MDA using the Generator Framework

Peter Wittmann, peter@wittmannclan.com, www.wittmannclan.com

In here I try to sum up everything you need to use the open ArchitectureWare (oAW) provided by b+m Informatik AG. It's not really hands-on, but I will try to come up with something like that soon.

Introduction

What's all the fuzz about? Why not get the latest zip from SourceForge extracting it and start using it? Well, it's not that easy. That's not because it's so extremely complicated to use, but the guys that developed this framework are using some specific names that charactarize their workflow and artifacts. And they still do not provide a complete reference and a useful documentation. There are documents in the old zip file, but there are also some in the latest release.

I have to admit, that right here I wanted to summarize all of my knowledge. I think I will be sure to fail as the whole subject should cover more than just the use of the generator. But maybe this is some help to you, anyway.

Conceptual

the Generator Framework is considered to be one of the pragmatic solutions in this market. So, what are the main differences between the way OMG sees MDA and the way the OpenGenerator uses the ideas of MDA? Well, it's not that easy to explain without going into detail (which we will do later in this article). But from a cnceptional point of view the part missing is the PIM. b+m informatik AG actually released a couple of papers boundled with their former releases of the framework where they stated that there is even supposed to be a certain process management in order to work correctly with the OpenGenerator Framework. Frankly, I think this idea is not very realistic. Small businesses might be able to change some of their processes to match the given criterias, but most businesses already have some kind of process and they're not willing to change that because of one specific tool being used. On the other hand b+m calls for teams that have architects whose knowledge in Java is vital to the whole process and developers who should also know a specific scripting language (called XPAND) to use their tool. I don't think that this is too much of a burden but in conjunction with professional knowledge in both UML and the OpenGenerator obtaining this specific knowledge is quite a challenge for both the architect and the developer.

As mentioned before, there are no PIMs in the Generator Framework. The framework is expecting UML (ot better say the XMI as it reads that) to be enriched with some platform specific attributes (such as stereotypes or namespaces). This is probably the greatest (conceptual) drawback aside from all the other technical obstacles to take. It doesn't actually represent the original idea of the MDA as it was suggested to be like by the OMG. We will get into more detail on that right after we take a look at the whole process and how we're supposed to use the generator.

The generator was called ArchitectureWare before its name was changed to OpenGenerator Framework. The generator is written in Java and reads XMI. But here's another problem you might run into: as the manufacturers of most modern UML tools don't seem to be able to comply with the standard, only a number of tools are supported (Rose and Poseidon, for

instance). But the developers promise to work on that and write more filters.

Let's try to go through an imaginary project and the appropriate steps. This is making the whole thing a lot easier to understand. Let's assume we want to create some web-application using the Struts framework.

First of all the architect is supposed to model the system in a CASE tool, such as Poseidon by Gentleware. He models the system specifications and what an application of this kind (web-application written with Struts) needs to have and what it shouldn't have. Those constraints are not modelled using the OCL but are also written in Java. There are some classes provided with the generator that implement all of the specifications of a class diagram and almost all specifications of activity diagrams. The thing is that he has to write some Java classes that match his idea of how the model should look.

But how is that achieved? How's the model matched to the resulting model of the application? The framework usesthe namespace attribute in activity diagrams and the stereotype attribute in class diagrams to match objects. The stereotype (or namespaces) are mapped to classes of the same package/name. The architect is able to write constraints by coding methods that have certain names. In those methods he has access to all of the classes' attributes, method names, connections/references, etc. All he has to do is write some code (in Java) that goes through these items and checks if something's not the way he suspected it to be.

Let's take a look at an example: the architect doesn't want the classes of type XController to have outgoing references. He writes a Java class XController.java (so that the name matches the stereotype in the UML model) and writes a method and calls it checkConstraint. In this method the program iterates through all the references and checks if the source of each reference is of an Object of type XController. If that is the case an exception will be thrown. Looking at the model this means that there was something wrong in the model as there are outgoing references which weren't supposed to be.

The Java model the architect writes, is reffered to as the meta model. This model is supposed to represent all of the Struts applications of a certain type. It represents constraints that developers can not override (without throwing an error of course) and that all of the developers' models have to comply with. Those constraints can differ from application to application just the way the requirements are being changed. So the idea is that after some time numerous meta models exist that describe different applications but can be reused to create other applications based on the same architecture.

The developer is supposed to write the real application using the architect's meta model (consisting of the java classes). Depending on the way the whole process is organized he takes the UML models from the architect that do not represent the requirements (all that is implemented in Java classes that form the meta model) but how to model the application. The developers extends these diagrams, writes new cases, classes, activity diagrams. He can check his design by running them through the generator together with the meta model.

But it's also the developers task to create the appropriate class files. He has to write some scripts in a certain language that is specific for this generator. Compared to AndroMDA the language of those scripts is not standardized (AndroMDA uses the Velocity Scripting Language to transform the XMI to code). Here, the XMI model is transformed into a tree of objects (i.e. instantiated classes) in the system memory. After the model is validated against the meta model (by invoking each of the check-methods on the Java objects), the scripts then walks through that tree. Everytime it finds a matching object it gets into action by accessing all of the objects parameters (such as method signatures, attribute definitions, etc.). But it can also access other methods, methods that were created in the meta model! As mentioned before: the architect has to write the checkConstraints methods in the meta classes, but he can also write other methods. Those can be accessed by the XPAND scripts. These scripts can then create platform specific code based on the definitions in those

methods.

The whole framework is not supposed to work without putting some business logic into the final code. That means, that there are still parts that need to be programmed by hand. Such parts can be explicitly defined in the XPAND-scripts. The sections that are supposed to be written by hand are marked. Once a programmer writes some code there and runs the generator again, this code will be preserved but the surrounding code might change. This is one key element of all of the different mda tools available today: code preservation is available.

Metamodels

Metamodels are used in two ways: validate the specifically created application model and provide ways to access certain artifacts of the model.

In the first case the application model will be instantiated using the meta model. Then all the Check-methods will be called - on classes of the meta model now with the specific attributes, names, etc. of the corresponding class in the application model. This mapping (which part of the model has to be instantiated using which class in the meta model) is defined in the file MetaMappings.xml. This file may look like this:

```
<?xml version="1.0"?>
<!DOCTYPE MetaMap PUBLIC
"-//b+m Informatik AG//DTD b+m Generator
FrameWork MetaMappings 2.1//EN"
"http://www.bmiag.com/dtds/metamappings_2_1.dtd">
<MetaMap>
  <Mapping>
    <Map>ViewTableModell>/Map>
      <To>de.bmiag.bmawf.ViewTableModell</To>
    </Mapping>
  </MetaMap>
</MetaMap>
```

metamappings.xml

Right here there stereotype ViewTableModell is mapped to the metaclass de.bmiag.bmawf.ViewTableModell. It will be instantiated with this class.

Which checks are going to be performed (which means: which Check-methods will be called) can be configured in the Check.tpl. If we want to run the CheckConstraint method on all classes of type Presentation in the meta model, we would have to put the following code in the Check.tpl:

```
«DEFINE Check FOR Presentation»
«CheckConstraint»
«ENDDEFINE»
```

Check.tpl

So all methods named CheckConstraints of all the objects of type Presentation will be called. All of these methods have to public void and must throw and IllegalDesignException in case

the validation goes wrong.

So then, how are we going to check to see if our models are implemented correctly? We just call the check methods (e.g. CheckConstraints) by defining them in the Check.tpl as mentioned before. In here we have several ways of finding out about the specific classes. Let's assume we do not want any class of one certain type to contain an association to any other class of the same type (e.g. no class of type Action should contain an association to another class of type Action).

Transformation

The Root.tpl is responsible for defining which EXPAND templates will be used on the instantiated metamodel. The Root.tpl should look like this:

```
«DEFINE Root FOR UIAction»
«EXPAND Action::file FOR This»
«ENDDEFINE»
```

Root.tpl

What this means is: for each object of type UIAction take the file Action.tpl and parse all the data that's in the definition called file.

Pros and Cons

As you might have guessed: the initial threshold to get this (undoubtedly) mighty tool up and running is extremely high. And there are other problems once you get this far: the documentation is hard to find and rather outdated, the source code in the cvs directory is not compatible with the other packages (like genfwutil), scarcely documented and there isn't a wide-range support for UML tools. This means that it's up to you to write some stuff to extend the original generator and go through the source code in order to find out about what changed compared to the documentation. This is extremely painful when it comes to problems in the genfwutil packages as there's almost no documentation. If there was at least the usual JavaDoc.

But going through the source code is also very helpful once it comes to extending the generator. Especially the idea of changing the input from XMI to any kind (even normal text) is a nice idea. And once you gain enough experience in the whole process of generating code you start modeling more abstract diagrams. Those abstract models are more likely to be reused in future projects. That's because you start modeling more provident concepts, e.g. concepts for persistence layers and those are very common in todays projects. On the other hand it's very easy to change the generated code. Maybe you have to switch from EJB to JDO one day. The steps to do that here are easy: just change the appropriate template files and run the generator again.

Summary

Once you get this up and running you got a fine tool. Due to the problems mentioned before I wouldn't want to call it a handy tool, but still a tool that helps you get as close as possible to the initial idea of MDA. And the closer we get to that idea the more efficient software is about to become. Some concepts might be transferrable to other systems and with MDA it's easy to

take that concept and generate different programming languages from it. The whole development process is becoming more and more agile, which means that changes and future development becomes easier. Generated code will never be as efficient as handwritten code, but then again generated code is more likely to be error-free.

Me, I'm up for trying this tool in real-life commercial projects. And even if it's going to be a drawback I still believe that this way of developing software is becoming more and more popular. With full-blown CRM applications like openCRX that were developed using MDA tools, MDA proves to be quite useful and not just a theoretical piece of paper.

Links

- open ArchitectureWare: http://architecturware.sourceforge.net
- openCRX: http://opencrx.sourceforge.net
- Slashdot article: http://developers.slashdot.org/article.pl?sid=02/11/13/1319226