MDA using AndroMDA

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

After taking a closer look at open ArchitectureWare, I was curiousXXX about AndroMDA since it states to be one of the most widely used MDA tools out there. In here I would like to give an overview of this tool looking at version 3.0M2.

Introduction

Ever since UML becomes more and more integrated in IDEs, the step towards generating code from models is becoming more and more popular. But MDA means more than just generating all of the classes from a UML class diagram.

As one of the very early projects in this field of MDA is AndroMDA (pronounced andromeda). Following is a short summary of how it works and how to use as well as my opinion on the whole tool.

Cartridges

Cartridges are the building bricks of the AndroMDA framework. The cartridges consist of one or more files combined in a Java archive (jar). There has to be at least one cartridge description file (which is written in XML) and one or more templates. The framework comes with some basic cartridges for BPM4Struts, EJB, Hibernate, Java, Struts and others.

Those cartridges are bundled in archives and placed in <andromda-home>/andromda/jars.

Templates

The cartridges includes several template files. Those are script operate on the Velocity sciprting language. The Velocity project is one of the many projects from the Apache Group. One of these scripts might look as the following excerpt from the StrutsAction.vsl from the Struts cartridge:

```
#set ($packagename = $class.packageName)
package $packagename;
#foreach ( $associationEnd in $class.associationEnds )
#set ($target = $associationEnd.target)
#if ($target.navigable)
    #set ($class2 = $target.type)
    #if ($class2)
        #set ($class2packagename = ${class2.packageName})
        #if ($class2packagename != $packagename)
import ${class2.fullyQualifiedName};
        #end
    #end
#end
#end

/**
$class.getDocumentation(" * ");
 *
 *
#foreach ( $tgv in $class.taggedValues )
#if ($tgv.tag != "documentation")
#if ($tgv.tag == "---")
    #set ($tag = "")
#else
    #set ($tag = $tgv.tag)
#end
 * $tag    $tgv.value
#end
#end
 *
 */
public class ${class.name} extends Action
{
]
```

StrutsAction.vsl

Without going in too many details: writing and dealing with these scripts is rather easy as most of the properties of the model elements (names of the methods, properties, associations, namespaces and others) are already available through the framework. There are methods to access these model elements.

Mappings

The jar also includes an andromda-cartridge.xml which is also referred to as the cartridge descriptor as it maps model elements to scipt files. Let's again have a look at an excerpt from the sample andromda-cartridge.xml from the Stuts cartridge:

```
<cartridge name="struts">
    ...
    <template
      path="templates/StrutsAction.vsl"
      outputPattern="{0}/{1}.java"
      outlet="actions"
      overwrite="false">
        <modelElements variable="class">
            <modelElement stereotype="WebAction"/>
        </modelElements>
    </template>
    ...
</cartridge>
```

andromda-cartridge.xml

Pros and Cons

Getting AndroMDA to work is quite easy: a good documentation and an easy to use framework makes it easy to understand what is going on. Extending the framework is easy considering the fact that you can write cartridges using the easy to use Velocity scripting language. The output is possible to be in any ASCII-readable language.

But is this really what the OMG proposed to be model driven architecture? I don't think so. After all this is just a little bit more than what is already possible with full-blown IDEs like Together. The idea to move from models that represent code to models that represent overall strategies that can be reused is not represented here. The models are very close to the actual code. And there is no way to access other models (like statechart or callaboration). That makes it quite difficult to describe the application completely. And it's also not possible to validate the models to the cartridges to see if any concrete design mistakes were done and if the model still complies with the desired application architecure.

Summary

I wouldn't consider AndroMDA to be real MDA (even if it might suggest that by chosing that name). AndroMDA is not really dependent on any certain IDEs and you can model applications close to the desired source code. It's easy to use, easy to understand and you will get some nice experiences out of it. But that's about it. The included and developed cartridges are not helping to develop some portable strategies or ideas.

Links

- a comparison of MDA tools: mda_compare.pdf
- AndroMDA: http://www.andromda.org
- Velocity: http://velocity.apache.org
- openCRX: http://opencrx.sourceforge.net
- Slashdot article: http://developers.slashdot.org/article.pl?sid=02/11/13/1319226