What is MDD/MDA and where will it lead the software development in the future?

Matthias Gally

Student of the Software Engineering Group of the
Department of Informatics at the University of Zurich,
Binzmühlestrasse 14, 8050 Zürich, Switzerland
mgally@access.unizh.ch, May 2007

Abstract. The role of modeling has become much more important in the last years and the potential benefits of using models are significantly greater in software than in any other engineering discipline[3]. New technology, which can get software development to a more abstract level, has been coming up.
The topic of this paper is to investigate Model-Driven Development (MDD), Model-Driven Architecture (MDA) and its future influence on software development. Therefore it will firstly introduce the concepts of MDD and MDA. Later in the text the benefits and problems will be showed. At the end of the paper there is a discussion about the interpretation of the MDA model and the future of the MDA topic.
Today the MDA approach has reached the study levels of undergraduate students and is therewith a serious vision for the future. Especially on large projects there is a greate potential benefit.

1 Introduction

Since the beginning of software development people are trying to get automated solutions for perseverative jobs. The goal is to find the fastest way from modeling to code generation. The task of Model-Driven Development (MDD) and especially Model-Driven Architecture (MDA) is moving a big step closer to this goal.

The Object Management Group (OMG) has created a Model-Driven Architecture model to create a base for MDD. There are a lot of benefits by using the MDA approach (e.g. Faster, better and cheaper production of applications, optimized source code, automatical use of design patterns, etc.). Certainly there are also problems like there is no explanation of how entities in the infrastructure relate to the real world, as this is possible by developing applications by hand.

2 What is MDD

Modeling is a way to think issues through before you code, because it lets you think at a higher abstraction level[1]. The idea of Model-Driven Development (MDD) covers basically the following development approaches: models, modeling and Model-Driven Architecture (MDA).

2.1 Behind MDD

Model-Driven Software Development (MDSD) keeps you away from coding the program. The generation of the code is mostly done software aided later in the development process. It is possible to concentrate the work on the development of the models and to come much closer to the problem domain, than by defining and considering the underlying implementation. This is possible because the necessary automation techniques have matured and industry-wide standards have emerged[3]. Nevertheless there are also risks of manual code completion. If code is integrated by hand later in the development process, it can happen, that this will cause some damages to the application. The goal is to let the tool generate all the code.

There are 5 things an MDD supporting infrastructure must define[2]:

1. The concepts available for creating models and clear rules which govern their use.
2. The notation to use in depicting models.
3. It has to be clear, how the model's elements represent real-world elements and software artifacts.
4. Concepts to facilitate dynamic user extensions to model concepts, model notation, and the models created from them.
5. Concepts to facilitate the interchange of model concepts and notation, and the models created from the concepts to facilitate user-defined mappings from models to other artifacts.

2.2 The advantages of models and modeling

Models are used to understand complex real-world systems in all forms of engineering tasks. The underlying motivation for MDD is to improve productivity.

There are two basic ways to deliver this benefit[2]:

1. Short-term: By increasing a primary software artifact's value in terms of how much functionality it delivers.
2. Long-term: By reducing the rate at which a primary software artifact becomes obsolete.

If we define the kinds of models that must be produced, and apply some rigor to the precise semantics of these models, we can define rules for[5]:

1. Automating many steps needed to convert one model representation to another.
2. Tracing between model Elements.
3. Analyzing important characteristics of the models.

2.3 The disadvantages of models and modeling

There are also some disadvantages of models and modeling. The abstraction level of the MDD approach is quite high and not everybody is able to successfully create a good model on this level of abstraction. The models have to be made by skilled people but these are rare.

There is also the problem, that the competition keeps vendors away from just using the standard. Almost every vendor creates anywhere some vendor specific code to keep their customers. It is interesting, that there are fewer vendors which provide standardized import and export functionality.

3 Model-Driven Architecture (MDA)

3.1 Overview

The MDA approach generally separates the system functionality from the implementation details. It is a framework for Model-Driven Software Development (MDSD) defined by the Object Management Group (OMG). MDA is language, vendor and middleware neutral and therefore a very interesting topic for every software development company[6].

The focus of MDA lies on the modeling task. Firstly you build a Computation Independent Model (CIM). Then you build a Platform Independent Model (PIM). To create the PIM you use UML, MOF and CWM (figure 1). And then you automatically create a Platform Specific Model (PSM) out of the PIM. The interesting thing is, that you can fully concentrate the development on the functionality and behaviour of the software and leave technology on the side.

When you're finished with the PIM you can transform your PIM in any proprietary platform you want (e.g. CORBA, J2EE, .NET, XMI/XML)[6]. This is the step of automatical code generation from PIM to PSM. The specific code can be for pervasive services, security, events, transactions, directory and more (figure 1). From there you have the base to go to every domain you like (finance, e-commerce, telecom, healthcare, transportation, space, manufacturing, and more).

MDA offers you also platform interoperability, portability, platform independence and productivity. If you once have completed your PIM, you can switch to another technology by regenerating the code from it[6].

MDA has nothing to do with the CASE-Approach (Computer Aided Software Engineering) from the past. The goal of the CASE approach has been, to describe complete technical requirements. MDA hasn't the goal to get a complete automatization.

There are four principles that underlie the OMG's MDA approach[5]:

1. Models expressed in a well-defined notation are a cornerstone to system understanding for enterprise-scale solutions.
2. Building 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.

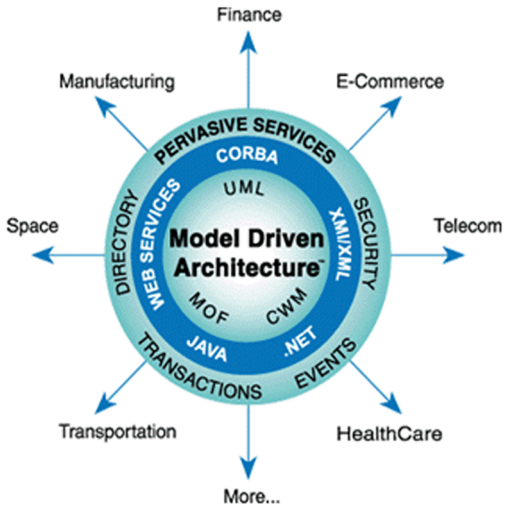

Fig. 1. OMG's Model-Driven Architecture model [10]

3. 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.
4. Acceptance and broad adoption of this model-based approach requires industry standards to provide openness to consumers, and faster competition among vendors.

3.2 The key standards

The core standards of MDA are[6]:

Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing and documenting the artifacts for software systems and can be used for designing models in PIM.

Meta Object Facility (MOF) is a integration framework for defining, manipulating and integrating metadata and data in a platform independent manner. It is the standard language for expressing metamodels. A metamodel uses MOF to formally define the abstract syntax of a set of modeling constructs[5].

XML Metadat Interchange (XMI) is a integration framework for defining, interchanging, manipulating and integrating XML data and objects. XMI can also be used to automatically produce XML DTDs and XML schemas from UML and MOF models.

Common Warehouse Metamodel (CWM) is a set of standard interfaces that can be used to enable easy interchange of warehouse and business intelligence metadata between warehouse tools, warehouse platforms and warehouse metadata repositories in distributed heterogeneous environments. It is a good example of applying the MDA paradigm to an application area.

3.3 The MDA Process

In the MDA-Approach there are four major steps[6]:

1. Generation of a Computation Independent Model (CIM).
2. Building a PIM model.
3. Transforming the PIM model into a PSM model.
4. Generate the code out of the PSM model.

Generation of a CIM The first step is done completely without computer support. This model is common language based and belongs to the Requirements Engineering process.

Building a PIM model This model is independent of any implementation technology and has a high level of abstraction. At this level UML is used as modeling language. This step specifies functionality and behaviour. Because the MDA is technology-independent at its core, an application or standard defined in the MDA can be implemented equivalently and interoperable on one or several middleware platforms[12]. MDA models must be extremely detailed: The application will be generated from it, and will include only functionality represented explicitly - in the MDA, the business designer and architects play a crucial role[12].

Transforming PIM to PSM The complete PIM is stored in Meta Object Facility (MOF) which is the input of the mapping step. The mapping step then produces a PSM.

There are 4 levels of automation[6]:

1. Transformation is done completely by hand
2. Using established patterns to convert from the PIM to a particular PSM
3. Implemented algorithm in a MDA tool produces a skeleton PSM, which is then completed by hand
4. Tool is able to produce the entire PSM

Generate Code This step produces the implementation of the PSMs for the particular platform. Therefore the PSM is taken as input and a tool is used to do the transformation.

3.4 MDA Development

MDA styles of development are being widely discussed in the software industry today as a way of increasing the quality, efficiency and predictability of large-scale software development[5].

The traditional way is to go back to the requirements (CIM) which are text based and then go ahead with analysis, low-level design, coding, testing, deployment.

The MDA way is to go back to the analysis which is the PIM and directly change the model. The automated processes then lead you through the other processes. The PSM is produced automatically and out of the PSM the code is generated. It is obvious that this automation process saves a lot of time.

3.5 MDA Transformation

Transformation is the automatic process of converting one artifact into the desired artifact, using a tool, providing a rule that describe how to transform it[6].

Transformation rules are the guidelines of how to use one model to generate others.

Transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. The definitions are used by the transformation tools to generate the transformations.

Transformation tool performs a transformation for a specific source model according to a transformation definition.

3.6 Benefits of MDA

This is a list of the major benefits of MDA:

1. MDA-enabled tools follow OMG-standardized pathways to automate the transformation from your designers' business model into your implementation, producing new applications faster, better and cheaper[12]. Faster and better in terms of better maintainability and cheaper in terms of less costs for code generation.
2. The MDA process ensures not only that the business requirements built into the design end up in the final implementation, but also that non-business functional requirements (such as scalability, security and transactionality) carry through as well[12]. For these topics the developer can trust on the tool.

3. Code generated by MDA-enabled development tools derives from libraries based on patterns designed by the industry's best developers. Few companies have access to programers with the skill level to produce code as well suited for infrastructure and application as the next few generations of MDA tools will produce[12]. It is quite difficult to make a good architecture, which is also in the future satisfactory.
4. MDA applications interoperate: The MDA, designed from the start to implement in multiple middleware platforms, codes cross-platform invocations where needed, not only within a given application, but also from one to another regardless of the target platform assigned to each[12]. Gives more possibilities for the useability of an application.
5. MDA applications are portable: Based on technology-independent business models, they can be generated on any middleware platform[12]. This leads to very high flexibility.
6. MDA applications are future-proof: When new platforms are introduced, OMG will add mappings for them and tool vendors will implement them in their offerings. Using these new mappings, existing MDA-based applications can be made either to interoperate with others, or can be reimplemented on the new platform entirely[12]. This benefit depends on the willingness of the vendors to use the mappings.
7. The MDA supports enterprise application integration (EAI): Think of your entire enterprise's suite of applications as a library of UML models. Select the ones that need to work together, bring them into your MDA tool simultaneously, and draw lines denoting the interoperability pathways. When you generate the applications, the invocations will be part of it[12]. This generates business benefits.
8. In an MDA development project, attention focuses first on the application's business functionality and behaviour, allowing stakeholders' investment to concentrate on the aspects that critically affect core business processes. Technical aspects, also critical but secondary to business functions, are well-handled by automated or semi-automated development tools[10]. This leads to a better application for the customer.
9. An architecture based on the MDA is always ready to deal with yesterday's, today's and tomorrow's 'next big thing' and makes it easier to integrate applications and facilities across middleware boundaries[10]. This benefit is right under the precondition that the OMG will create a mapping for this 'next big thing'.
10. Domain facilities defined in the MDA by OMG's domain task forces will provide much wider interoperability by always being available on a domain's preferred platform, and on multiple platforms whenever there is a need[10].

3.7 Problems of MDA

There are four key problems:

1. No explanation exists of how entities in the infrastructure relate to the real world. Does the infrastructure accommodate real-world elements or do they

lie outside[2]? This problem occurs because many steps of the development are executed automatically and therefore it is not possible to get relations to the real world.
2. The infrastructure implies that all instance-of relationships between elements are fundamentally of the same kind. Is this a valid approach or should we be more discriminating[2]. A tool always has to follow certain rules and needs some generalisation descisions.
3. No explicit principle exists for judging at which level particular elements should reside. Using a single instance-of relationship to define the metalevels always seems to introduce inconsistencies. Can we use instantiation to define metalevel boundaries and if so, how[2]?
4. A preference exists for using metalevel description (sometimes in the form of stereotypes) to provide predefined concepts. How can we integrate the other way of supplying predefined concepts - libraries of (super-) types, to be used or specialized - within the architecture[2]?

4 The future of MDA

4.1 Overview

Today, a majority of software developers still take a code-only approach and do not use separately defined models at all. They rely almost entirely on the code they write and they express their model of the system they are building directly in a third-generation programming language such as Java, C++[5].

In the model-only approach developers use models purely as thought aids in understanding the business or solutions domain or for analyzing the architecture of a proposed solution. Models are frequently used as the basis for discussion, communication, and analysis among teams within a single organization, or across multi-organizational projects[5].

In practice the implementation of a system, whether from scratch or updating an existing solution, may be practically disconnected from the models. An interesting example of this approach can be seen in the growing number of organizations which outsource implementation and maintenance of their systems while maintaining control of the overall enterprise architecture[5].

4.2 Where will it lead?

There are a lot of questions when you start to think about the future of MDA. Are there so many developers who have the sophisticated modeling skills required to use MDA-based tools? Do the other developers have the desire to gain those skills[8]? Scott Ambler a well known software developer, and book writer suspects that the vast majority of developers are tended to non-visual artifacts such as use cases and tests instead of UML sequence diagrams and class diagrams for specifying software[1].

The success of MDD relies on the careful introduction to the existing technological and social environments. One reason why we're moving toward MDA is

that the technologies we use have grown significantly more complex over the last few years. Technology changes faster than the businesses we're trying to support with this technology[13].

If we have a look at compiler building, there is no discussion about the automatic code generation. Hardly anyone questions compiler technology these days because it is quite mature and extensively proven in practice. In some cases, it might be possible for domain experts rather than computing technology specialists to produce systems in the future[3]. That can be because a key premise behind MDD is that programs are automatically generated from their corresponding models.

4.3 Two interpretations of MDA

A closer look at what tool developers and methodologists are doing reveals that there are two rather different interpretations of how the MDA vision might be achieved. This can have a major impact on the success of the MDA approach in the future.

The elaborationist In the elaborationist approach, the definition of the application is built up gradually as you progress through from PIM to PSM to Code[7]. There is a possibility to elaborate and add further information or detail on the PSM and the Code level. Therefore it is important, that there is resynchronisation ability as we see in figure 2, which changes the PIM also upwards.

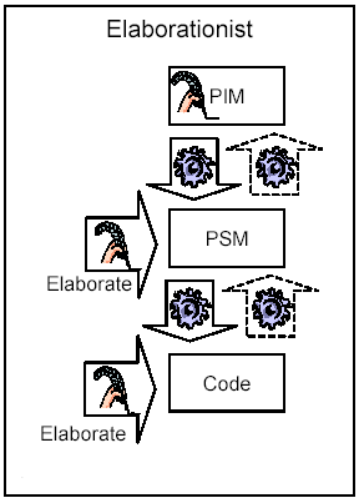

Fig. 2. Model of the elaborationist approach [7]

The translationist In the translationist approach, the PIM is translated directly into the final code of the system by code generation as it is shown in figure 3[7]. The PSM is an internal step of the process and not visible or editable by the developer.

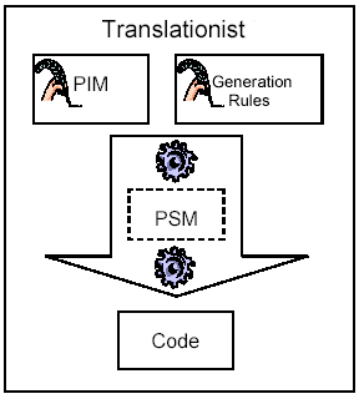

Fig. 3. Model of the translationist approach [7]

There is a big debate between these two approaches. The elaborationists say that "Exectuable UML [=translationist] is suitable within specialized domains, but even there the benefits are less than you would expect ..."[7]. The translationists say, that elaboration is stupid. The translationist approach derives from work in real-time/embedded systems, where the use of statemachines for specifying behaviour in this type of system has a long history. In business information and transactional systems, state machines are seldom used. There it is normal to use Use Case and Collaboration diagrams of UML. Unfortunately for the developers these diagrams are not suitable for code generation or model execution and do not feature in MDA as mainstream artefacts.

The decision of the winners, will probably come from the vendors. Supposedly there will be vendors for both translationists and elaborationists, but this leads them also away from the task of standard thinking and overall use.

5 Summary

MDA is a framework for software development, defined by the OMG. It defines guidelines for structuring specifications expressed as models[11]. The MDA promotes an approach where the same model specifying system functionality can be realized on multiple platforms through auxiliary mapping standards, or through point mappings to specific platforms. It also supports the concept of explicitly relating the models of different applications, enabling integration and interop-

erability and supporting system evolution as platform technologies come and go.

As we can see that the term of Model Driven Development has reached the study levels of undergraduate students, the MDA approach is a serious vision for the future, which is partially already realized. There have been already several projects, where this technique has lead to success and cost savings. There is a potential benefit of this approach on large projects, but it will go probably much longer, to get a benefit for small projects out of it.

5.1 What's next?

University classes are already teaching UML, and bookshelves are beginning to fill with MDA material. This will automatically lead the software industry to develop more model-driven in the future. PricewaterhouseCoopers' technology centre predicted in 2003 that MDA will be one of the most important methodologies in the next two years[7]. As we are in the year 2007 now, we can say that MDA has a place in software industry, but it is a long way, to get the MDA approach everywhere useful.

Even if there is a standard from the OMG, there will be the problem, that vendors will often claim support for a standard but then for competitive reasons implement their own version of it. It is quite difficult to find a vendor tool, that let's you export back and forth without loss of information[1].

There have been already successful industrial MDA software development projects (e.g. Deutsche Bank Bauspar or the Austrian National Railroads), reporting total savings around 40 percent[13]. Certainly we will hear much more on this topic in the next years as you can read already several articles in weekly magazines today.

References

1. Ambler, S.W.: Agile Model Driven Development Is Good Enough, IEEE Software, Vol.20, No. 5, (http://www.agilemodeling.com/shared/mda.pdf). (2003) 71 - 73,
2. Atkinson, C., Kühne, T.: Model-Driven Development: A Metamodeling Foundation, IEEE Software, Vol.20, No. 5. (2003) 36 - 41
3. Bran Selic: "The Pramatics of Model-Driven Development", IEEE Software, Vol.20, No. 5, September / October 2003, p. 19-25
4. Czarnecki, K., Antkiewicz, M., Hwan, C., Kim, P., Lau, S., Pietroszek, K.: Model-driven software product lines, OOPSLA 05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, San Diego, CA, USA. (2005) 126 - 127
5. Drown, A.W., Conallen, J., Tropeano, D.: Introduction: Models, Modeling and Model-Driven Architecture (MDA), IBM software group, Durham USA
6. Mashkoor, A.: Investigating Model Driven Architecture, master thesis, Department of Computing Science, Umea University, Sweden. (2004) 15 - 25
7. McNeile, A.: MDA: The Vision with the Hole?, (http://www.metamaxim.com/download/documents/MDAv1.pdf). (2003)

8. Miller, G., Ambler, S., Cook, S., Mellor, S., Frank, K., Kern, J.: Model driven architecture: the realities, a year later, OOPSLA 04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, Vancouver, BC, CANADA, (http://doi.acm.org/10.1145/1028664.1028719). (2004) 138 - 140
9. Mukerji, J., Miller, J.: Model Driven Architecture, Document number ormsc/2001-07-01, Architecture Board ORMSC, Draft, (http://www.omg.org/cgi-bin/doc?ormse/2001-07-01). (2001)
10. OMG Web site: OMG Model Driven Architecture, (http://www.omg.org/mda). (May 2007)
11. Scott W. Ambler, S.W.: (http://www.agiledata.org/essays/enterpriseArchitectureTechniques.html). (May 2007)
12. Siegel, J.: OMG's Model Driven Architecture, Software development times, (http://www.sdtimes.com/article/special-20021015-01.html). (May 2007)
13. Uhl, A.: Model Driven Architecture Is Ready for Prime Time, IEEE Software, Vol.20, No. 5 (http://www.agilemodeling.com/shared/mda.pdf). (2003) 70 + 72