Comparison of MDA tools

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

The following is a short comparison which might be suitable for evaluation purposes to find out about the differences in approach, features and concepts which are to be considered as the implementation of OMG's MDA specification. It might also serve as a reference for evaluation purposes for other evaluations of these kinds of tools.

Introduction

OMG's rather new specification MDA [OMG01] has become very popular during the last couple of years. While OMG's reference web page of MDA committed products and companies [OMG03] reveals numerous entries not all of them are really following the standard step by step. A lot of them follow totally different ways of providing a way to develop applications using MDA.

This article is going to compare just three of the many available products available today. Focusing on the concepts behind the tools as well as how to use them.

The process of modeling and getting to use MDA tools is filled with concepts, abstractions and buzzwords that users with no experience in this field will find very hard to deal with and can get confused easily. That combined with the ability to be forced to think abstract and in models is nothing that can be put into hard numbers. Therefore the assumption that only experienced developers should start develop applications using MDA seems quite reasonable. Therefore this article is targeted at developers who have knowledge in modeling activities. Even though a short introduction to the standard is given in the next chapter, it's assumed that the reader is familiar with the basic concepts of MDA and the basic programming techniques.

MDA

First, let me introduce you to the idea and the concept behind MDA. Although MDA is quite young, it's a very popular specification. That's because MDA came up with an idea that has become nearly standard in many different fields of software development: generation of code. XDoclet [XDoclet] is very well known by now.

OMG's specification is quite abstract: it describes the process of modelling a PIM that could be transformed by a PSM to virtually any platform. While the PIM is supposed to be abstract and platform independent (hence the name PIM which stands for platform independent model), the PSM (which stands for platform specific model) is the specific model for a platform. According to the OMG this mapping from PIM to PSM is done with the idea to reuse concepts manifested in PIMs throughout different projects. The difference to CASE tools is that code can be generated to virtually any platform. Instead of describing a more abstract modelling language for a certain programming language, MDA tries to be on top of that by staying product and platform independet. The idea behind MDA is that while programming languages and platforms change throughout time, there are concepts that can be reused. Preserving those concepts and being able to transform them to different platforms would be a major step in software development. It would not just preserve established concepts and ideas but also reduce time to develop applications. Through a process of transforming

abstract models to very concreteXXX platforms a more elegant and a clean implementation can be realized. OMG's vision goes even further: through model transformations from PIM to different PSMs the activity to code should be obsolete. But that's only the vision and by now tools are fighting to go through the first steps of this specification.

JMI

JMI [JCP01] (also known as JSR 040) is not that young anymore. Its final specification was published in June 2002 but it still hasn't become very popular. JMI defines a Java mapping for the MOF. That mapping is defined through four different kinds of metaobjects. Those metaobjects are supposed to map all of the UML model elements to Java classes and interfaces. They are referred to as metaobjects as they provide a layer above the actual model. If mapped to the MOF levels those metaobjects would be at MOF level 1. The metaobjects are: instance objects, class proxy objects, association objects and package objects.

Instance objects hold the state of the instance-scoped attributes. Therefore it contains simple set, get, add and remove operations. Instance objects can be created by calling the corresponding create operations in the class proxy objects (which are factory operations).

Class proxy objects serve their instance objects by providing approriet factory methods to for producing instance objects within the package extent. But they also serve as the specialized container for their instance objects. Class proxy objects might also hold the state of any attribute that is of the scope of the class and also provide operations corresponding to classifier-scoped operations.

An association object is the object that holds one or more links. It's static and its interface provides operations on these links. Those operations include: querying the links, adding, removing, modifying and returning all the links. The link itself is nothing but an instance of an association object that represents a physical link between two instance of the classes connected by the association object.

Package objects are defined to be quite primitive. They're nothing more than a directory of operations that give access to metaobjects defined by the metamodel. The outermost package is also referred to as the root of the object-centric model of the metadata. All other objects are contained within that root and are created by using the appropriet accessors provided by the MOF.

Tools

The following tools were selected: the open ArchitectureWare [oAW01], the popular AndroMDA [Andr01] and the rather new openMDX [oMDX01]. They are self-proclaimed to follow OMG's MDA specification. And even though they are all open source and can be xtended and changed in many different ways, the focus will be on their implemented characteristics and not on what could be virtually implemented with great efforts. If there are open interfaces to extend the code base through plug-in mechanisms or anything like that, that's a considerable feature. Anything else is beyond the scope of this article as it would come close to develop a new application on the basis of an existing one rather than extending the existing ones through forseen mechanisms. Even though the tools mentioned above are inexpensive in their initial costs, the TCO is nothing that will be paid attention to here. It's hard to determine those numbers anyway as they're a combination of many aspects.

Approach

The above products implement this specification in different ways in both their modelling activities (both PIM and PSM) and their way of mapping the models to different platforms.

Starting off with the modeling activity, the specification is based on UML [OMG02] but this is not mandatory and thus the model that serves as an input for the implementation could be virtually anything. open ArchitectureWare is the only one of those three that takes this into consideration and offers a way to take different input formats through an interface that can be extended through a plugin-like mechanism. This way it's even possible to take all kinds of formats (even normal text) as input. AndroMDA and openMDX are less flexible. They both can convert XMI [OMG04]. XMI is the OMG's standardization for an interchange format for UML. While XMI is a hard standard there are still differences in the XMI among different modelling tools. Even the most popular tools (like MagicDraw UML [NMI01], Poseidon for UML [Gen01], Rational Rose [Rat01] and Together [Bor01]) still don't implement the reference accurately without any dialects. Therefore the MDA tools offer input for only some of the tools on the market. But they all provide an open interface to import future XMI dialects. Especially AndroMDA is very fixed to XMI. No other format can be used as input. openMDX is also bound to the input of XMI because of its MOF to Java mapping, specified in JMI [JCP01], which is fixed to UML and therefore to XMI (as the intended product independent exchange format).

As mentioned before: one of the many false assumptions is that MDA is pretty much like any other CASE tool. This is absolutely wrong as MDA does not want to provide just another way of writing software in terms of drawing nice pictures instead of writing code. MDA is all about concepts and the reuse of conceptual artifacts. Looking at the modeling activities (and let's just focus on UML and XMI as all three tools provide a way to deal with this format), there are huge differences among those tools in how they treat the exported models.

openMDX follows a rather straight class model transformation based on the MOF to JMI [JCP01] mapping and AndorMDA's just considering the class diagrams for transformations. open ArchitectureWare seems like the most extensible tool among those three as it doesn't limit the model and is able to take the major three model diagrams into account: class model, collaboration diagram and state chart. You can of course model your application starting from use case diagrams to collaboration, state charts and deployment diagrams. The question is: what is the MDA tool able to take into consideration when it comes to model transformation, how do you have to model the charts and which of these are taken into account when it comes to generation of code and finally: is it really of any use if you use all aspects of modeling diagrams? There's no straight answer to the last question. Considering the fact that all of these diagrams were created for a good reason, not all of them serve to model an application in parts that are relevant for code generation. Let's have a look at use case diagrams: those might serve as presenting how different roles of users interact with different parts of an application or what business cases are supposed to be accessed by different users and how these business cases are dependent on each other. But are they really that necessary and important for the generation of code? If you take a look at class diagrams and associations - can't you use these to model this kind of activity, too? You surely can if you use your very own rules and maybe OCL (which is part of the UML since version 1.1). But was that suggested to serve for this purpose? Once again, there's no best way and especially no explicit restriction. UML offers quite a lot of diagrams and even those that are available with the most recent UML 2.0 specification offer a huge variety to represent application behaviour and design. It's up to the developer to chose from the available diagrams that represent his program best. If you feel the need for use case diagrams you might feel discomforted with the fact that both openMDX and AndroMDA can't take those into consideration. Even if e.g. use case diagrams are used more in aspects of analysis then in design phases of a software lifecycle (see [] for more on software lifecycles) some aspects are much easier to describe using other diagrams than just class diagrams. A MDA tool should offer the comfort to be able to use all of them if possible and not just the ones that

were coming in most handy.

There are different aprroaches in the way the resulting code is being generated. The UML diagrams need to be modelled either close to the resulting code or they can be more abstract. AndroMDA is pretty close to the resulting code and so is openMDX. Both follow the same kind of abstraction: map the class diagrams to one or more classes or edit other files upon the existence of classes. But openMDX uses a far more sophisticated approach as it forces the model to follow certain rules. Those rules are (e.g.) that all of the classes have to extend a certain base class. This might sound as if you aren't capable of defining your very own interfaces. But as the resulting code is spiced with patterns and interfaces it still offers a lot of freedom but forces you to stay within the barriers generated upon the rules manifested in the model. open ArchitectureWare is hard to catch on this point. Considering the fact that the only mapping between the model and the metamodel is through specifying the stereotype, abstraction is possible. On the other hand this mapping is the only possible mapping between those two models (UML models and metamodels). Upon the instantiation of the metamodel based on these mappings, scripts that generate the resulting code are being called.

The scripts that generate the resulting code are done using scripting languages. Those languages can be proprietary or implemented using other, more standardized languages. open ArchitectureWare focuses on Java (for implementing the metamodel and doing the evaluation on that) and uses a self-developed scripting language called EXPAND. This language is rather simple and seems a bit like XSLT [W3C01]. The number of control statements (like if-statements) are limited which keeps the language small and simple. It's easy to read and understand but does only feature the most important control mechanisms. AndroMDA uses the Velocity Template Language (VTL) for producing code. VTL [Apa01] is more complex than EXPAND and offers a variety of control flow and pattern expansion. openMDX uses

Approach			
	open ArchitectureWare	AndroMDA	openMDX
Ways to build code	script files	script files	JMI to MOF
scipt language	EXPAND	Velocity	n/a

Concept

The concept might differ in terms of what a developer has to model, write, program, what is possible and what isn't. Diminishing of code and model.

Concept: open ArchitectureWare

b+m Informatik AG, who wrote open ArchitectureWare, are binding a whole development process to the usage of their tool. They are drawing a hard line between two phases: the development of a family of applications and the development of an application itself. For these two tasks they take two different roles into account: the architect and the developer. Both can be impersonated by the same developer. Besides the fact that their responsibility differs, the initial step is to be taken by the architect. With different steps to take by different roles and persons, i.e. architects and developers, b+m Informatik AG's approach for the development process to a final application might seem thought-out but overdone and a bit unrealistic. In many companies development processes have already been established and will probably not be changed because of one tool. But separating these two roles makes sense when you take a look at the development of the tasks they have to perform.

Before you can begin writing a program you need to decide of which application family the resulting program should be. If you're starting off with open ArchitectureWare there aren't any application families. Thus the initial step is to develop a specific application family. An application family is defined by a meta model which is to be developed by an architect. This meta model can be considered as rules for the development of applications that are of this kind, e.g. all the applications that are supposed to run with the Struts framework have to follow the rules stated in the meta models. The term meta model might come in handy once you're familiar with OMG's MOF. In this context a meta model is a more abstract definition of a number of applications of the same platform. If one or many applications are described by the same meta model these applications are of the same family. And the applications based on the meta model can be considered as the resulting specific instance of such a meta model. Meta models consist of meta model elements and definitions of rules between these (meta) model elements (for a detailed explanation see my article on this tool; see link).

What might sound confusing is actually quite simple and easily understood once you understand how open ArchitectureWare works. From the modelled elements (e.g. in UML) it builds a number of objects in memory. It takes the stereotypes of the model elements to create instances of the Java objects that represent these model elements. That representation is achieved through a mapping file which maps the UML class files to the corresponding Java classes. These objects than have all the approprietXXX names, methods, attributes, associations and so on. Based on these you are able to define rules like: "report an error if there's an association between an object A and an object B" or "traverse the associations named X to see if there's an object of type A that has an attribute called C." Your actions are limited to the class, collaboration and state chart diagrams. But within these you have all the possiblities to perform validations of the model. You might of course go ahead and perform actions on e.g. the use case diagram - but if you want to do that, you will have to extend the tool yourself (so this is actually a little bit beyond the scope of this article).

If the modelled application matches the application family, which means that the meta model could be applied without causing any error messages due to validation errors, scripts are being applied to create code from these rather abstract design objects. The scripts need to be written using a languaged developed by b+m Informatik AG called XPAND. It is a very simple scripting language that consists of conditions, iterations and simple operations (like file creations, stringXXX concatenationsXXXs, etc.). From these scripts the actual files and code is being generated. Both the script files and the meta model are considered to be the full description of an application family. These can be used for other applications that are based on the same meta model. Therefore you don't have to develop a meta model and the script files again if you want to create an application of the same family. You just use the existing meta model and the corresponding script files (and some configuration files) and your application is supposed to be generated correctly.

Let's sum this up. The steps to take are: create a meta model defining an application family, create the appropriet scripts that generate code and develop your program by drawing charts in UML and feed them in as XMI.

The development of meta models has been going through quite some hypeXXX. With people like Markus Völter [Voel01] who wrote numerous articles on this activity, the creation of meta models has drawn quite some attention. Although it still is an activity less sophisticated than it might sound. The reason it got hyped is because you can define rules independently from your code. The layer of abstraction might be something that attracts quite some people but in the end it's nothing but writing rules in another way. If you consider the original approach that consists of PIMs and PSMs this meta model can be considered as a PSM as it already maps the abstract models to (meta) model elements that define rules for their application family which is bound to a specific platform.

Concept: AndroMDA

AndroMDA is the most uncomplex tool of all three and it is probably also the oldest of them three. That might be one of the reasons why AndroMDA is very well known among developers in this area and according to SourceForge statistics [AnSF01] AndroMDA has far more hits and downloads than both of the other two together.

AndroMDA might seem simple at first, but its been designed well. AndroMDA's core works on a plug-in mechanism for building the code. The plug-ins are called cartridges. A cartridge is nothing else but a bundle of files. Those files include a mapping file which is called the cartridge descriptor. This descriptor is written in XML and follows a simple DTD. It's nothing but a mapping declaration between classes in an UML class diagram and the actions AndroMDA should perform on these classes. The mapping is based on stereotype declarations. This means that AndroMDA calls its actions on each of the classes based on their stereotype. Those actions are bound to call scripts which are also bundled with the cartridge.

Those scripts (referred to as templates) are responsible for generating the code. They are written using VTL [] which is a rather simple but powerful scripting language. VTL is simple but features enough functionality to master the almost all of the tasks in AndroMDA.

Besides AndroMDA's age the simplicity is also responsible for its popularity. To achieve a model to code transformation AndroMDA follows the following steps: model some classes in UML, write some scripts and run the scripts against the model to generate the code. As AndroMDA's only able to provide access for UML class models it's by far no sophisticated approach and quite easy to handle if you're familiar with programming in a more object oriented way. Therefore the step from writing most of the classes is being substituted by letting AndroMDA generate them. One of the major drawbacks has been the lack of validating the models. With scripts being called directly on the XMI there was no way for checking the model (except for checking the resulting code of course).

This disadvantage/nachteilXXX is being eliminated with the new release of AndroMDA. One of the new features in AndroMDA 3 is the possibility to use metamodels through metafacades. This feature is still under heavy development that's why documentation on this subject is rather scarceXXX compared to the rest of AndroMDA's compressed but comprehensive documentation. Metafacades are just facades for the metamodel. Simply speaking they are MOF modules. With this extension it is now possible to get access to the information held in the UML models for your templates in a more object oriented way. Just like in open ArchitectureWare it is now possible to run your models against a metamodel and do some checking on the basis of constraints which could be almost anything. Just like in open ArchitectureWare the metamodels are written in Java. But the metamodels and therefore the templates as well could only refer to UML class models. This is a major drawback compared to the possibilities of open ArchitectureWare. With this constraint it is not possible to generate code from other diagrams which might be sufficient for many developers as it surely is possible to create a lot of information in a class diagram. But that also bears the risk that developers tend to design the application too close to the resulting platform at this point. With the definition of UML use case diagrams it would be possible to model access for different roles - something that would have to be placed in the class diagrams in another way. Characterizing everything through class diagrams is quite short-sighted as it does not take advantage of diagrams that would be much more suitable for modeling program flows. On the other hand this is probably the easiest way for many developers who aren't into UML and want to learn about a simple form of MDA quick. But overall program behaviour can't be described with just class models.

Operating on the metamodels the templates are able to access almost all of the data open ArchitectureWare can access through its metamodel. That includes associations, attributes in certain spellings (something which has become available in open ArchitectureWare through Markus Voelter's genfwutils[]) and so on.

Concept: openMDX

Standardized mappings from the MOF modeling constructs to Java; openMDX does not required to extend application models beyond the MOF standard. Models do not have to be tagged or enriched by component-specific or platformspecific attributes. As of current version 1.4 cares only about class diagrams but activity and state diagrams can be added at a later time. Class diagrams must be MOF compliant (which means: class must use only MOF model elements). This is the greatest difference compared to the other two tools: this way standard MOF mappings are being applied to the models and not to the metamodels. openMDX describes a development not too far fetched from reality. With chronological steps to take from modelling to developing to deploying the application. Developing means: implementing the generated interfaces. As the generated interfaces could become huge this is a time consuming task which can be supported by plugins. These plugins can be quite useful depending on the target (role plugin as example). JMI defines formal mapping of MOF to Java. openMDX follows the JMI specification except for two aspects:

Concept			
	open ArchitectureWare	AndroMDA	openMDX
PSM	no	no	no
Model validation	yes (through metamodels)	yes	no

Features

Features include code output, extensibility, validation.

Features			
	open ArchitectureWare	AndroMDA	openMDX
Model validation	yes	yes	no
Open Source	yes	yes	yes
Code ouput bound to specific language	no	no	no

Summary

Summarizing all of these aspects leads to...

Even if those tools do not fully comply to the MDA standard and do not offer a real PIM to PSM to code transformation like full-blown development platforms (like Compuware's OptimalJ [Com01] does) they still have the same potential as a standards conformXXX tool: Time2Market, reuse of models, etc.; lower development time and costs; Future: Request for Proposal at OMG for TPL from Compuware [] makes templates reusable among MDA tools; use case diagrams are being looked at nowhere which bears the risk for overdefining the application in other diagrams;

Links

- MDA using openArchitectureWare: mda_genfw.pdf
- MDA using AndroMDA: mda_andromda.pdf

[oAW01]	openArchitectureWare; http://archicteturware.sourceforge.net
[Andr01]	AndroMDA; http://www.andromda.org
[oMDX01]	openMDX; http://www.openmdx.org
[OMG01]	OMG MDA; http://www.omg.org/mda
[OMG02]	UML; http://www.omg.org/uml
[OMG03]	MDA committed products and companies; http://www.omg.org/mda/committed-products.htm
[OMG04]	http://www.omg.org/technology/documents/formal/xmi.htm
[TSS01]	Tilkov, Stefan: MDA from a Developer's Perspective; http://www.theserverside.com/articles/article.tss?l=MDA
[Voel01]	Völter, Markus; http://www.voelter.de
[XDOC01]	XDoclet; http://xdoclet.sourceforge.net
[NMI01]	Magic Draw UML from No Magic Inc.; http://www.magicdraw.com
[Gen01]	Poseidon for UML from Gentleware AG; http://www.gentleware.com
[Rat01]	Rational Rose from Rational Software Corp.; http://www.rational.com
[Bor01]	Together from Borland Software Corp.; http://www.borland.com
[JCP01]	JSR-000040 Java Metadata Interface Specification; December 5, 2001;http://www.jcp.org/aboutJava/communityprocess/review/jsr04
[Apa01]	JSR-000040 Java Metadata Interface Specification; December 5, 2001;http://www.jcp.org/aboutJava/communityprocess/review/jsr04
[W3C01]	XSL Transformations (XSLT) W3C Recommendation Version 1.0; November 16, 1999;http://www.w3.org/TR/xslt
[AnSF01]	SourceForge statistics on AndroMDA; XXXXXXXXXXXXXXXXXXXXXXXXXXXXxx
[Com01]	http://www.compuware.com/products/optimalj/
[OMG]	Request For Proposal: MOF 2.0 Query / Views / Transformations RFP; April 24, 2002; http://www.omg.org/docs/ad/02-04-10.pdf