

OpenArchitectureWare 4.1 Using AOP in templates

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

PRIMARY SPONSORS



Informatik AG



Enterprise
Software Solutions



SECONDARY SPONSOR

Table of Contents

INTRODUCTION.....	3
THE PROBLEM.....	3
EXAMPLE.....	3
TEMPLATES.....	3
WORKFLOW FILE.....	4
RUNNING THE NEW GENERATOR.....	5
MORE AO.....	6

Introduction

This example shows how to use aspect oriented programming techniques in Xpand templates. It is applicable to EMF based and Classic systems. However, we explain the idea based on the *emfExample* – hence you should read that before.

The problem

There are many circumstances when template-AOP is useful. Here are two examples:

Scenario 1: Assume you have a nice generator that generates certain artefacts. The generator (or cartridge) might be a third-party product, delivered in a single JAR file. Still you might want to adapt certain aspects of the generation process – *without modifying the original generator*.

Scenario 2: You are building a family of generators that can generate variations of the generate code, e.g. implementations for different embedded platforms. In such a scenario, you need to be able to express those differences (variabilities) sensibly without creating a non-understandable chaos of *if* statements in the templates.

Example

To illustrate the idea of extending a generator without „touching“ it, let's create a new project called *oaw4.demo.emf.datamodel.generator-aop*. The idea is that it will „extend“ the original *oaw4.demo.emf.datamodel.generator* project introduced in the *emfExample*. So this new project needs to have a project dependency to the former one.

Templates

An AO system always needs to define a joinpoint model; this is, you have to define, at which locations of a (template) program you can add additional (template) code. In Xpand, the joinpoints are simply templates (i.e. *DEFINE .. ENDDEFINE*) blocks. An „aspect template“ can be declared *AROUND* previously existing templates.

If you take a look at the *oaw4.demo.emf.datamodel.generator* project's source folder, you can find the *Root.xpt* template file. Inside, you can find a template called *Impl* that generates the implementation of the Java Bean.

```
<<DEFINE Entity FOR data::Entity>>
  <<FILE baseClassName() >>
    // generated at <<timestamp()>>
    public abstract class <<baseClassName()>> {
      <<EXPAND Impl>>
    }
  <<ENDFILE>>
```

```

«ENDEFFINE»

«DEFINE Impl FOR data::Entity»
    «EXPAND GettersAndSetters»
«ENDEFFINE»

«DEFINE Impl FOR data::PersistentEntity»
    «EXPAND GettersAndSetters»
    public void save() {

    }
«ENDEFFINE»

```

What we now want to do is as follows: Whenever the *Impl* template is executed, we want to run an additional template that generates additional code (for example, some kind of meta information for frameworks ... the specific code is not important for the example here).

So, in our new project, we define the following template file:

```

«AROUND Impl FOR data::Entity»
    «FOREACH attribute AS a»
        public static final AttrInfo «a.name»Info = new AttrInfo(
            "«a.name»", «a.type».class );
    «ENDFOREACH»
    «targetDef.proceed()»
«ENDAROUND»

```

So, this new template „wraps around“ the exiting template called *Impl*. It first generates additional code and then forwards the execution to the original template using *targetDef.proceed()*. So, in effect, this is a *BEFORE* advice. Moving the *proceed* statement to the beginning makes it an *AFTER* advice, ommitting it makes it an override.

Workflow File

Let's take a look at the workflow file to run this generator.

```

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

    <cartridge file="workflow.oaw"/>

    <component adviceTarget="generator"
        id="reflectionAdvice"
        class="oaw.xpand2.GeneratorAdvice">
        <advices value="templates::Advices"/>
    </component>
</workflow>

```

Mainly what we do here is to call the original workflow file. It is available from the classpath. After this cartridge call, we define an additional workflow component, a so-called advice component. It specifies *generator* as it's *adviceTarget*. That means that all the properties we define inside this advice component will instead be added to the component referenced by name in the *adviceTarget*, in our case the generator. So, in effect, we add the `<advices value="templates::Advices"/>` to the original generator component (without invasively modifying its own definition! This contributes the advice templates to the generator.

Running the new generator

Running the generator produces the following code:

```
public abstract class PersonImplBase {
    public static final AttrInfo
        nameInfo = new AttrInfo("name", String.class);
    public static final AttrInfo
        name2Info = new AttrInfo("name2", String.class);
    private String name;
    private String name2;

    public void setName(String value) {
        this.name = value;
    }

    public String getName() {
        return this.name;
    }

    public void setName2(String value) {
        this.name2 = value;
    }

    public String getName2() {
        return this.name2;
    }
}
```

More AO

In general, the syntax for the *AROUND* construct is as follows:

```
<<AROUND fullyQualifiedDefinitionNameWithWildcards
    (Paramlist (*)?) FOR TypeName>>
do Stuff
<<ENDAROUND>>
```

Here are some examples:

```
<<AROUND *(*) FOR Object>>
```

matches all templates

```
<<AROUND *define(*) FOR Object>>
```

matches all templates with *define* at the end of it's name and any number of parameters

```
<<AROUND org::oaw::* FOR Entity>>
```

matches all templates with namespace *org::oaw::* that do not have any parameters and whose type is *Entity* or a subclass

```
<<AROUND *(String s) FOR Object>>
```

matches all templates that have exactly one *String* parameter

```
<<AROUND *(String s,*) FOR Object>>
```

matches all templates that have at least one *String* parameter hat

```
<<AROUND my::Template::definition(String s) FOR Entity>>
```

matches exactly this single definition

Inside an *AROUND*, there's the variable *targetDef*, which has the type *xpand2::Definition*. On this variable you can call *proceed*, and also query a number of other things:

```
<<AROUND my::Template::definition(String s) FOR String>>  
  log('invoking '+<<targetDef.name>>+' with '+this)  
  <<targetDef.proceed()>>  
<<ENDAROUND>>
```

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.