Model Driven Engineering: A Position Paper

Marcus Alanen, Johan Lilius, Ivan Porres and Dragos Truscan
Software Construction and Embedded Systems
Laboratories
Turku Centre for Computer Science
Lemminkäisenkatu 14, FIN-20520 Turku, Finland

Abstract

The Model Driven Approach (MDA) as supported by the Object Management Group (OMG) describes the structural requirements of an engineering discipline where models, instead of source code, comprise the primary artifact. Model Driven Engineering (MDE), as outlined by Stuart Kent, brings forth the dynamic aspects of engineering, where process adherence and rigorous commitment to analysis are equally important. As such, MDE has a broader scope than MDA. We discuss our position on MDE and its requirements on tools and technology, especially considering the dynamics of a model-based software development method. We demonstrate our approach with an example of the specification of an IPv6 router targeted to a customized processing architecture.

1 Introduction

Model Driven Engineering (MDE) tackles the elusive problem of system development by promoting the usage of models as the primary artifact to be constructed and maintained. The term was first proposed by Kent in [1] and is probably derived from the OMG's Model Driven Architecture (MDA) initiative [2]. OMG's MDA is based on the idea of platform independent models (PIM) and platform description models (PDM) that can be realized using a variety of middleware and programming languages into platform specific models (PSM). We understand MDE as a broader term that includes all models and modeling tasks needed to carry out a software project from beginning to end.

We consider that OMG's vision of MDA, although valid, is just one of the possible scenarios in an MDE process. A PIM to PSM transformation may be a necessary task in an MDE context, but also PIM to PIM transformations, e.g. how a PIM representing some customer requirements can be transformed into another PIM that realizes those requirements. In our understanding, the main key concept behind MDE is that all artifacts generated during software development are represented using common modeling languages. As a consequence,

software development can be seen as the process of transforming a model into another until it can be executed outside its development environment. If we only study PIM to PSM transformations as described in the MDA approach instead of a more general framework, we may miss important issues and be unable to provide a general solution to a broader problem. This scenario occurs if we consider the OMG standards as an authoritative description of the way that we build software instead of as an authoritative description of the way that we represent our software.

The current OMG standards present a static and structural view of models. They define several standard modeling languages, e.g. what is a valid model in a given language using OCL constraints and how to store a model in a file using XMI [3]. However, they do not discuss how models are created or how models evolve. This may be explained by reviewing the origins of UML: it was developed as a method-independent notation to document software artifacts. UML can be used in combination with practically any software development method and, as a consequence, the OMG standards do not contain any reference or support for software development. We believe that the OMG standards should also consider the dynamic aspects of model development. This ranges from the basics of model evolution using algorithms for model transformation to more sophisticated reasoning about why a model transformation meets new requirements. We may consider that the Software Process Engineering Metamodel standard (SPEM) [4] addresses this issue. However, SPEM tells us how to document a process, while "planning and executing a project using a process described with SPEM is not in the scope [of the standard]".

In this position paper we summarize our approach to defining Model Driven Engineering methods. We discuss the basic collection of concepts, methods and tools needed to support such methods. The ideas presented here have been implemented both in the SMW toolkit, and the newer Coral toolkit. We have verified the validity of these ideas by developing an instance of a MDE Method for developing protocol processing applications. The method has been applied to the specification and implementation of an IPv6 router both on a software platform, using the Java programming language, and on a hardware platform, using the TACO [5] protocol processing architecture.

2 Model-Driven Software Development Methods

We define a model-driven software development method as a software construction method where all the relevant information in the project is stored in some kind of abstract model. Model development is then carried out as a sequence of model transformations.

Model driven engineering is the result of the recent development on computer languages, awareness of the need of software development methodologies

and the constant need to tackle larger and more complex development projects. These forces are not new. Indeed, we could use the same naming pattern to create terms such as punched card driven development, to describe the development methods used when compiler time was a luxury, or source code driven development, to describe the methods used in Extreme Programming and many open source projects, where source code is the key artifact. However, we believe that MDE opens a window for new development methods and tools that are not available or are too expensive to implement in other approaches such as source code driven development. These tools and methods can take profit of the fact that the artifacts describing our software are stored in a standardized way and are, to a certain extend, independent of the implementation technology.

The description of a model driven engineering method should contain all the elements that are usually present in any software development method. It should describe which final deliverables and intermediate milestones should be produced, which language should be used to create the previous artifacts and which tasks we should perform, and in which sequence, so that we can effectively create the required artifacts. However, we consider that there exists two main differences in a model driven engineering method with respect to a traditional development method. First, all artifacts are represented using a well-defined modeling language. Secondly, and as a consequence, we can create tools that process and transform all the artifacts in our projects. Therefore, we will require that all tasks in a model driven engineering method should be performed with the assistance of specialized tools. In this context, a clear understanding of model dynamics is a prerequisite to define any MDE method.

We have identified a four-layer approach to model dynamics. Each layer depends on the functions provided by the layers beneath it. Every layer empowers the modeling environment with new dynamic aspects, which would not be possible by the lower layers alone.

- In our approach, layer 0 defines the basic model management possibilities. This consists of creation and deletion of model elements, modification of the various associations between elements, and the evaluation of model constraints. In essence, the power at this first layer is the power given by the Meta Object Facility (MOF) [6]. A good overview of basic model manangement, and the use of Python as a scripting language for e.g. model constraint evaluation is given in [7].

- Layer 1 acknowledges model evolution as a continuous temporal process. Here, versioning is the key element, whereby tools can support undo/redo facilities, displaying and calculating differences between models, merging of models from multiple collaborative sources and, finally, providing full revision control of the development process. Several of these issues are discussed in [8], [9] and [10].

- Layer 2 implements the desired behavior using interoperable tools with editors, and transformation rules. This requires a complete set of modeling standards for the various activities that developers can rely on.

    1. *Queries* are applied on a model expressed in one language and returns a set of elements of the same model expressed in the same language.
    2. *Model Transformations* are applied on a model expressed in a given language and either modifies the model in place, or creates a model, possibly expressed in a different language. The upcoming QVT standard from OMG addresses this problem. Model transformations conceive a plethora of new interesting questions and topics, such as transformation taxonomy, correctness-preserving transformations, consistency checking and/or verification.
    3. *Code Generation*, although a form of transformation, is sufficiently different from a model-to-model transformation to merit its own classification. The goal is to produce suitable input to a second-stage compilation or analysis tool. The target language is not a metamodel.

    The main difference between queries and transformations is that the queries are free of side-effects, meaning that when applied on a model they do not change the model in any way. Many examples of queries and model transformations are given in [11].

- Layer 3 includes *intent* in model development. It studies why changes are made in a model, and when the method in use *allows* us to make the change. Far too long has intent and process adherence been considered a second-class citizen in software engineering. We regard MDE as a possible savior, by enabling us to describe process methodologies, problem analysis, estimations, and testing frameworks together with an evolving platform description model, and with the consistent ways to describe models, metamodels, transformations and constraints in MDA. We are currently developing a MDE based flow which is targeted towards protocol processor design. The basic philosophy behind the flow has been discussed in [12]. The technical report [11] discusses the implementation of the flow.

3 Conclusions

In this position paper we have discussed the main ideas behind our research on Model Driven Engineering. More information on the current status of the project can be found on the MDE web-site `http://mde.abo.fi`, where

you can also find both the SMW-tool (which is discontinued) and the new Coral toolkit.

Acknowledgments

Dragos Truscan gratefully acknowledges the financial support for this work from the HPY and TES research foundations.

References

[1] Stuart Kent. Model Driven Engineering. In *IFM 2002*, volume 2335 of *LNCS*. Springer-Verlag, 2002.

[2] OMG. OMG Model Driven Architecture. Document ormsc/2001-07-01, available at `http://www.omg.org/`, July 2001.

[3] OMG. OMG XML Metadata Interchange (XMI) Specification. OMG Document formal/00-11-02. Available at `http://www.omg.org/`.

[4] OMG. Software Process Engineering Metamodel Specification (SPEM). OMG Document formal/02-11-14. Available at `http://www.omg.org/`.

[5] Seppo Virtanen, Dragos Truscan, and Johan Lilius. TACO IPv6 Router - A Case Study in Protocol Processor Design. Technical Report 528, Turku Centre for Computer Science, April 2003.

[6] OMG. OMG Meta-Object Facility (MOF). OMG Document formal/01-11-02. Available at `http://www.omg.org/`.

[7] Ivan Porres. A toolkit for model manipulation. *Springer International Journal on Software and Systems Modeling*, 2(4), 2003.

[8] Marcus Alanen and Ivan Porres. Difference and Union of Models. In *UML 2003*, volume 2863 of *LNCS*. Springer-Verlag.

[9] Ivan Porres and Marcus Alanen. A generic deep copy algorithm for MOF-based models. Technical Report 486, TUCS - Turku Centre for Computer Science, Nov 2002.

[10] Marcus Alanen and Ivan Porres. Issues on the design of an XML-based configuration management system for model driven engineering. Technical Report 567, TUCS, Nov 2003.

[11] Marcus Alanen, Johan Lilius, Ivan Porres, and Dragos Truscan. Realizing a model driven engineering process. Technical Report 565, TUCS, Nov 2003.

[12] Johan Lilius and Dragos Truscan. UML-driven TTA-based Protocol Processor Design. In *Forum on specification and Design Languages (FDL'02)*, September 2002.