Model Driven Architecture

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

This is a short (mostly conceptual) introduction to MDA and comparing the reference to the b+m informatik AG OpenArchitecureWare Framework. There are more documents on MDA and other tools in this directory. Just take a look at the links.

Introduction

If you've ever come across MDA you hopefully know that the goal of that specification is to describe a software development process starting from a higher level of abstraction using UML and some tools to convert the abstract definitions of software in UML diagrams into (human readable) source code.

Before going into more detail and focussing on different tools supporting MDA like the OpenGenerator Framework I want to outline the original MDA idea as given by the OMG and then compare it to a tool.

OMG's MDA

The initial idea was to start modelling systems from a more conceptual approach. The models that describe application behaviour from a very abstract point of view are called platform independent models (PIM) as they can be used to model many different systems and thus could be reused. The models' job is to describe the behaviour not caring about how exactly this is achieved on a specific machine using a specific programming language. Therefore these models are often visualized using diagrams that represent more conceptual aspects. Diagrams of this kind include state charts, use case diagrams and activity charts, but it's possible to also use class diagrams. It's actually just a matter of how specific those diagrams should become for a certain machine and what kind of concepts should or must be represented.

The next step is to do some fine grain modelling and extend the existing model by specifying the system for a certain platform. Here mappings from class diagrams to certain code fragments as well as direct mappings of stereotypes or other UML specific features are being performed. Those models are called platform specific models (PSM) and they are created by transforming (one or many) PIMs through a specific mechanism to (one or many) PSMs.

The transformation from PIMs to PSMs can (but doesn't necessarly has to) be achieved through the standardized exchange format for UML models, XMI. XMI is a specification for UML to XML mapping. This XML can be read by different tools (such as the OpenGenerator Framework) to create code or code stubs and fragments in a specific language. But once again OMGs specification doesn't state anything about how the transformation is supposed to be achieved. Therefore many different implementations exist. One of them is OptimalJ. CompuWare's OptimalJ uses its very own program for both modelling activities and transformations and doesn't care about XMI or any XML transformation. In the end it's pretty much the same after both OptimalJ and OAW created the corresponding source files. These files can now be filled with business logic.

Once a new feature is going to be added, MDA assumes that there's the need for a new process and thus for a change in one of the diagrams that represent the results of the

requirements elicitation process. Once the feature was added to the appropriate diagrams the generation of PIM and code can be started again. Depending on the tool, hand written code (e.g. in method stubs) is kept and deprecated methods (that have become deprecated due to a change in the PSM) are marked specifically. So this process is a top down view of development in which requirements can not be stated from a level below the requirements elicitation process.

This whole procedure has also some obvious drawbacks. Although MDA is considered to be one of the most agile and fastest ways of developing software, due to the features outlined before, it's also based on one paradigmXXX. MDA can be used in almost all software lifecycles and offers, at first sight, all of the key features necessary for changing requirements rapidly and still being able to develop on a rather stable basis. Even if MDA is used in processes of rapidly changing requirements, MDA can support more strictly organized software development processes much better. First of all MDA encourages programmers to develop a stable and robust basis they won't have to change into something completely different (this is something that could happen in XP all the time). Secondly, MDA provides all of the tools necessary to work in an environment where analysts should not only define programs through a process of requirements engineering but also through definition and application of reusable patterns. Therefore results of any of the preliminary stages of programming emphasize to reuse already existing pieces of abstraction in the PIMs. This is the main idea behind MDA as constituted by the OMG: ZITAT!!!reuse of patternsZITATXXX That of course doesn't mean MDA is absolutely unsuitable in XP environments but it is prejudicedXXX to be applied in more stable software development lifecycles where changes in the requirements are not happening at the coding base but in the elicitation process of the whole software development process. But MDA isn't just another specification for a more complex generator. Through its thouroughlyXXX definition of processes it can also be seen as a whole software development process. There's no actual need to use MDA in any non-iterativeXXX development process. Even if it might sound charming to be able to commit changes in the software by changing requirements and still being able to keep a lot of the code, this has never been the purpose of MDA. And if MDA is just used to straighten out processes, keeping track of changes and controlling the software development there's probably something else going completely wrong. These seems more like a simple generative development process which tries to imitate some of the features MDA offers. But MDA is by all means neither a tool to control any of the other layers nor does it act as a saviourXXX to faults in the development process.

OMG vs Reality

OMG's impact on the software development community has always been great. One of the most famous specifications which still had a broad impact on many fields of software development is UML. By today UML is in version 2.0 and has become so universal that it's now possible to perform business process modelling in UML. Since the introduction of the MDA specification numerous tools, that proclaim to support this approach, have emerged. OMG lists those that approve the idea in reference to the specification on its site as committed products (see link). But even if all of these tools are approved by the OMG there are still quite some differences among them and the way they handle the specification. Let's take a closer look at one of the more famous tools, OpenArchitectureWare (OAW), and how well it plays along with the ideas of MDA as the OMG states them in their specification.

It's not that easy to explain the differences without going into too much detail of OpenArchitectureWare (which I will do later in another article). From a cnceptional point of view the part OAW lacks the PIM/PSM concept. b+m informatik AG, the creators of the tool, actually released a couple of papers in former versions of the framework. In those papers they stated that there is supposed to be a certain process management in order to work

correctly with OAW. Frankly, I think the idea of having to use a certain process to be able to use OAW, is not very realistic. Small businesses might be able to change some of their development processes to match the given criterias, but most businesses (especially larger ones) already have some kind of process and they're probably not willing to change that because of one specific tool in use. And considering my experiences with this tool I'm convinced that it's possible to to use OAW efficiently without having to change any processes at all.

On the other hand b+m Informatik AG calls for teams that have architects, whose knowledge in Java is vital to the whole process, and developers both who should also know an OAW-specific scripting language (called XPAND). That's not really much but in conjunction with professional knowledge in both UML and the specific behaviour of the tool itself this is quite a challenge for both the architects and the developers.

But the main drawback still remains: the generator will not exactly use PIMs to formulateXXX behaviour. Due to the lack of PSMs OAW expects the UML models (and therefore the XMI as well) to be enriched with some platform specific attributes (such as stereotypes and/or namespaces). This is probably the greatest (conceptual) drawback aside from other technical obstacles. Missing this aspect OAW doesn't actually represent the original idea of the MDA in its full detail as it was suggested to be by the OMG. That's one of the reasons why b+m informatik AG actually calls it a 'pragmatic approach' (XXXlink einfuegen mit referenz). If pragmatic were to mean less this would fit. This will become much more evidentXXX once the whole process and the way the generator works has become clear.

Open Architecture Ware

The generator is written completely in Java and reads XMI to do the transformations. But even if that might sound too good to be true (platform independence is guaranteed through universal file formats and a system independent programming language), there's another problem: the manufacturers of today's UML modelling tools don't seem to be able to comply with the XMI standard. That's why only a number of tools are supported (for instance Rose and Poseidon). For each new tool there has to be an adapter that translates the tool's specific proprietary extensions in the XMI to a format which complies with the specification and which the generator understands. The developers of OAW promise to work on further tool support though.

Let's try to go through an imaginary project and the appropriate steps to take. This is probably going to make the tool a lot easier to understand.

Architect

First of all the architect is supposed to model the system in a CASE tool, such as Poseidon by Gentleware. An architect models the system specifications and what an application of this kind (e.g. a web-application written with Struts) needs to have and what it shouldn't have considering modelling details. He can e.g. specify that there shoudln't be any helper classes that directly interact with the JSPs. Those constraints are not modelled using the OCL but instead are 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. It's the architect's responsibility to write some Java classes that match his idea of how the model should look.

He does that by using the namespace attribute in activity diagrams and the stereotype attribute in class diagrams. The stereotype (or namespaces) are mapped to classes of the same package/name. Creating constraints by writing methods that have certain names is as simple as it sounds. In those methods access to all of the classes' attributes, method names,

connections/references, etc. is possible. All that is left to do is write some code (in Java) that goes through these items and checks if something's not the way as suspected.

The Java model, written by the architect, is reffered to as the meta model. This model is supposed to represent all of the applications of a certain type. It represents constraints that developers can not override (without throwing an exception 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 change. 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.

Developer

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 and forms the meta model) but some of the architects diagrams on how to model the application. He extends these (by overwriting/changing the diagrams) and writes new cases. He can check his design by running it through the generator together with the meta model.

But it's also the developers task to create the appropriate class files that in the end represent his application and whose stubs he has to fill. He has to write some scripts in a specific language for this generator. Compared to AndroMDA the language of those scripts is not standardized (AndroMDA uses the VSL to transform the XMI to code). Here, the XMI model is transformed into a tree of objects (i.e. instantiated classes) in the memory. After the model is validated against the meta model (by invoking each of the checkXYZ-methods on the Java objects), the scripts then go through that tree. Everytime they find a matching object they get into action by accessing all of the objects parameters (such as method signatures, attribute definitions, etc.). But they can also access other methods. Methods that were created in the meta model! As I said 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.

Code

The whole framework is not supposed to work without putting some business logic into the final code. That means, that there are still parts which need to be programmed by hand (and human brain). Such parts can be defined in the XPAND-scripts. The sections that are supposed to be written by hand are marked specifically. Once a programmer writes some code in there and runs the generator again, this code is preserved but the surrounding code might be changed. This mechanism allows for further changing the meta-model and performing some iterative development.

Review

The concept of the whole framework is nice. The idea to create families of applications is going to support the reuse of existing patterns. Even having the option to test the models against predefined rules is something architects will find helpful for sure. Writing those rules in a far more complex environment using Java opens a lot of opportunities and is superior to OCL.

But that's also one of the worst things about this tool: the mix of PSM and PIM. What has been noted as one of the primary tasks of the OMG specification has not found its way into OAW.

Summary

The generator works fine and b+m informatik AG even states that they already used it in a number of real projects. From my personal point of view the whole generator's concept is nice but doesn't quite match the original idea behind MDA. It still is a much more elegant approach than AndroMDA and fascinating by its concept. But if it really helps developing applications faster and decrease the time to market is something I doubt. Even if you don't change your way of developing applications to the way b+m informatik AG suggests, it still is requiring that the architect has advanced knowledge of the Java language and the designer is good in writing scripts using XPAND.

The other thing is that there are some major issues concerning the way the tool operates. Those originate from manufacturers of UML CASE tools (as they don't seem to be able to apply the given standard for exporting UML to XML) as well as the fact that the only models you can access at the moment are activity diagrams, state charts and class diagrams.

Links

- OpenGenerator Framework: http://architecturware.sourceforge.net
- Markus Voelter's site: http://www.voelter.de
- MDA: http://www.omg.org/mda
- XMI: http://www.omg.org/xmi
- Gentleware: http://www.gentleware.com
- MDA committed products: http://www.omg.org/mda/committed-products.htm