classes - they are defined as part of the UML2 metamodel. So we have to import that metamodel first. So what you do is to select your profile object, and then go to the *UML2 Editor* menu (in the Eclipse menu bar) and select *Profile -> Reference Metaclass*. Select *uml::Class*. You can then select your stereotype, and select *Stereotype -> Create Extension* from the *UML2 Editor* menu. Select *uml::Class*. This should lead to the following model. Save it, you're done with the profile definition.



Applying the Profile

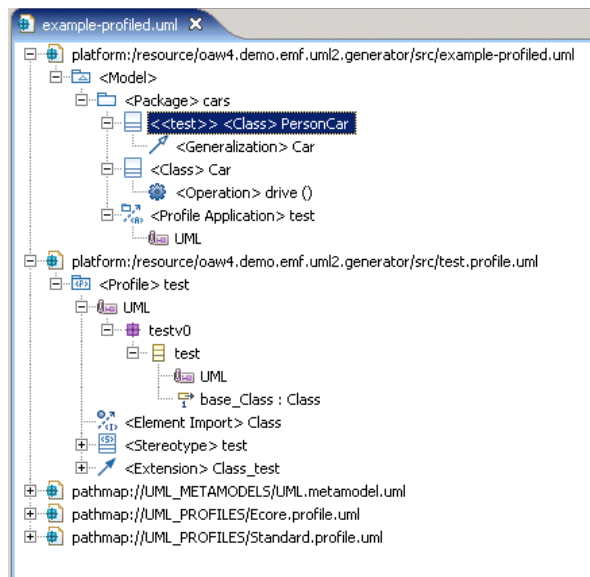
To make any use of the profile, we have to apply it to some kind of model. To do that, we copy the *example.uml* model to a *example-profiled.uml*. We then open that file and *Load a Resource*, namely the profile we just defined. This then looks somewhat like this:



Now, to make the following stuff work, you first have to select the profile and select the *Profile -> Define* operation from the *UML2 Editor* menu. This creates all kinds of additional model elements, about which you shouldn't care for the moment.

Now, finally, you can select your *cars* package (the one from the example model) and select *Package -> Apply Profile* from the *UML2 Editor* menu. Select your *test* profile to be applied.

For the purpose of this example, you should now apply the *test* stereotype to the *PersonCar* class. Select the class, and select *Element -> Apply Stereotype* from the *UML2 Editor* menu. This should result in the following model:



Generating Code

Note that all the stuff above was not in any way related to oAW, it was just the „bare bones“ means of creating and applying a profile to a UML2 model.

There are two things we have to change: The workflow (specifically, the configuration of the generator component) needs to know about the profile, and the template needs to generate different code if a class has the *test* stereotype applied. Let's look at the second aspect first. Here's the modified template (in *RootWithProfile.xpt*):

```

«DEFINE Root FOR uml::Model»
    «EXPAND PackageRoot FOREACH (List[uml::Package])ownedElement»
«ENDDEFINE»

«DEFINE PackageRoot FOR uml::Package»
    «EXPAND ClassRoot FOREACH (List[uml::Class])ownedType»
«ENDDEFINE»

«DEFINE ClassRoot FOR uml::Class»
    «FILE name+".java"»
        public class «name» {}
    «ENDFILE»
«ENDDEFINE»

«DEFINE ClassRoot FOR test::test»
    «FILE name+".java"»

```

```
public class <name> {} // stereotyped
<<ENDFILE>>
<<ENDDDEFINE>>
```

As you can see, the stereotype acts just like a type, and even the polymorphic dispatch between the base type (`uml::Class`) and the stereotype works.

Adapting the workflow file is also straight forward (*workflowWithProfile.oaw*), here's the modified generator component:

```
<component id="generator" class="oaw.xpand2.Generator"
  skipOnError="true">
  <metaModel class="oaw.type.emf.EmfMetaModel"
    metaModelPackage="org.eclipse.emf.ecore.EcorePackage"/>
  <metaModel class="oaw.uml2.UML2MetaModel"/>
  <metaModel id="profile"
    class="oaw.uml2.profile.ProfileMetaModel">
    <profile value="test.profile.uml"/>
  </metaModel>
  <expand
    value="templates::RootWithProfile::Root FOR model"/>
  <outlet path="src-gen">
    <postprocessor class="oaw.xpand2.output.JavaBeautifier"/>
  </outlet>
</component>
```

The only thing we have to do is add a new metamodel that references the profile we just created.

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.