Introduction to Model Driven Architecture (MDA)

Seminar Paper

June 2007

Igor Sacevski, sachead@gmx.net
Jadranka Veseli, jaca@gmx.at
Abstract

The purpose of this paper is to provide a general overview of OMG’s Model Driven Architecture (MDA). The paper describes the advantages and disadvantages in using the MDA from a business perspective. It also provides the basic specifications and functionality of the MDA as defined by OMG and gives a brief introduction to the most commonly used modelling languages and tools that support the MDA standard.

Keyword: Model Driven Architecture, MDA, Model Driven Development, MDD, UML, XMI, CWM, MOF, Modelling, OMG
1. History

OMG was founded in 1989 and today has a consortium from over 800 companies over the world. Some of OMG’s accomplishments are the Common Object Request Broker Architecture™ (CORBA™), Unified Modelling Language™ (UML™), Meta Object Facility™ (MOF™), XML Metadata interchange (XMI™) and the Common Warehouse Metamodel™ (CWM™). All of these standards contribute to making the idea of model driven development a reality. Roughly around 2001 OMG adopted a new framework called the Model Driven Architecture (MDA). Unlike the other standards of the OMG the MDA offers a way to use models instead of the traditional source code. It remains to be seen if this new way of software development will be accepted among the developers and the companies.

2. Why use models?

Model is an abstraction of a system or a part of it. Depending on the type of the model it can provide a simple view of the system or a more detailed one. In the software engineering models have a long and rich history. Designers often use models for planning their projects or explaining them to someone. These models are normally simple sketches that don’t contain any relevant information of the way the project is to be implemented. One preparing a presentation of a project we use somewhat more specialized models. Nowadays the most commonly used models are the UML models. In his book Martin Fowler explains three uses for the UML: UML as a Sketch, UML as a Blueprint and UML as a Programming language. The last one became a possibility with the new UML 2 which we will discuss later on, and also a good foundation for using the MDA. Today a large percent of the developers use the UML models parallel with the software systems they design. It provides an effective way for communication between the designers especially if the systems are complex and involve a number of teams in the design. What would be easier to understand: Graphical models or hundreds or thousands of lines of code?

3. Is MDA the next big breakthrough in software development?

Usually when working on a software project the requirements are often written documents with bullet points or textual scenarios, models are frequently just pictures with
annotated details of programming interfaces, programs are almost always source code in a programming language and "bug" reports are kept in databases or logs. But none of the relationships between these parts of the project is usually stored. This removes the possibility of automation of some task and does not provide any consistency between the various aspects of the software project. This is problematic especially in cases of refactoring or reusing a project or even extending one. This is where the Model Driven Development comes in. It is an approach to software development that uses a single source that contains all of this information. The main goals are simplifying and standardizing the activities included in a projects lifecycle. It raises the programming to an even higher level of abstraction. MDA defines standards for Model driven development. Programming has come a long way from machine code to assembler, FORTRAN and C. Now we have object oriented programming languages like C++ and Java. With each change the abstraction level raises as shown in Figure 1.

The higher abstraction level is suppose to make MDA easier to use and easier to understand. It also provides to certain degree of platform independence. Some of the other key benefits from using MDA are:

- **Portability**, increasing application re-use and reducing the cost and complexity of application development and management, now and into the future.
- **Cross-platform Interoperability**, using rigorous methods to guarantee that standards based on multiple implementation technologies all implement identical business functions.
- **Platform Independence**, greatly reducing the time, cost and complexity associated with re-targeting applications for different platforms—including those yet to be introduced.
• **Productivity**, by allowing developers, designers and system administrators to use languages and concepts they are comfortable with, while allowing seamless communication and integration across the teams.

This brings to a reduced cost throughout the application life-cycle, reduced development time for new applications, improved application quality, increased return on technology investments and rapid inclusion of emerging technology benefits into their existing systems. But this is not always the case. There is also a down side in using MDA.

"Raising the level of abstraction may lead to oversimplification when there is not enough detail for any useful purpose"

For instance it creates the issue of redundancy. According to the MDA principles there are multiple representations of artefacts inherent in a software development process, representing different views of or levels of abstraction on the same concepts. When these are manually created, duplicate work and consistency management are required. Another problem that comes from modelling is the round-trip problem. In complex systems a lot of models and artefacts are required, and several different levels of abstraction. This also increases the complexity of the relationships between them. So when a change needs to be done in an artefact that affect other artefacts and relationships in some cases it is impossible to automate the entire process. So a manual intervention is needed. And this is especially difficult if a change is done in the lower levels given the fact the most of the models at the lower levels are automatically generated. Sometimes not all the code can be generated automatically so developers need to manually edit generated code. But this also brings to question how well optimized will the generated code be. And MDA also requires more training and expertise. A developer must learn additional language or languages used in the modelling. Because artefacts resulting from any stage in the life cycle can impact those produced at any other stage, knowledge of different model technologies and terminologies must exist. Considering the sorts of transformation technologies that are a big part of MDA, developers also may have to be fluent in various transformation notations which can be extremely complex. This is a big issue that comes from the lack of standard used in the MDA. MDA does not offer a formal standardized system of languages used for modelling and specifications from the transformation, only suggestions. This reduces interoperability between tools because different manufacturers have different implementations. Because of this in some cases a project may become dependent on the tools it uses.

In the next part we will focus on what the basic principles of MDA are.
4. How does MDA actually work?

To explain the functionality of MDA we must explain some of the basic terminology. Some of the basic concepts of the MDA are the models: Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM) and Platform Model, and the transformation techniques, mainly mapping, trace and notation.

4.1. Models

CIM is basically a simple representation of a system without specific information about how it is to be implemented. It only show the system in an environment in which it will operate and helps to represent what the system is supposed to do. By MDA specifications a CIM is usually a simple UML model which should be traceable to the PIMs and PSMs that implement him.

PIM is somewhat more detailed view of a system containing more details but from a platform independent viewpoint. It contains details about business functionality and behaviour but no information about the technical details or the platforms on which it may be implemented. A common technique for achieving platform independence is to target a system model for a technology-neutral virtual machine.

PSM is a view of a system from the platform specific viewpoint. A PSM contains the specifications from a PIM but with details about the usage on a concrete platform. PSM usually contains enough information to allow code generation

Platform Model contains technical information about a specific platform and the services it provides. The Platform Model provides concepts for use in the PSM.

4.2. Model Transformation

Model transformation is the process of converting one model to another model of the same system. The platform independent model and other information are combined by the transformation to produce a platform specific model. This transformation can be done in many ways. However it is done, it produces, from a PIM, a model specific to a particular
platform. In cases where the PIM is based on virtual machine, transformations are not necessary. Instead the PIM of the virtual machine is transformed to a PSM for a particular platform. The PSM can then be transformed in a concrete implementation for that platform. An implementation provides all the information needed to construct a system and to put it into operation.

The transformation from one model to another can be done with mapping. Mapping in MDA provides specifications for the transformation of a PIM to PSM. There are different kinds of approaches in using mapping. There is the Model type mapping which specifies how models built using types defined in the PIM language are transformed into types defined in the PSM language. Example of such a mapping is the metamodel mapping in which the types of model elements in the PIM and the PSM are both specified as MOF metamodels. In this case the mapping gives rules and/or algorithms for the transformation of all instances of types in the metamodel of the PIM language into instances of types in the metamodel of the PSM language(s). Figure 2 illustrates this.

![Diagram](image)

Another type of mapping is the Model Instance Mapping. These types of mapping usually use Marks. This technique identifies model elements in the PIM that should be transformed in a specific way depending on the platform and marks them. Marks represent a concept in the PSM and specify how a marked object in the PIM is to be transformed. But
these marks can also contain information about requirements of a target platform and some of them may use parameters. This is shown in Figure 3.

Also a combination of the two types of mapping can be used. In this way you can collect much more information and create a more complex and detailed PSM from a PIM. A mapping may also include templates. Templates are parameterized models that specify particular kinds of transformations. These templates are like design patterns, but may include much more specific information to guide the transformation. In some cases additional information can be added to the transformation process. This additional information is provided by the developer and usually defines specific choices of implementations, quality of service etc. In Figure 5 is the MDA pattern for this type of a transformation.
After marking the PIM and collecting the necessary information from the metadata the transformation begins. As a result of the transformation we have the PSM to the specific platform and a Record of transformation. The record of transformation contains information of the transformation from every PIM element to the resulting element(s) in the PSM. This allows the traceability (the ability to round-trip across models) which is a very important part of the MDA. It provides the consistence between different types of models or between a model and a code. Without traceability after the transformations the only thing of value would be the implementation. The resulting PSM from the transformations can then be an implementation if it contains enough details about the system or it may act as a PIM that will be transformed to an even more detailed PSM that can become a direct implementation. There are also some tools that provide direct transformation from a PIM to a source code if the PIM contains all the needed information.

You can see a more specific example from the OMG of a MDA project life-cycle in Figure 5.

The metamodelling technologies often use an abstract syntax. But the end user needs to be exposed to a concrete syntax as in any other programming language. That is why notation is needed in order to create a standard for model driven development. A notation is a
type of model transformation specification that determines how to represent a model as a specific set of symbols. Such a representation of a model is the result of a transformation from an abstract to a concrete syntax.

4.3. Modelling languages and standards

In the practice most commonly used language in the PIM and the PSM now is the UML 2. It is scaleable to large software systems, modular structure for easier adoption (core + optional specialized sublanguages), increased semantic precision and conceptual clarity and represents a suitable foundation for MDA (executable models, full code generation). When it is necessary to introduce new notations or terminologies, UML provides user-defined extensions (profiles) through the use of stereotypes, tagged values and constraints. Two extensions are defined currently: Software Development Processes and Business Modelling. They allow the flexibility to create new elements or reassign existing elements to new meanings of existing UML meta-model. When using UML as a modelling language there are a lot of standard UML profiles for different platforms. Here is a list of the current UML profiles from OMG:

- Platform Independent Model (PIM) & Platform Specific Model (PSM) for Software Radio Components (also referred to as UML Profile for Software Radio)
- UML Profile for CORBA® and CORBA® Component Model (CCM) [This specification, nearly complete, will supersede the separate profiles listed just above.]
- UML Profile for Enterprise Application Integration (EAI)
- UML Profile for Modelling QoS and Fault Tolerance Characteristics and Mechanisms
- UML Profile for Schedulability, Performance and Time
- UML Profile for System on a Chip (SoC)
- UML Profile for Systems Engineering (SysML)
- UML Testing Profile
- UML Profile for Enterprise Distributed Object Computing (EDOC)

EDOC is composed of seven specifications:

- Enterprise Collaboration Architecture (ECA);
- Metamodel and UML Profile for Java and EJB;
- Flow Composition Model (FCM);
o UML Profile for Patterns;
o UML Profile for ECA;
o UML Profile for Meta Object Facility
o UML Profile for Relationships

There are also a lot of on-the-shelf UML profiles for platforms such as C#, C++, .NET etc.

The other three key OMG modelling technologies, all based on UML, support the MDA:

The Meta-Object Facility MOF is an abstract language used for defining and representing of meta-information. MOF is an OMG standard which introduces the concept of technology independent metamodel that could be used to define the modelling languages. MOF is at highest level of metalayer architecture so MOF is defined using MOF itself. It also enables the building of tools for defining modelling languages.

XML Metadata Interchange XMI is an interchange standard for UML models from OMG. XMI is usually used to map the MOF to eXtensible Markup Language (XML). It specifies how MOF compliant models could be represent in form of XML tags, helping in the translation of MOF based meta-models into XML Document Type Definition (DTD) thus allowing models to be translated into XML documents that are in accordance with correspondent DTDs. This XML based XMI data provides a serialized mechanism of artefacts produced by the UML and MOF models be able to interchange between various tools and repositories.

Common Warehouse Model CWM is another standard used to define a common interchange format for metadata in a data warehouse. It provides a common language and metamodel definitions for the data in the data warehouse. CWM is a modelling language that focuses mainly on modelling data warehousing applications. The metamodel has additional features like special metaclasses for modelling relational databases. In the context of MDA, the CWM specification is especially useful for legacy integration.

The basic principles of the MDA are best described by their logo in Figure 6:
The four principles that underlie the OMG's view of MDA:

- Models expressed in a well-defined notation are a cornerstone to understanding systems for enterprise-scale solutions.
- The building of systems can be organized around a set of models by imposing a series of transformations between models, organized into an architectural framework of layers and transformations.
- A formal underpinning for describing models in a set of metamodels facilitates meaningful integration and transformation among models, and is the basis for automation through tools.
- Acceptance and broad adoption of this model-based approach requires industry standards to provide openness to consumers, and foster competition among vendors.

5. Tools

There are a lot of MDA tools today available on the market some of them commercial and some open source projects.

The following list of MDA tools is provided by www.modelbased.net
5.1. Open source tools

- **Kermeta** - Developed by the INRIA Triskell team, the Kermeta workbench is an Eclipse-based OO metaprogramming environment dedicated to metamodel engineering activities, including specification of abstract syntax, static semantics (OCL) and dynamic semantics with connection to the concrete syntax, model and meta-model prototyping and simulation, model transformation and aspect weaving. Kermeta is built as an extension to **Eclipse EMF**.

- **MOFScript**, a model to text transformation tool, based on one of the OMG MOF Model to Text Transformation submissions - Eclipse plugin, based on metamodels/models in EMF.

- The IBM Model Transformation Framework (MTF) is an EMF based model transformation framework, for now available at alphaWorks. It provides a declarative means of specifying metamodel relationships, similar to that of QVT relations.

- The **ATL Engine** is a QVT-like transformation language, developed by the INRIA Atlas team. ATL is one of the core technologies in the **Eclipse M2M** project. It is developed as a set of Eclipse plugins and works as a development IDE for transformations, with execution and debugging. Currently integrates with EMF and MDR.

- **ModFact** A MOF Repository and QVT-like engine from LIP6, Paris. Based on the TRL language. LIP6 are also working on an open source ModelBus implementation, which will enable MDD tools interoperability.

- **Kent Modelling Framework** (KMF), a tool for generation of languages with support for dynamic constraint checking.

- **OpenArchitectureWare**, a flexible, template-based generator framework integrated with XMI.

- **Generative Model Transformer** (GMT), an eclipse incubator project that provides research-oriented model transformation technology for the eclipse platform. Several tools are part of GMT: AMW (Model Weaving), Epsilon (Model Merging), MoDisco (Model Discovery), MOFScript (Model 2 Text), openArchitectureWare, UMLX (Graphical Transformation), VIATRA2 (Visual Automated Transformations).

- **OpenMDX**, an open source MDA environment, which integrates with several tools through XMI and supports code generation towards several target platforms (J2EE, .Net).

- **AndroMDA**, an open source template-based tool for J2EE code generation from UML/XMI. Uses VTL (Velocity Template Engine) as scripting language and Netbeans MDR as a model API.

- **XDoclet**, an open source, attribute based code generation tool for J2EE. Not really model-based, but can be combined with generation tools such as UMT to achieve good model-based value.

- **Middlegen**, an open source, database driven code generator based on JSBC, Velocity, Xdoclet and Ant.

- **OOMEGA**, an open source, integrated modelling and metamodelling tool for defining languages and code generators. Supports db4objects, Hibernate, Versant, OOMEGA's MemoryDB, XML and SDF (a standard binary codec) for storing both, the metamodels and models. Code generation templates are written in JSP where you can inspect model information via a powerful and unified Persistence API.
5.2. Commercial tools:

- ArcStyler, is a commercial MDA tool from Interactive Objects. It is bundled with MagicDraw UML-tool, but can also support other UML-tools through tool adaptors.
- MCC (Model Component Compiler), a commercial product from InferData, supporting generation towards J2EE
- OptimalJ, a commercial product from Compuware, uses a notation of patterns to achieve PSM transformations. Has an integrated UML tool for analysis, but uses a slightly different notation (structural) for the MDA-part of the tool.
- Xactium XMF Mosiac, a commercial model-based mapping, generation and execution tool suite
- SosyInc Modeler and Transformation Engine - The transformation engine provides generation of GUI and server-side, based on models OASIS/UML and rules for application structure and business rules.
- Model-in-Action, and MDA tool suite from Mia software supporting code generation and model to model transformation in a flexible framework.
- MetaEdit+, an integrated modeling and metamodeling tool for defining languages and code generators. Supports XML and SOAP/Webservice interfaces for both the metamodels and models.
- MDWorkbench is a text and model transformation toolset, which supports any kind of metamodel as input. The tool is based on Eclipse and EMF and it is available in a free edition (requires registration), an academic edition, and a profession edition.
- iQgen 3.0 is a template-based generator that can import models in different formats, including XMI, XML and ECore, and uses templates written as JSP templates

6. Conclusion

As we have seen from this paper OMG’s MDA is widely accepted and there are a lot of tools supporting this. It offers a lot of advantages over the current way of software development, but there are some down-sides particularly with the standardizations. OMG is working on formal standards for parts of the development process but this is something that is necessary for the acceptance of model driven development. MDA has a chance to succeed in large, distributed, industrial software development, but it is still not so useful in small business and small development project. However MDA is a work in progress and still has a lot of issues that need to be resolved before we can say for sure.
7. Trademarks

MDA, UML, XMI, MOF, CWM and CORBA are registered trademarks of the Object Management Group.

8. References

[2] Model Driven Architecture (MDA) FAQ, Object Management Group, 
   http://www.omg.org/MDA/FAQ_mda.htm#whatismda
   Addison Wesley Professional, March 2004
[4] Hailpern and Tarr: Model-driven Development: The Good, the bad and the 
[5] Object Management Group, Model Driven Development homepage: 
   http://www.omg.org/mda
[7] Alan Brown, Staff IBM: An introduction to Model Driven Architecture Part II, 
   Technologies, Position paper on ECOOP 2001
[9] Steve Cook, Microsoft Corporation: Domain-Specific Modeling and Model 
   Driven Architecture MDA Journal, January 2004
[10] M. Fowler, UML As Sketch (2003), 
    http://martinfowler.com/bliki/UmlAsSketch.html
    http://martinfowler.com/bliki/UmlAsBlueprint.html
    http://martinfowler.com/bliki/UmlAsProgrammingLanguage.html