New Roles in Model-Driven Development

Jan Øyvind Aagedal and Ida Solheim

SINTEF Information and Communication Technology, Forskningsvn 1, N-0314 Oslo, Norway
{jan.aagedal|ida.solheim}@sintef.no

Abstract. In this paper we outline a set of roles that are needed in model-driven development (MDD). The set of roles are based on state-of-the-art component-based methodologies, and we add new roles to accommodate the new activities of meta-modelling, transformation specification and method engineering. Finally, we list a set of tools to support the proposed roles.

1 Introduction

Model-driven development (MDD) has been advocated by academia and industry for many years. Today, most of the popular and widely used software engineering (SE) methodologies use models as the primary tool to develop software, and can thus claim to follow a model-driven approach (e.g., [1, 2]). This trend has increased as a consequence of the Model Driven Architecture initiative (MDA®) [3] launched by the Object Management Group (OMG). During its relatively short lifetime, MDA has gained a lot of attention by SE researchers, practitioners, tool vendors and others. MDA promises an integrated framework for model-driven software development. Since the Unified Modeling Language (UML™), the Meta Object Facility (MOF™) and the Common Warehouse Metamodel (CWM™) are in the core of the MDA, the *models* are the core artefacts of an MDA-based development process. An important part of the MDA vision is to equip developers with fully integrated tools to support the development of system models as well as executable code. These tools should provide synchronization of code and models, cope with different model views and abstraction levels, and provide utilities for model transformation and code generation.

General adoption of such advanced tools implies a new practise in systems development. In addition to the activities and responsibilities defined in current model-based methodologies, someone must be responsible for 1) specifying the meta-models of the chosen PIM and PSM levels, 2) defining appropriate transformations between the PIMs and the PSMs, and 3) checking the consistency between models, both on different levels of abstraction and between viewpoints on the same level of abstraction. The new responsibilities call for new roles to be included in an MDA process. In the following, we outline the responsibilities of these new roles and specify their contributions to the systems development process.

2 Additional MDD Roles

2.1 Background

From the MDA Guide [3], one gets the strong impression that OMG's current vision of model-driven architecture is still open for some interpretation. Indeed, in relevant forums, much effort is used to discuss the meaning of central concepts such as "platform", "independent", "transformation" and "architecture". Despite these sometimes philosophical discussions, practical tools and techniques are emerging which assist software developers moving towards the MDD vision. When these tools are introduced into an organisation, one soon discovers that they assume new ways of working that requires new skills in the organisation. These skills build upon, but are not similar to, existing skills that one needs in traditional model-based development. The MDD community assumes that the investments an organisation has to make in order to get these new skills are outweighed by the returns in software productivity, maintenance and flexibility.

In the MDA Guide and elsewhere, it is explicitly stated that the OMG will not propose any standard methodology or process; it will only provide standardised building blocks for making domain- or organisation-specific methodologies. In [4], the assumption is that MDA will fit with most state-of-the-art methodologies, including agile software development, extreme programming and more heavy-weight processes like the Rational Unified Process. In the EU project MODA-TEL [5], efforts have been made to define a MDD methodology, the results of which are summarised in [6]. This is a general methodology that spans the identified phases of project management, preliminary preparation, detailed preparation, infrastructure setup, and project execution. However, in this paper we focus on the additional skills that are needed in an MDD project, and that may for instance be positioned in the methodology outlined in [6].

2.2 The meta team

We have grouped a number of skills into what we call "the meta team". These skills are needed to define modelling languages, domain and platform concepts, and to customise tools. In the following we detail these skills. Note that we use the term "platform" to denote any coherent and agreed-upon set of concepts, not limited to a computing platform such as J2EE or .Net. A PIM is independent of the concepts in the platform, whereas a PSM is dependent on them. Note also that when we refer to "the PIM level" or "the PSM level", we do not indicate that there is only one such level. Indeed, we appreciate the recursive structure of "PIMness"; a PSM may be a PIM with respect to another platform.

2.2.1 The domain expert

Domain experts are necessary irrespective of how the software is developed. The domain expert is a person with detailed understanding of the application domain and who is able to abstract and categorise the required concepts and their relationships in the domain. In MDD, the domain expert should also to be able to capture this knowledge in a domain model that can be used as a baseline for the PIM meta-model. In [7], this skill is referred to as ontological meta-modelling since it focuses on the meaning of things instead of the form, which linguistic meta-modelling does.

2.2.2 The platform expert

This expertise is also necessary irrespective of software development techniques and processes. Detailed knowledge about the platforms is needed in order to produce quality software that utilises the features of the platforms. In MDD, platform experts need to be able to specify the essential platform properties in a platform model that can be used as a baseline for the PSM meta-model. Again, this is an ontological meta-model, with the subject matter being the platform.

2.2.3 The language engineer

The language engineer creates customised modelling languages suited for a purpose. This may be to identify a UML subset or to design a new domain-specific language. In any case, in MDD, the language engineer needs to use a meta-meta-modelling framework, such as the MOF from the OMG or the Ecore in Eclipse, to define the language(s) in a uniform manner if the concepts in each language are to be related in a transformation process. The language engineer performs linguistic meta-modelling, creating languages that are able to express the concepts from the platform model(s) and the domain model(s). Thus, the language engineer creates the PIM and PSM meta-models. In addition, the language engineer may create mapping languages, i.e., languages used to annotate PIMs so that they can be the source of transformations to PSMs. The language engineer needs to have expert knowledge in language design to define the abstract syntax of the languages, and needs knowledge in semiotics to create the concrete syntax of the languages, especially if they are diagrammatic. If a language is related to, or a subset of, another language (such as the UML), the language engineer also needs intimate knowledge of that language definition.

Note that we assume existing modelling languages can be used without the involvement of a language engineer. This is especially important for special-purpose modelling languages that are designed to support different kinds of model analysis. For instance, a real-time modelling language may support schedulability analysis for an organisation without the involvement of a language engineer, unless this modelling language should be tailored to specific needs.

2.2.4 The transformation specifier

From the crucial role of model transformations in MDD, it follows that the skills of the transformation specifier are extremely vital for an MDD organisation. It is the responsibility of the transformation specifier to define the relationships between PIMs and PSMs. This can be done at the model level or at the meta-model level by relating the PIM meta-model to the PSM meta-model. In any case, the transformation specifier needs to know both source and target of the transformation, and needs to know the transformation language (e.g., the language which is emerging from the QVT-Merge proposal [8]). In addition to creating the transformation, the transformation specifier also defines what should be recorded from the transformation. These records are essential to support traceability and round-trip engineering. The transformation specifier is the one to bridge the worlds of the domain expert and platform expert, and must as such understand both worlds in sufficient depth to be able to relate the concepts. It is absolutely essential that the transformation utilises the features of the platform, which may be hard to obtain without intimate knowledge of the platform. Therefore, in many cases one person will play the roles of both the transformation specifier and the platform expert.

A transformation may not only be to take one PIM and turn that into a PSM. In many cases, several models are weaved together on the PIM level and then turned into a PSM. The ability to weave together models requires insight into the different domains of the models so that consistency criteria can be defined. In the terms of IEEE 1471 [9] that is used in the MDA Guide, model weaving may be regarded as view integration. This can be done at the meta-model level by defining consistency criteria between the different meta-models, or, in IEEE 1471 terms, between the different viewpoints. It remains to be seen whether the result of the QVT process is suitable to also address the issue of model weaving and viewpoint consistency. However, the transformation specifier needs to handle this issue irrespective of whether standardised mechanisms exist.

2.2.5 The method engineer

The final skill needed in the "meta group" is that of the method engineer. The responsibility of the method engineer is to identify and orchestrate the activities needed in the MDD software development project. The method engineer needs to identify the modelling artefacts that should be produced during the project, and relate them with appropriate transformations. Furthermore, the method engineer should customise the tools to support the individual tasks in the software development. Finally, the method engineer should organise the activities into a process and possibly customise a process support tool to support the enactment of the process. In [10], the authors define the notion of MDA Component as a collection of know-how about the individual tasks in a MDD process. Using this term, the responsibility of the method engineer is to identify and organise the MDA Components available in an organisation.

2.3 The project team

The project team does the application development. They base their work on the foundations of the meta team, and applies the MDD tools and techniques in each project. Their skills do not differ substantially from a regular development team; they need to use state-of-the-art tools to solve complex problems. In the following we briefly outline the skills that are pertinent to MDD.

2.3.1 The application designer

We group all aspects of application construction under this role. Requirements capture, architectural design, detailed design, coding and testing are all activities performed by the application designer. The difference in an MDD setting is that the designer should use the modelling languages provided by the language engineer when performing their activities. Moreover, the application designer should use the transformations provided by the transformation specifier instead of performing the transformations manually as in the traditional approaches. The application designer needs to understand the transformations that are used during application construction so that the consequences of different design choices are known. The use of (semi-) automatic transformations also assumes that the application designer uses one or more marking languages to mark the PIMs to become transformable.

2.3.2 The system analyser

Again, this role is part of traditional application development. System analysis may include analysis of the system's real-time behaviour, scaleability, maintainability, etc. The distinguishing feature in MDD is that the models are the primary artifacts, not the code, so system analysis can be done at the model level instead of at the system level. This means that the system analyser needs to be able to instrument the models in order to get them analysable.

2.3.3 The system tester

The system tester is the final role that requires additional skills in an MDD approach. As opposed to the traditional approaches, the models can also be tested in an MDD approach since model transformation steps are made explicit and can be verified. Note the difference between testing the models and generating tests that can be used for testing the system. Model-based test generation is already part of state-of-the-art approaches, whereas model testing is still largely unexplored. Model simulation is a technique to support model testing. Some modelling languages have accompanying simulation tools, but this is not the case for the UML. The skills needed for model testing is largely those needed for testing in general. The difference is that one tests the models, and for this the system tester needs to interpret the testing results in terms of modelling concepts instead of as system concepts. However, most of this should be supported by tools and most good traditional testers should be able to become good model testers.

3 MDD Tools

To support the activities outlined in the previous section, a number of useful tools can be identified. Below we list and briefly characterise the tools we have identified.

- Model editor. The obvious tool in a MDD approach is a model editor that supports creation and manipulation of models. The model editor should not care whether the models are application models or meta-models, a model is a model as far as the model editor is concerned. However, the model editor should perform conformance checks so that the modeller only can produce models that are according to the relevant meta-model. Preferably, a model editor should be able to be customised to support any modelling language that the language engineer produces.
- Model repository. The models need to be stored and managed; this is the responsibility of the model repository. The model repository should support management of models in any modelling language according to the meta-meta-modelling approach that is chosen, in addition to traditional repository services such as persistence and browsing.
- Model transformers. The model transformations should be encoded in a tool so that the transformations can be as automatic as possible. Note that total automation is in many cases not achievable or desired, human intervention is often needed in each case to decide some of the issues that the transformation addresses.
- Model analysis tools. Many kinds of analysis can be performed, and existing analysis tools can in most cases be used, perhaps tailored to deal with the chosen modelling approach.
- Model simulator. Finally, a model simulator is useful for certain tests. Many modelling languages already have simulator tools that can be used, but for UML this is still not available.

4 Conclusions

In this paper we have identified and discussed the different skills needed in MDD. We have also outlined some necessary tools needed to support the roles that an MDD approach prescribes.

If a manager in a software developing organisation reads this list, some concerns may arise, especially with respect to the meta team. Most software developing organisations look at method engineering as unproductive work that preferably someone else should do for them, and they should be able to pick up an appropriate methodology from a book or a course. In MDD, this is in general not the case since one of the basic ideas is to have specific tools and techniques (hereunder languages and transformations) for each application domain. However, most software developing organisations are not alone in their problem domain, and one can foresee standardised (de facto or more formal) techniques that are useful for many kinds of software development organisations.

Large organisations, however, may want to use a proprietary language to protect their investments from being directly transferable to competitors. Such organisations may want to have all roles in the meta team filled by internal resources.

Acknowledgements. The work reported in this paper is carried out in the context of MODELWARE, an EU IP-project in FP62003/IST/2.3.2.3.

References

1. Jacobson, I., G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Object Technology Series, ed. G. Booch, I. Jacobsen, and J. Rumbaugh. 1999: Addison-Wesley. 463.
2. Atkinson, C., et al., *Component-based Product Line Engineering with UML*. Component Software Series, ed. C. Szyperski. 2002: Addison-Wesley. 506.
3. Miller, J. and J. Mukerji, eds. *MDA Guide Version 1.0.1*. 2003, Object Management Group: Needham.
4. Kleppe, A., J. Warmer, and W. Bast, *MDA Explained*. Object Technology Series, ed. G. Booch, I. Jacobson, and J. Rumbaugh. 2003: Addison-Wesley. 170.
5. MODATEL, www.modatel.org.
6. Gavras, A., et al. *Towards an MDA-based development methodology*. in *First European Workshop on Software Architecture (EWSA 2004)*. 2004. St Andrews, Scotland: Springer Verlag.
7. Atkinson, C. and T. Kühne, *Model-Driven Development: A Metamodeling Foundation*. IEEE Software, 2003. **20**(5): p. 36-41.
8. QVT-Merge Group, *Revised submission for MOF 2.0 Query/Views/Transformations RFP*. 2004, Object Management Group.
9. IEEE, *Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems*. 2000. p. 23.
10. Bézivin, J., et al. *MDA Components: Challenges and Opportunities*. in *First International Workshop on Metamodelling for MDA*. 2003. York, UK.