Challenges in Model-Driven Software Engineering

Ragnhild Van Der Straeten[1], Tom Mens[2], and Stefan Van Baelen[3]

[1] Software and Systems Engineering Lab, Vrije Universiteit Brussel,
rvdstrae@vub.ac.be
[2] Service de Génie Logiciel, Université de Mons-Hainaut,
tom.mens@umh.ac.be
[3] DistriNet, Department of Computer Science, K.U.Leuven,
Stefan.VanBaelen@cs.kuleuven.be

Abstract. After more than a decade of research in Model-Driven Engineering (MDE), the state-of-the-art and the state-of-the-practice in MDE has significantly progressed. Therefore, during this workshop we raised the question of how to proceed next, and we identified a number of future challenges in the field of MDE. The objective of the workshop was to provide a forum for discussing the future of MDE research and practice. Seven presenters shared their vision on the future challenges in the field of MDE. Four breakout groups discussed scalability, consistency and co-evolution, formal foundations, and industrial adoption, respectively. These themes were identified as major categories of challenges by the participants. This report summarises the different presentations, the MDE challenges identified by the workshop participants, and the discussions of the breakout groups.

1   Introduction

MoDELS'08 is already the eleventh conference on UML, modelling and model-driven engineering (MDE). After more than a decade, research in MDE has significantly evolved and improved. Nevertheless, still a lot of fundamental and practical issues remain. A recent article by France and Rumpe [1] described some challenges to realise the MDE vision of software development. The existence of the article and a number of recent events show the real need for a forum to discuss future challenges in MDE. One such forum was the Dagstuhl Perspectives Workshop 08331 on *"Model Engineering of Complex Systems"*[4] which was organised at Dagstuhl, Germany on August 10-13, 2008.

The MoDELS workshop on *"Challenges in Model-Driven Software Engineering"* can be considered as a continuation of the discussions held during this Dagstuhl seminar. More specifically, our workshop addressed the question of how to proceed next, by identifying the future short-term and long-term challenges in MDE, and proposing ways to address these challenges. The main objective of the workshop was to provide a forum for people from academia and industry to:

---

[4] http://kathrin.dagstuhl.de/08331/

- identify obstacles to MDE research and practice;
- facilitate transfer of research ideas to industry;
- propose "revolutionary" novel ideas;
- proclaim important challenges that are either fundamental or pragmatic.

Our initiative was strengthened by the MoDELS'08 panel on the *"Past and Future of MDD"* that took place three days after our workshop. Partly based on questions raised during our workshop, panelists presented their vision of how MDD technologies and particular aspects of model-driven development research will evolve over the next 10 or more years.

This report summarises the presentations, the discussions held, and the challenges identified during the workshop. It can be used as a guide to researchers in software engineering to help them plan their future research, and to convince policy makers of the continuing and increasing importance of MDE research.

2 About the Workshop

The event was one of the most successful workshops co-located with MoDELS'08. There were 66 registered participants coming from 19 different countries, of which 5 non-European ones. From a gender perspective, about 21% of the participants were female.

In total, we received 15 submissions, of which 11 were accepted. Seven of those were invited to present their ideas during the workshop. A short abstract of each presentation is listed below. The articles of the three presentations marked with (*) are included in this Workshop Reader. For the other articles, as well as for the accepted submissions that were not presented during the workshop, please consult the electronic workshop proceedings [2].

Parastoo Mohagheghi (*) *MDE Adoption in Industry: Challenges and Success Criteria* [3]
MDE has been promoted to solve one of the main problems faced by software industry today: coping with the complexity of software development by raising the abstraction level and introducing more automation in the process. The promises are many. Among them are: improved software quality by increased traceability between artifacts, early defect detection, reducing manual and error-prone work and including knowledge in generators. However, to be successfully adopted by industry, MDE must be supported by a rich ecosystem of stable, compatible and standardised tools. It should also not introduce more complexity than it removes. The presentation reported on the authors' experience in adoption of MDE in industrial and research projects. It also discussed the areas in which MDE has potential for success and what the key success criteria are.

Dimitrios Kolovos (*) *Scalability: The Holy Grail of MDE* [4]
Scalability is a desirable property in MDE. The current focus of research in MDE is on declarative languages for model management, and scalable mechanisms for persisting models. The presenter claimed that, instead, modularity and encapsulation in modelling languages should be the main focus. This

claim was justified by demonstrating how those two principles apply to a related domain, namely code development, where the issue of scalability has been addressed to a much greater extent than in MDE.

Ernesto Posse *A foundation for MDE* [5]

MDE is still lacking adoption by developers. To live up to its full potential MDE must rest on a solid foundation. Therefore, one of the main challenges facing MDE today is the establishment of such a foundation. In this presentation, UML-RT was used as a case study to illustrate what can be achieved, what is missing and what kind of issues must be addressed by a successful approach to MDE.

Antonio Vallecillo (*) *Behaviour, Time and Viewpoint Consistency: Three Challenges for MDE* [6]

Three problems that MDE should tackle in order to be useful in industrial environments were outlined in this presentation. Firstly, the specification of the behavioural semantics of meta-models so that different kinds of analysis can be conducted, e.g., simulation, validation and model checking. A second challenge is the support of the notion of time in these behavioural descriptions, to be able to conduct, e.g., realistic performance and reliability analysis of industrial systems. As a third challenge, not only the accidental complexity involved in building software systems needs to be tackled, but their essential complexity should be addressed too. To achieve this, more effective use needs to be made of independent but complementary viewpoints to model large-scale systems, and correspondences between them to reason about the consistency of the global specifications need to be specified.

Dennis Wagelaar *Challenges in bootstrapping a model-driven way of software development* [7]

According to the presenter, current MDE technologies are often demonstrated using well-known scenarios that consider the MDE infrastructure to be already in place. If developers need to develop their own infrastructure because existing tools are insufficient, they will encounter a number of challenges. Generally, developers cannot just implement all their model transformations and other MDE infrastructure immediately, because it simply takes too long before they get usable results. An incremental approach to putting model-driven development into place gives you the necessary "breakpoints", but poses extra challenges with regard to the MDE technologies used. Some of these challenges are: how to bootstrap a step-wise refinement chain of model transformations, how to bootstrap the modelling language usage, how to fit in round-trip engineering, and what are the useful properties for a model transformation tool.

Robert Clarisó *UML/OCL Verification in Practice* [8]

One of the promises of model-driven development is the ability to identify defects early, at the level of models, which helps to reduce development costs and improve software quality. However, there is an emerging need for "lightweight" model verification techniques that are usable in practice, i.e., able to find and notify defects in realistic models without requiring a strong verification background or extensive model annotations. Some promising ap-

proaches revolve around the satisfiability property of a model, i.e., deciding whether it is possible to create a well-formed instantiation of the model. Existing solutions in the UML/OCL context were discussed. The presenter claimed that this problem has not yet been satisfactorily addressed.

Jordi Cabot *Improving Requirements Specifications in Model-Driven Development Processes* [9]

Understanding the organisational context and rationales that lead up to system requirements helps to analyse the stakeholders' interests and how they might be addressed or compromised by the different design alternatives. These aspects are very important for the ongoing success of the system but are not considered by current MDE methods. The presenter argued for the necessity of extending existing methods with improved requirement techniques based on goal-oriented techniques for the analysis and specification of the organisation context, and discussed the benefits and challenges of such integration.

3 Identified Challenges

During the plenary session that took place after the presentations, all workshop participants identified some of the major challenges in MDE. Those challenges are enumerated below. It is important to note that this list is inevitably incomplete. Also, the order in which we present the challenges here is of no particular importance.

Model quality. We need to deal with quality aspects in modelling and model-driven engineering. This gives rise to a number of open questions:
  - How can we define model quality?
  - How can we assure, predict, measure, improve, control, manage quality?
  - How can we reconcile conflicting quality aspects?

These and many related challenges are very important, and have been discussed in more detail in the MoDELS'08 workshop on *"Quality in Modelling"*. There is also a recent book that addresses these topics [10].

Run-time models. In model-driven software engineering focus has been primarily on using models at analysis, design, implementation, and deployment stages of development. The use of models during run-time extends the use of modelling techniques beyond the design and implementation phases of development and introduces a number of challenges:
  - How can we represent dynamic behaviour?
  - What should a run-time model look like? How can we use and maintain such models at run-time?
  - How do they relate to "static" models?
  - What are good approaches to follow when developing run-time models?
  - What are the differences, advantages and shortcomings of model interpretation, model simulation/execution and code generation techniques?

These and many related challenges have been discussed during the MoDELS'08 workshop on *"Models@run.time"*.

Requirements modelling. Research related to requirements is underrepresented in the MDE community. Nevertheless a number of important challenges remain to be tackled:
- How can we model requirements?
- How can we bridge the gap between informal (textual) requirement specifications and formal requirement models?
- How can we integrate the activity of requirement specifications into traditional modelling?
- How can we achieve traceability between requirement specifications and design models?

Standards and benchmarks. There is a need for standards and benchmarks to compare different tools and approaches. Benchmarks provide an excellent resource to measure progress and the significance of a contribution. However, widely accepted benchmarks do not exist yet. This leads to the following open questions:
- How to design and develop benchmarks that facilitate comparison between tools and approaches?
- Which standards are needed to facilitate interoperability between tools?
- How can we obtain and share common data (models, model transformations, case studies)?

Modelling languages. Models cannot be developed without precise modelling languages. Modelling languages are one of the main themes of the MoDELS conferences. Although the state of research in modelling languages has significantly progressed, a number of open questions remain:
- Which languages, methods, principles and tools are necessary to design precise meta-models?
- How can we support better modularity in MDE?
- How to describe design *pragmatics* (as opposed to syntax and semantics)?
- How can we allow for and deal with multi-models and multi-formalism modelling?

Domain-specific modelling. Domain-specific modelling languages are designed to provide precise abstractions of domain-specific constructs. Models for complex systems encompass several domains. Capturing all important aspects of such a complex system requires developing multiple models using different DSMLs and introduces many challenges.
- How to develop and integrate models using different domain-specific modelling languages?
- What process and which tools should we use to analyse, design, develop, verify and validate domain-specific models and languages?
- How can we increase reuse across different domain-specific modelling languages?
- How can we facilitate code generation from domain-specific modelling languages?
- What is the trade-of between general-purpose modelling languages, tools, techniques and domain-specific ones? Do we get a higher degree of specialisation, higher expressiveness, and higher potential for code generation, model execution and formal reasoning?

Panelists of the MoDELS'08 panel on *"Addressing the Challenges of Multi-Modelling for Domain-Specific Modelling Languages"* have commented on these and related challenges.

Empirical analysis. In the context of MDE, the topic of empirical analysis raises a number of interesting challenges:

- What are the main obstacles and potential remedies when performing empirical studies of MDE?
- What are the strengths and weaknesses of evaluating MDE activities, tools and techniques in laboratory and field settings, as well as industrial case studies?
- How should we deal with the unavoidable trade-offs between realism and control?
- How can we obtain adequate estimations for an MDE process and which measurements are relevant for MDE?

These and related challenges have been addressed and discussed at the MoDELS'08 workshop on *"Empirical Studies in Model Driven Engineering"*.

Model verification and validation. As in any software development approach, verification and validation are essential for MDE. In the context of MDE, it imposes a number of additional challenges:

- How can we verify, validate, debug, and test the models and the code generated from those models?
- How can we automatically generate test cases from models?
- How can we provide *incremental* support for verification and validation?
- How can we deal with partially incomplete and inconsistent models?
- How can we support formal verification of models?
- How can we address validation and verification in a multi-model world?

Process support. Model-driven engineering encompasses many more activities than merely modelling. One important aspect that is often overlooked by the scientific community is *process support*. This gives rise to a number of essential questions:

- Which processes should be used for MDE?
- How should existing processes embrace MDE?
- How should we teach and educate people in adopting MDE technology?
- How can we incorporate the MDE environment in the MDE process?

Fuzzy modelling. Models are not always complete and sometimes inconsistencies need to be tolerated. This gives rise to specific questions like:

- How can we deal with modelling in presence of uncertainty?
- How can we deal with models that are imperfect or ambiguous?
- How can we reason about models that are incomplete or inconsistent?
- How can we cope with imprecision of models?

Industrial adoption. This topic will be discussed in section 4.1.

Formal foundations. This topic will be discussed in section 4.2.

Scaleability issues. This topic will be discussed in section 4.3.

Model consistency and co-evolution. This topic will be discussed in section 4.4.

As a general kind of meta-challenge, it was suggested by one of the participants that we need to be aware more of relevant past research (possibly in other software engineering domains), rather than trying to reinvent the wheel.

4 Discussion of Breakout Groups

Because it was not possible to explore all of the above challenges in detail during the workshop, we decided to break out into 4 different groups, each one focusing on a particular set of challenges that were considered to be important by the majority of participants.

4.1 Industrial Adoption

The "engineering" aspect of model-driven engineering implies that research in MDE is useless without having industrial adoption. The MDE community could benefit a lot from concrete industrial use cases, both positive and negative ones.

From the positive side, it would be good to learn which companies have successfully adopted MDE technology, and what was the reason of this success: Which tools, techniques and processes have been used, and what was the added value brought by MDE? Several participants mentioned examples of such success stories. For example, in the automotive industry, the use of MDE technology is standard practice. The same appears to be true for real-time and embedded systems. Also in the area of web application development there are various approaches that support MDE (e.g., AndroMDA). Finally, there were examples of the use of MDE in the telecommunications and insurance domains.

The opposite question was also discussed. Can we find failures of the use of MDE in industry, and the reasons underlying these failures? Similarly, can we find reasons why some software companies are not using MDE? Some interesting points were raised when discussing about these questions. First of all, there is a significant technological threshold and learning curve before you can actually use MDE in industry. Therefore, using MDE technology may not be cost effective for many industrial application scenarios. The argument was also made that, although some companies do not use MDE, they do use models for communication. Finally, while some companies may be interested in using MDE technology, it may not be possible if they still have a large legacy code base available that has been around for decades, and has been developed in "old" technology (e.g, COBOL code) that may be too costly or too hard to migrate.

One thing that appeared to be common to all discussed industrial use cases was the use of domain-specific modelling languages. This seems to indicate that MDE works well for specific problems in specific domains, and that the use of a *universal* modelling language (e.g., UML) may possibly not work well in an industrial setting.

We also discussed how MDE can bring value to industry. In some of the industrial cases discussed, MDE was used because of its ability to formally specify and analyse (part of) the system. This leads to a reduction in ambiguity and improved quality. Other potential advantages are cost reduction, productivity improvement, easier maintenance, and detection of problems and inconsistencies in the software system early in the life cycle. Of course, it is essential to keep in mind that any of these potential advantages should not be detrimental to the performance of the software system to be produced.

The main issue with the above is that we need to convince industry that there is actually added value of MDE, and we should come up with a deployment model to enable the adoption of MDE technology in industry. The only way this can be done is by performing convincing empirical case studies of the use of MDE in industry. Obviously, to be able to perform such studies, we need to be able to obtain detailed data about MDE practice from industry itself. In practice, it turns out to be very difficult to obtain industrial data, and to transfer technology and research results from academia to industry. Participants of the breakout group basically agreed that the only viable way to achieve this is by direct contact between the research community and industry. One way to establish such contact is via exchange programmes in which PhD students spend a couple of months in a company to understand the process used and the particular activities that are amenable to automation via MDE technology, as well as to raise awareness of industry in the benefits of MDE. Other possibilities are the use of dedicated industrial education and training programmes.

4.2   Formal Foundation

Verification and validation is an important research theme in the MDE community. To be able to verify and validate models and model transformations, a formal foundation is a necessity. The first challenge identified by the participants of this breakout group was to integrate formal verification tools into modelling environments. This needs to be done in such a way that the user of the modelling environment does not need to have expertise in the different formalisms and techniques used for verification. The feedback of these verification tools needs to be formulated in a language or formalism that the end user of the environment is familiar with.

To realise this smooth integration, the participants of the breakout group agreed that transformations from modelling languages to formal verification and analysis models need to be defined. The definition of transformations triggers several interesting challenges. How to define such transformations? How to prove correctness of model transformations, especially if the source models are informal? And how to proof that the transformation is correct? It is possible that the transformation of an informal model to a formal model is correct by construction, since the main goal of such semantic mapping is to define a precise semantic meaning for the concepts of the informal model. All participants of the breakout group agreed that first of all the notion of correctness needs to be defined, because many variations of correctness definitions exist in state-of-the-art literature. Once models are transformed into a certain formalism and verification of properties has been executed in this formalism, feedback needs to be given to the end user and incorporated into the source model. The question arises on how to reinterpret these results in the source models and tools.

There is a lot of existing work on using formalisms to support model-driven engineering. Examples are graph transformation theory, algebraic specifications, model checking, logic-based approaches and SAT solvers, and category theory.

In the programming language area, operational, denotational and axiomatic semantics exist. These different approaches are useful for investigating different kinds of properties. The participants also recognised that different semantics and formalisms may be necessary at different phases in the development life cycle, and at at different levels of abstraction, since a single formalism may not fit all the models describing various aspects of a complex system. This gives rise to an interesting challenge: how to relate these levels and how to define the relationships between them. The participants posed the question whether it is necessary to define relations between the different formalisms. As a complex system is gradually being modelled using a multitude of often large models, and regularly extended and adapted, incremental verification and validation and scalability of the verification and validation tools become key challenges for MDE.

Precisely defining domain-specific modelling languages was another discussion topic. This raises the question how to help developers design good modelling languages that guarantee useful properties to the users of these languages. Reusability was recognised as a key issue in modelling language design. In analogy with design patterns, the participants propose to identify and define patterns and anti-patterns for designing modelling languages. The main challenge that was identified is to define domain-specific modelling languages that enable and enforce model correctness by construction.

All participants agreed that, despite the multitude of existing formalisms and experiments to use them in MDE, a lot of research still needs to be done. This is especially true for tool support for verification and validation, as well as support for defining well-designed modelling languages. A goal-driven approach for MDE was suggested, by focusing on the question of what needs to be the added value of the languages, techniques, and tools?

4.3   Scaleability

Scaleability is a general problem in software engineering. Many software engineering research areas are struggling to cope with scaleability issues, and a large research effort has already been spent to develop solutions for overcoming scaleability problems. The MDE community must therefore focus on (1) the kind of scalability issues that are intrinsic for MDE; (2) elements about MDE do not scale well and the underlying reasons thereof; and (3) specific scalability problems for MDE that cannot be addressed by existing solutions from other software engineering domains.

Concerning the intrinsic type of scalability needed for MDE, one of the main problems is that MDE has to be able to cope with very large models in order to model systems of systems and Ultra-Large-Scale (ULS) systems. These models have to be constructed, transformed, merged, and used as a base for code generation. So one could try to develop solutions for optimising these activities in order to use them adequately on large models. However, often solutions for one type of activity can be rather different than those necessary for other types of activities. The question arises whether generic optimisation solutions can be developed for MDE activities. In addition, one must be aware that the time to

load huge models is often greater than the time needed for checking, merging or transforming such models.

Elements that can cause scalability problems in an MDE approach are, among others, multi-site collaborative development, complexity of algorithms manipulating models, computer resources needed to manage huge models, and technical limitations of the used notations and tools (concerning support for modularity, concurrent access, distribution, etc.). In addition, the accidental complexity of underlying MDE technology should be reduced.

Mechanisms and techniques from other software engineering domains could be useful for solving MDE scaleability issues. From the programming community, techniques such as modular engineering principles, incremental processing, caches, and indices could be beneficial for MDE. Further solutions can come from logic inference engines for model checking, and high performance computing for optimisation techniques. The participants assessed that there are known solutions to all the problems they thought of, however, the issue is to generalise them for MDE. In addition, the design of modelling languages seems not always to respect known scaling problems in concrete languages.

4.4 Model Evolution and Inconsistency Management

Models do not appear after a big bang, but are often developed by different persons in a distributed setting using different modelling languages. Such multi-user distributed setting, combined with the usage of different modelling languages to model a system, can cause inconsistencies in and between models. Models evolve and so do their meta-models. The major challenge is to assess the impact of change of a model or meta-model on the other models and meta-models. The challenge increases if models are distributed. The participants of this breakout group propose – as a small step towards a solution – to categorise the different change types and the possible ways to resolve the inconsistencies introduced by the changes.

Models are built using a variety of domain-specific modelling languages. The question arises how to develop DSMLs can be *efficiently* extended, adapted or customised. The term *efficiency* is strongly related to traditional quality measures. More effort should go to a DSML process. One possible solution would be to have rapid prototyping for building DSMLs. Prototyping has the advantage of giving continuous, incremental feedback. In the context of evolution the question arises how a DSML evolves from version $n - 1$ to version n and what happens with the existing models adhering to the DSML version $n - 1$?

Other challenges that were identified are: how to deal with long-lived models and legacy models? How to maintain models? How to avoid model erosion? How to support long-lived software intensive systems that have been generated using MDE? The participants stated that more effort should go to a model-driven development process.

Once inconsistencies are identified in or between models, the question arises how to deal with these inconsistencies. Model-driven development environments need built-in support for inconsistency handling and resolution. Built-in support

for inconsistency detection and handling is also needed in versioning tools. In a model-driven software development process, handling inconsistencies at model level will also affect the (generated) code. As such, an important research question is how to support model-code synchronisation and round-trip engineering.

Formalisms and techniques to detect, handle and resolve inconsistencies can be based on formal verification techniques used in other software engineering domains such as programming language engineering, but also on formalisms and techniques used in database engineering and artificial intelligence.

The participants also discussed the question how collaboration in this field can be improved. They suggest two possible approaches. First, the development of an ontology for the consistency/evolution area, and second, a survey that unifies the different existing perspectives.

5 Past and Future of MDD

As one of the outcomes of the workshop, besides the current workshop report, each breakout group prepared a single question that was passed to the panelists of the MoDELS'08 panel on the *"Past and Future of MDD"*. These questions were:

– How can we better understand the software process and activities that companies use and improve them with MDE technology?
– Can incremental model verification and model checking contribute to successful adoption of MDE?
– Are there scalability problems that are specific to MDE, and how can we address them?
– How can we deal with co-evolution of models, meta-models and transformations in a distributed multi-developer environment?

6 Conclusions

During the workshop, a large number of challenges for MDE have been identified, covering a broad range of topics that are important for the successful application of MDE in practice. The organisers hope that the workshop results help to identify an MDE research agenda, to define the roadmap for future MDE research, and to inspire researchers for tackling important problems and develop novel and adequate solutions.

Acknowledgements

References

1. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: FOSE '07: 2007 Future of Software Engineering, Washington, DC, USA, IEEE Computer Society (2007) 37–54
2. Van Baelen, S., Van Der Straeten, R., Mens, T., eds.: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering. VUB, UMH, K.U.Leuven (2008)
3. Mohagheghi, P., Fernandez, M., Martell, J., Fritzsche, M., Giliani, W.: MDE adoption in industry: Challenges and success criteria. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering. (2008) 5–9
4. Kolovos, D.S., Paige, R.F., Polack, F.A.: Scalability: The holy grail of model driven engineering. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering. (2008) 10–14
5. Posse, E., Dingel, J.: A foundation for MDE. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering. (2008) 15–19
6. Rivera, J.E., Romero, J.R., Vallecillo, A.: Behavior, time and viewpoint consistency: Three challenges for MDE. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering. (2008) 20–24
7. Wagelaar, D.: Challenges in bootstrapping a model-driven way of software development. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering. (2008) 25–30
8. Cabot, J., Clarisó, R.: Uml/ocl verification in practice. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering. (2008) 31–35
9. Cabot, J., Yu, E.: Improving requirements specifications in model-driven development processes. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering. (2008) 36–40
10. Rech, J., Bunse, C., eds.: Model-Driven Software Development: Integrating Quality Assurance. Information Science Reference (2008)

MDE Adoption in Industry: Challenges and Success Criteria

Parastoo Mohagheghi[1], Miguel A. Fernandez[2], Juan A. Martell[2],
Mathias Fritzsche[3] and Wasif Gilani[3]

[1] SINTEF, P.O.Box 124-Blindern, N-0314 Oslo, Norway
`parastoo.mohagheghi@sintef.no`
[2] Telefónica Research & Development, Valladolid, Spain
`mafg@tid.es, jamartell@gfi-info.com`
[3] SAP Research CEC Belfast, United Kingdom
`{mathias.fritzsche, wasif.gilani}@sap.com`

Abstract. Model-Driven Engineering has been promoted for some time as the solution for the main problem software industry is facing, i.e. complexity of software development, by raising the abstraction level and introducing more automation in the process. The promises are many; among them improved software quality by increased traceability between artifacts, early defect detection, reducing manual and error-prone work and including knowledge in generators. However, in our opinion MDE is still in the early adoption phase and to be successfully adopted by industry, it must prove its superiority over other development paradigms and be supported by a rich ecosystem of stable, compatible and standardized tools. It should also not introduce more complexity than it removes. The subject of this paper is the challenges in MDE adoption from our experience of using MDE in real and research projects, where MDE has potential for success and what the key success criteria are.

Keywords: Model-driven engineering, challenges, domain-specific modeling, performance engineering, traceability.

1 Introduction

Today's software systems are complex in nature; the size has been growing because of the increased functionality, heterogeneity is also becoming a bigger concern as systems are built from several systems or include legacy code, systems are distributed over multiple sites and there are new requirements such as dynamicity and autonomy (self-* properties, for example self-healing). Handling each of these challenges requires specific approaches which often include domain-specific knowledge and solutions. However, based on the experience gained from multiple domains and projects, some solutions may be identified as beneficial to complex software development in general.

Model-Driven Engineering (MDE) is an approach built upon many of the successful techniques applied in software engineering: It can be characterized by: a) raising the abstraction level by hiding platform-specific details ; b) taking advantage

of models in all the phases of software development to improve understanding; c) developing domain-specific languages and frameworks to achieve domain appropriateness; and d) taking advantage of transformations to automate repetitive work and improve software quality [6]. These are all techniques useful for complex system development and therefore one may expect rapid adoption of the paradigm by industry. So far, we cannot see such wide adoption, as also confirmed by a review of industrial experiences presented in [7]. In fact, and based on the model of technology adoption life cycle presented in [8], we think that MDE is still in the early adoption stage. Early adopters do not rely on well-established references in making their buying decisions, preferring instead to rely on their own intuition and vision. However, they are keys to opening up any high-tech market segment. To be accepted by the majority, the industry must gain confidence on the promises of MDE and have access to proper tools and experts.

There are many challenges in complex system development, such as managing requirements, which MDE is not a direct answer to, but it might facilitate their handling by providing mechanisms for easy traceability between artifacts. There are also challenges such as dealing with legacy code that may be difficult to handle and must be either worked around or, better yet, integrated into the MDE approaches. But there are challenges that MDE may provide an answer to based on the MDE core practices (such as extensive modeling and the usage of transformations) as discussed in [6].

The European research projects MODELWARE[1] and its continuation MODELPLEX[2] have focused on MDE approaches and tools with the goal of making them suitable for complex system development. Some of the companies involved in these projects have experience from applying MDE in real projects while others think that MDE is not yet mature enough to be taken from research projects to industry production. This paper therefore elaborates on where we can expect added value from MDE and what the barriers are from experiences gained in the context of these projects. In the remainder of this paper we discuss industry expectations and experience in Sections 2 and 3 and conclude our discussion in Section 4.

2 SAP Experience

SAP has already started working towards applying MDE concepts, and currently employs models in various stages of business application development. The tool called NetWeaver BPM within the Composition Environment [10] is one example where MDE concepts are applied for efficient development of Composite Applications. Composite Applications are self-contained applications that combine loosely coupled services (including third party services) with their own business logic, and thereby provide user centric front-end processes that transcend functional boundaries, and are completely independent from the underlying architecture, implementation and software lifecycle. With Composition Environment even the non-technical users, such as business domain experts, consultants, etc., having no

---

[1] http://www.modelware-ist.org/
[2] http://www.modelplex-ist.org/

programming skills, are able to model and deploy customized applications suited to their specific business requirements.

Based on our experience [5] with the currently employed MDE tools for business processes, such as the Composition Environment, we identified the general need of supporting non-technical users with regards to non-functional requirements, such as the impact of their design decisions on performance, etc. Within the context of performance engineering, for instance, such a support means guidance towards better design / configuration that actually meets the timelines, and optimized resource mapping against each activity in the business process.

We implemented such performance related decision support as an extension of MDE. By implementing this extension, named Model-Driven Performance Engineering (MDPE), we realized the need for supporting requirements with respect to non-functional aspects, especially performance. The implementation of MDPE heavily uses the MDE concepts such as meta-modeling, transformations, model weaving and mega-modeling. For instance, ten different meta-modeling languages are employed in order to make the process usable for a number of domain-specific modeling languages. During the implementation of MDPE, we recognized that the application of MDE concepts enabled us to focus on the creative tasks of development rather than repetitive coding. For instance, code generation for our meta-models saved us significant development effort. The only place where a significant amount of coding effort was required was for the integration of MDPE into the existing tool infrastructure.

Meta-model extension is the generally employed technique for model annotations, such as done with profiles in the case of UML [3]. However, this is not applicable while dealing with the proprietary models. The application of model weaving enabled us a high degree of flexibility as we are able to annotate any kind of proprietary model with the help of a generic editor [3]. Higher-order transformations are used to enable traceability in our approach [4]. Additionally, mega-modeling enables us to locate our model artifacts, such as the tracing models related to the models in our transformation chain [1].

As for the challenges, we experienced that MDE concepts are on the one hand very systematic and efficient, but on the other hand also difficult to understand for developers as they require quite a high level of abstraction and training. Also, the MDE tool support is sometimes not mature enough. Especially the available tooling to define model transformation chains lacks capabilities of modern IDEs (Integrated Development Environments), which could decrease the development time for model transformations significantly.

Concluding, based on the experiences gained with the development of MDPE, we are optimistic regarding the capabilities of MDE in case the tool support improves, and the MDE community meets the challenges associated with the MDE process, such as providing support for dealing with non-functional aspects of system development.

3 Telefónica Experience

In [2], we have discussed the experience of Telefónica in moving from a code-centric to a model-centric software development. Earlier efforts in modeling failed due to the complexity of UML, the lack of proper tools and the inability to maintain models in synch with code, among other issues. Due to the above problems with UML, we decided to develop our own programming tools and frameworks addressing the problem domain. But without any industry standards to rely on, this approach had no future in the long term and was also difficult to use for non-technical staff, such as telecom domain experts, as it did not have the required abstraction level.

This was an experience from eight years ago, but not so many things seem to have fundamentally changed. What we look for is a domain-specific modeling (DSM) language integrated in a development environment that will permit the modeling of our basic domain concepts, such as interfaces, devices, networks, protocols and services. We also emphasize adhering to current industry standards in the domain, since we now look for a domain-specific solution, not a company-wide solution. Other requirements are: a) the ability to model in multiple abstraction levels, hiding details as desired; b) the integration of model verification tools based on OCL or other constraint languages and c) the composition / weaving of the models at run time to reflect the changes in the network's operational status. Some of these approaches are further discussed in [9].

In the road toward these objectives we foresee numerous challenges. First of all, the UML standard has evolved but, with this evolution, the syntax has become even more complex and the necessary supporting mechanisms and tools for dealing with this added complexity are not yet available. Even something as conceptually simple as exporting a UML diagram from one tool to another has not been accomplished yet with ease. On the other hand, developing a DSM solution requires high skills related to meta-modeling and tool development. Also a big concern with Domain-Specific Languages (DSLs) is getting the people in that domain to agree upon a standard syntax. Another challenge is having that DSL interact properly with anything outside of its domain, having a different underlying syntax to that of other languages.

Model synchronization (for example applying multiple profiles to a source model) and roundtrip engineering are yet to be addressed successfully and mechanisms for dealing with very large and complex models, such as hierarchical models, traceability and model management in general are also in an inception phase right now, at least regarding to the aspect of tool support. The evolution of meta-models, in a business as dynamic as ours, is also a big concern and tools have much to improve in order to adequately manage variability at meta-model level and not only at model level. All these features are important to make a full-fledged MDE process work in complex, real-life projects.

Another challenge for organizations wanting to get started in MDE, closely related with the previous idea of managing all these artifacts, is that they may end up dealing with more complexity than anticipated at first. From our experience in the field we have gotten the impression that, if not adequately managed, the development of complex systems with MDE gets treated with more complexity. The underlying problem here is: are the techniques for handling complexity in danger of making the software engineering process itself too complex? To adequately address complexity

we have to substitute it for something simpler not for something different but equally complex.

It is our opinion also that there are some basic milestones a new technology has to go through for it to be considered mainstream. To start with, we need a proper context for it to flourish and be nurtured in. The fabric of this context is made of the proper professionals with the proper knowledge and expertise and supporting material which helps in turn to create these professionals. We are seeing shortcomings in this regard so far. The community is in fact there and growing but perhaps it is not reaching critical mass yet. We also see a gap between the academic and industrial worlds that needs to be bridged. In the past, new paradigms have been promoted by well-known professionals lending credibility and raising interest in the new approach. This has to be accompanied by the development of high-quality literature, tutorials and proper material to draw new professionals in.

The main question that an organization has to ask itself is "do I really need MDE?" The second question relates with its ability to adapt its processes to the ones needed from an MDE point of view (partially discussed also in [2]), adapt their staff to new ways of looking at problems and create new layers of software development supporting all the aspects MDE has to offer. Companies may be reluctant to change either their structure or part of it.

To conclude, it is worth mentioning that, apart from software factories for product line engineering (PLE), we have not seen clear evidence of other good candidates for MDE to be fully applied to, as a complete lifecycle solution. We feel that it can be partially applied, though, to some other scenarios like large-scale integration of heterogeneous systems, as it is the case with Telefónica's Operating Support Systems (OSS), area in which we hope to start making some progress in the short term with Model-Based Testing (MBT).

4 Conclusions

Probably most companies cannot take the risk of adopting MDE end-to-end in large-scale projects from scratch. They should look for areas of improvement and take the approach incrementally and integrated with their own development environment. This is also the best way to train people. There is an initial high cost related to developing or adopting tools and transformations. MDE is a long-term investment and needs customization of environment, tools and processes, and training. For companies that have a product line, MDE can pay off since this cost is amortized over several projects. For one-of–a-kind projects this will not pay in most cases. Despite differences in domain and the type of systems developed in the two companies, there are common challenges as described in this paper. The most important one is the complexity of developing an MDE environment tailored to the company needs. This environment requires:

- Developing proper languages for communication between technical and non-technical experts and for modeling various aspects. One of the successes of MDE lies in bridging the gap between technical and non-technical experts. The major challenge here is to have the required language engineering expertise since

creating own profiles or meta-models are difficult and for complex systems we probably need several languages. Hence more domain-specific meta-models and profiles are needed that are supported by tools and may be reused. The current tools for developing meta-models and editors are not user friendly, the learning curve is steep and the documentation and support is not satisfactory.

- Several tools are required for modeling, model-to-model and model-to-text transformation, verification and simulation, and other tools to store, reuse and compose models. There is no tool chain at the moment and companies must integrate several tools and perform adaptation themselves.

Both of the above requirements put a high burden on companies that traditionally used third-party tools for modeling and performed programming by hand. Training is another major challenge here. We see advantages in gradual introduction and support by management, as well as in the creation of teams of experts that can give support and create the necessary tools for MDE adoption in the whole company.

References

1. Barbero, F. Jouault, J. Bezivin: Model Driven Management of Complex Systems: Implementing the Macroscope's Vision. In: 15th ECBS'08, IEEE Press, pp. 277--286 (2008)
2. Fernandez, M.: From Code to Models: Past, Present and Future of MDE Adoption in Telefónica. In: 3rd Europen Workshop From Code Centric to Model Centric Software Engineering: Practices, Implications and Return on Investment (C2M), co-located with ECMDA 2008, pp. 41—51 (2008)
3. Fritzsche M., Johannes J., et al: Systematic Usage of Embedded Modelling Languages in Model Transformation Chains. Accepted at the Software Language Engineering Conference, SLE'08 (2008)
4. Fritzsche, M., Johannes, J., Zschaler, S., Zherebtsov, A., Terekhov, A.: Application of Tracing Techniques in Model-Driven Performance Engineering. In: ECMDA-FA 4th Workshop on Traceability (2008)
5. Fritzsche, M., Gilani, W., Fritzsche, C., Spence, I.T.A, Kilpatrick, P., Brown, T.J.: Towards utilizing Model-Driven Engineering of Composite Applications for Business Performance Analysis. In: ECMDA-FA, LNCS 5095, pp. 369—380 (2008)
6. Mohagheghi, P.: Evaluating Software Development Methodologies based on their Practices and Promises. In Proc. Somet'08: New Trends in Software Methodologies, Tools and Techniques, IOS Press, ISSN 0922-6389, pp. 14—35 (2008)
7. Mohagheghi, P., Dehlen, V.: Where is the Proof? A Review of Experiences from Applying MDE in Industry. In ECMDA-FA 2008, LNCS 5095, Springer, pp. 432—443 (2008)
8. Moore, G.A.: Crossing the chasm: Marketing and Selling High-Tech Products to Mainstream Customers. HarperBusiness Essentials, 2nd edition (2002)
9. Pickering B., Fernandez M., Castillo A., Mengusoglu E.: A Domain-Specific Approach for Autonomic Network Management. In: 3rd IEEE MACE Workshop (2008)
10. Snabe, J.H., Rosenber, A., Møller, C., Scavillo, M.: Business Process Management: The SAP Roadmap, SAP Press, ISBN 978-1-59229-231-8 (2008)

The Grand Challenge of Scalability for Model Driven Engineering

Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack

Department of Computer Science, University of York, UK.
(dkolovos,paige,fiona)@cs.york.ac.uk

Abstract. Scalability is particularly important for the adoption of Model Driven Engineering (MDE) in an industrial context. The current focus of research in MDE is on declarative languages for model management, and scalable mechanisms for persisting models (e.g., using databases). In this paper we claim that, instead, modularity and encapsulation in modelling languages should be the main focus. We justify this claim by demonstrating how these two principles apply to a related domain – code development – where the issue of scalability has been addressed to a much greater extent than in MDE.

1   Introduction

The adoption of MDE technologies in an industrial context involves significant benefits but also substantial risks. Benefits in terms of increased productivity, quality and reuse are easily foreseeable. On the other hand, the most important concerns raised of MDE are those of scalability [1], the cost of introducing MDE technologies to the development process (training, learning curve) and longevity of MDE tools and languages. To our perception, the latter two concerns (cost of induction and longevity) are not preventive for the adoption of MDE; however scalability is what is holding back a number of potential adopters.

2   Scalability in MDE

Large companies typically develop complex systems, which require proportionally large and complex models that form the basis of representation and reasoning. Moreover, development is typically carried out in a distributed context and involves many developers with different roles and responsibilities. In this context, typical exploratory questions from industrial parties interested in adopting MDE include the following:

1. *In our company we have huge models, of the order of tens of thousands of model elements. Can your tool/language support such models?*
2. *I would like to use model transformation. However, when I make a small change in my (huge) source model, it is important that the change is incrementally propagated to the target model; I don't want the entire target model to be regenerated every time.*

3. (similarly) *I would like to use code generation. However, when I make a small change in my (huge) model I don't want all the code to be regenerated.*
4. *In my company we have many developers and each manages only a specific part of the model. I would like each developer to be able to check out only a part of the model, edit it locally and then merge the changes into the master copy. The system should also let the developers know if their changes are in conflict with the rest of the model or with changes done by other developers.*

Instead of attempting to answer such questions directly, we find it useful to consider analogies with a proven and widely used environment that addresses those problems in a different – but highly relevant – domain. The domain is code development and the environment is the well known and widely used Eclipse Java Development Tools (JDT).

As a brief overview, JDT provide an environment in which developers can manage huge code-bases consisting of (tens of) thousands of Java source code files (concern 1). JDT supports incremental consistency checking and compilation (concerns 2,3) in the sense that when a developer changes the source code of a particular Java class, only that class and any other classes affected by the change – as opposed to all the classes in the project or the workspace – are re-validated and re-compiled. Finally, JDT is orthogonal to version control, collaborative development (concern 4), and multi-tasking tools such as CVS and SVN and Mylyn.

3   Managing Volume Increase

As models grow, tools that manage them, such as editors and transformation engines, must scale proportionally. A common concern often raised is that modelling frameworks such as EMF [2] and widely-used model management languages do not scale beyond a few tens of thousands of model elements per model. While this is a valid concern, it is also worth mentioning that the Java compiler does not allow Java methods the body of which exceed 64 KB, but in the code-development domain this is rarely a problem.

The reason for this asymmetry in perception is that in code development, including all the code of an application in a single method/file is considered – at least – bad practice. By contrast, in modelling it is deemed perfectly *reasonable* to store a model that contains thousands of elements in a single file. Also, it is *reasonable* that any part of the model can be hard-linked with an ID-based reference to any other part of the model.

To deal with the growing size of models and their applications, modelling frameworks such as EMF support lazy loading and there are even approaches, such as Teneo [3] and CDO [4], for persisting models in databases. Although useful in practice, such approaches appear to be temporary workarounds that attempt to compensate for the lack of encapsulation and modularity constructs in modelling languages. In our view, the issue to be addressed in the long run is not how to manage large monolithic models but how to separate them into

smaller modular and reusable models according to the well understood principles defined almost 40 years ago in [5], and similarly to the practices followed in code development.

4   Incrementality

In the MDE research community, incrementality in model management is sought mainly by means of purely declarative model transformation approaches [6, 7]. The hypothesis is that a purely declarative transformation can be analysed automatically to determine the effects of a change in the source model to the target model. Experience has demonstrated that incremental transformations are indeed possible but their application is limited to scenarios where the source and target languages are similar to each other, and the transformation does not involve complex calculations.

JDT achieves incrementality without using a declarative language for compiling Java source to bytecode; instead it uses Java which is an imperative language. The reason JDT can achieve incremental transformation lies mainly the design of Java itself. Unlike the majority of modelling languages, Java has a set of well-defined modularity and encapsulation rules that, in most cases, prevent changes from introducing extensive ripple effects.

But how does JDT know what is the scope of each change? The answer is simple: it is hard-coded (as opposed to being automatically derived by analysing the transformation). However, due to the modular design of the language, those cases are relatively few and the benefits delivered justify the choice to hard-code them. Also it is worth noting that the scope of the effect caused by a change is not related only to the change and the language but also to the intention of the transformation. For example, if instead of compiling the Java source code to bytecode we needed to generate a single HTML page that contained the current names of all the classes we would unavoidably need to re-visit all the classes (or use cached values obtained earlier).

5   Collaborative Development

As discussed in Section 2, a requirement for an MDE environment of industrial strength is to enable collaborative development of models. More specifically, it is expected that each developer should be able to check out an arbitrary part of the model, modify it and then commit the changes back to the master copy/repository. Again, the formulation of this requirement is driven by the current status which typically involves constructing and working with large monolithic models. With enhanced modularity and encapsulation, big models can be separated into smaller models which can then be managed using robust existing collaborative development tools such as CVS and SVN, augmented with model-specific version comparison and merging utilities such as EMF Compare [8]. Given the criticality

of version control systems in the business context, industrial users are particularly reluctant to switching to a new version control system[1]. Therefore, our view is that radically different solutions, such as dedicated model repositories, that do not build on an existing robust and proven basis are highly unlikely to be used in practice.

6 Modularity in Modelling Languages

The above clearly demonstrate the importance of modularity and encapsulation for achieving scalability in MDE. There are two aspects related to modularity in modelling: the design of the modelling language(s) used and the capabilities offered by the underlying modelling framework. In this section we briefly discuss how each of those aspects affect modularity and envision desirable capabilities of modelling frameworks towards this direction.

6.1 Language Design

With the advent of technologies such as EMF and GMF [9], implementing a new domain-specific modelling language and supporting graphical and textual editors is a straightforward process and many individuals and organizations have started defining custom modelling languages to harvest the advantages of the context-specific focus of DSLs. When designing a new modelling language, modularity must be a principal concern. The designers of the language must ensure that large models captured using the DSL can be separated into smaller models by providing appropriate model element *packaging* constructs. Such constructs may not be part of the domain and therefore they are not easily foreseeable. For example, when designing a DSL for modelling relational databases, it is quite common to neglect *packaging*, because relational databases are typically a flat list of tables. However, when using the language to design a database with hundreds of tables, being able to group them in conceptually coherent packages is highly important to the manageability and understandability of the model.

6.2 Modelling Framework Capabilities

In contemporary modelling frameworks there are three ways to capture relationships between two elements in a model: containment, hard references and soft references. Containment is the natural relationship of one element being a composite part of another, a hard reference is a unique-ID-based reference that can be resolved automatically by the modelling framework and a soft reference is a reference that needs an explicit resolution algorithm to navigate [10].

To enable users to split models over multiple physical files, contemporary modelling frameworks support cross-model containment (i.e. the ability of a

---

[1] Evidence of this is that CVS which was introduced in the 1980s is still the most popular version control system despite its obvious limitations compared to newer systems such as SVN

model element to contain another despite being stored in different physical files). With regard to hard and soft non-containment references, hard references are typically proffered because they can be automatically resolved by the modelling framework and thus, they enable smooth navigation over the elements of the model with languages such as OCL and Java. Nevertheless, in our view hard references are particularly harmful for modularity as they increase coupling between different parts of the model and prevent users from working independently on different parts. On the other hand, soft references enable clean separation of model fragments but require custom resolution algorithms which have to be implemented from scratch each time.

To address this problem, we envision extensions of contemporary modelling frameworks that will be able to integrate resolution algorithms so that soft references can be used, and the efficient and concise navigation achievable with languages such as OCL can still be performed.

7   Conclusions

In this paper we have demonstrated the importance of modularity and encapsulation for achieving scalability in MDE. We have identified two main problems: neglect of modularity constructs during the design of modelling languages and extensive use of ID-based references that lead to high coupling between different parts of the model. With regard to the first issue we have been working on preparing a set of guidelines for the design of scalable and modular DSLs and expect to report on this soon. The second issue is quite more complex and we plan to elaborate and prototype a solution based on EMF and Epsilon [11] in the near future.

Acknowledgements

References

1. Jos Warmer, Anneke Kleppe. Building a Flexible Software Factory Using Partial Domain Specific Models. In *Proc. 6th OOPSLA Workshop on Domain-Specific Modeling*, Portland, Oregon, USA, October 2006.
2. Eclipse Foundation. Eclipse Modelling Framework. http://www.eclipse.org/emf.
3. Eclipse Foundation. Teneo, 2008. http://www.eclipse.org/modeling/emft/?project=teneo.
4. Eclipse Foundation. CDO, 2008. http://www.eclipse.org/modeling/emft/?project=cdo.
5. David L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of ACM*, 15(12):1053–1058, 1972.

6. David Hearnden, Michael Lawley, Kerry Raymond. Incremental Model Transformation for the Evolution of Model-Driven Systems. In *Proc. Model Driven Engineering Languages and Systems*, pages 321–335.

7. Holger Giese, Robert Wagner. From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling*, pages 1619–1374, March 2008.

8. Eclipse Foundation. EMF Compare, 2008. http://www.eclipse.org/modeling/emft/ ?project=compare.

9. Eclipse GMF - Graphical Modeling Framework, Official Web-Site. http://www.eclipse.org/gmf.

10. Dimitrios S. Kolovos, Richard F. Paige and Fiona A.C. Polack. Detecting and Repairing Inconsistencies Across Heterogeneous Models. In *Proc. 1st IEEE International Conference on Software Testing, Verification and Validation*, pages 356–364, Lillehammer, Norway, April 2008.

11. Extensible Platform for Specification of Integrated Languages for mOdel maNagement (Epsilon). http://www.eclipse.org/gmt/epsilon.

Behavior, Time and Viewpoint Consistency: Three Challenges for MDE

José Eduardo Rivera[1], José Raul Romero[2], and Antonio Vallecillo[1]

[1]Universidad de Málaga (Spain)
[2]Universidad de Córdoba (Spain)
{rivera,av}@lcc.uma.es, jrromero@uco.es

Abstract. Although Model Driven Software Development (MDSD) is achieving significant progress, it is still far from becoming a real Engineering discipline. In fact, many of the difficult problems of the engineering of complex software systems are still unresolved, or simplistically addressed by many of the current MDSD approaches. In this position paper we outline three of the outstanding problems that we think MDSD should tackle in order to be useful in industrial environments.

1 Introduction

Although both MDSD and MDA have experienced significant advances during the past 8 years, some of the key difficult issues still remain unresolved. In fact, the number of engineering practices and tools that have been developed for the industrial design, implementation and maintenance of large-scale, enterprise-wide software systems is still low — i.e. there are very few real *Model-Driven Engineering* (MDE) practices and tools. Firstly, many of the MDSD processes, notations and tools fall apart when dealing with large-scale systems composed of hundred of thousands of highly interconnected elements; secondly, MDE should go beyond conceptual modeling and generative programming: it should count on mature tool-support for automating the design, development and analysis of systems, as well as on measurable engineering processes and methodologies to drive the effective use of all these artifacts towards the predictable construction of software systems. In particular, engineering activities such as simulation, analysis, validation, quality evaluation, etc., should be fully supported.

We are currently in a situation where the industry is interested in MDE, but we can easily fail again if we do not deliver (promptly) anything really useful to them. There are still many challenges ahead, which we should soon address in order not to lose the current momentum of MDE.

In this position paper we focus on three of these challenges. Firstly, the specification of the behavioral semantics of metamodels (beyond their basic structure), so that different kinds of analysis can be conducted, e.g., simulation, validation and model checking. A second challenge is the support of the notion of time in these behavioral descriptions, another key issue to allow industrial systems to be realistically simulated and properly analyzed — to be able to conduct,

e.g., performance and reliability analysis. Finally, we need not only to tackle the *accidental* complexity involved building software systems, but we should also try to deal with their *essential* complexity. In this sense, the effective use of independent but complementary viewpoints to model large-scale systems, and the specification of correspondences between them to reason about the consistency of the global specifications, is the third of our identified challenges.

2 Adding Behavioral Semantics to DSLs

Domain Specific Languages (DSLs) are usually defined only by their abstract and concrete syntaxes. The abstract syntax of a DSL is normally specified by a metamodel, which describes the concepts of the language, the relationships among them, and the structuring rules that constrain the model elements and their combinations in order to respect the domain rules.

The concrete syntax of a DSL provides a realization of the abstract syntax of a metamodel as a mapping between the metamodel concepts and their textual or graphical representation (see Fig. 1). A language can have several concrete syntaxes. For visual languages, it is necessary to establish links between these concepts and the visual symbols that represent them — as done, e.g, with GMF. Similarly, with textual languages links are required between metamodel elements and the syntactic structures of the textual DSL.
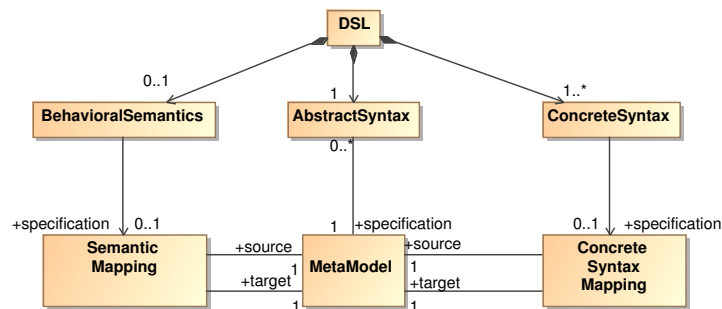
Fig. 1. Specification of a Domain Specific Language

Current DSM approaches have mainly focused on the structural aspects of DSLs. Explicit and formal specification of a model semantics has not received much attention by the DSM community until recently, despite the fact that this creates a possibility for semantic mismatch between design models and modeling languages of analysis tools [1]. While this problem exists in virtually every domain where DSLs are used, it is more common in domains in which behavior needs to be explicitly represented, as it happens in most industrial applications of a certain complexity. This issue is particularly important in safety-critical

real-time and embedded system domains, where precision is required and where semantic ambiguities may produce conflicting results across different tools. Furthermore, the lack of explicit behavioral semantics strongly hampers the development of formal analysis and simulation tools, relegating models to their current common role of simple illustrations.

The definition of the semantics of a language can be accomplished through the definition of a mapping between the language itself and another language with well-defined semantics (see Fig. 1). These *semantic mappings* [2] are very useful not only to provide precise semantics to DSLs, but also to be able to simulate, analyze or reason about them using the logical and semantical framework available in the target domain. In our opinion, in MDE these mappings can be defined in terms of model transformations.

Describing Dynamic Behavior. There are several ways for specifying the dynamic behavior of a DSL, from textual to graphical. We can find approaches that make use of, e.g., UML diagrams, rewrite logic, action languages or Abstract State Machines [3] for this aim. One particular way is by describing the evolution of the state of the modeled artifacts along some time model. In MDE, model transformation languages that support in-place update [4] can be perfect candidates for the job. These languages are composed of rules that prescribe the preconditions of the actions to be triggered and the effects of such actions.

There are several approaches that propose in-place model transformation to deal with the behavior of a DSL. One of the most important graphical approaches on this topic is graph grammars [5, 6], in which the dynamic behavior is specified by using visual rules. These rules are visually specified as models that use the concrete syntax of the DSL. This kind of representation is quite intuitive, because it allows designers to work with domain specific concepts and their concrete syntax for describing the rules [5]. There are also other graphical approaches, most of which are in turn based on graph grammars. Among them, we can find the visual representation of QVT [7] (where QVT is given in-place semantics) or the use of different (usually extended) UML diagrams [8, 9]. These approaches do not use (so far) the concrete syntax of the DSL, but an object diagram-like structure. Furthermore, most of them (including graph grammars approaches) use their own textual language to deal with complex behavior, such as Java [8] or Python [10].

Model Simulation and Analysis. Once we have specified the behavior of a DSL, the following step is to perform simulation and analysis over the produced specifications. Defining the model behavior as a model will allow us to transform them into different semantic domains. Of course, not all the transformations can always be accomplished: it depends on the expressiveness of the semantic approach. In fact, simulation and execution possibilities are available for most of the approaches in which behavior can be specified (including of course in-place transformations), but the kind of analysis they provide is normally limited. In general, each semantic domain is more appropriate to represent and reason about certain properties, and to conduct certain kinds of analysis [3].

A good example of this is Graph Transformation, which has been formalized into several semantic domains to achieve different kinds of analysis. Examples include Category theory to detect rule dependencies [11]; Petri Nets to allow termination and confluence analysis [5]; or Maude and rewrite logic to make models amenable to reachability and model-checking analysis [12]. We have been working on the formalization of models and metamodels in equational and rewriting logic using Maude [13]. This has allowed us to specify and implement some of the most common operations on metamodels, such as subtyping or difference [14], with a very acceptable performance. This formalization has also allowed us to add behavior [15] in a very natural way to the Maude specifications, and also made metamodels amenable to other kinds of formal analysis and simulation.

3 Adding Time to Behavioral Specifications

Formal analysis and simulation are critical issues in complex and error-prone applications such as safety-critical real-time and embedded systems. In such kind of systems, timeouts, timing constraints and delays are predominant concepts [16], and thus the notion of time should be explicitly included in the specification of their behavior. Most simulation tools that enable the modeling of time require specialized knowledge and expertise, something that may hinder its usability by the average DSL designer. On the other hand, current in-place transformation techniques do not allow to model the notion of time in a quantitative way, or allow it by adding some kind of clocks to the DSL metamodel. This latter approach forces designers to modify metamodels to include time aspects, and allows them to easily design rules that lead the system to time-inconsistent states [16].

One way to avoid this problem is by extending behavioral rules with their duration, i.e., by assigning to each action the time it needs to be performed. Analysis of this kind of timed rules cannot be easily done using the common theoretical results and tools defined for graph transformations. However, other semantic domains are better suited. We are now working on the definition of a semantic mapping to real-time Maude's rewrite logic [17]. This mapping brings several advantages: (1) it allows to perform simulation, reachability and model-checking analysis on the specified real-time systems; (2) it permits decoupling time information from the structural aspects of the DSL (i.e., its metamodel); and (3) it allows to state properties over both model states and actions, easing designers in the modeling of complex systems.

4 Viewpoint Integration and Consistency

Large-scale heterogeneous distributed systems are inherently much more complex to design, specify, develop and maintain than classical, homogeneous, centralized systems. Thus, their complete specifications are so extensive that fully comprehending all their aspects is a difficult task. One way to cope with such complexity is by dividing the design activity according to several areas of concerns, or *viewpoints*, each one focusing on a specific aspect of the system, as

described in IEEE Std. 1471. Following this standard, current architectural practices for designing open distributed systems define several distinct viewpoints. Examples include the viewpoints described by the growing plethora of Enterprise Architectural Frameworks (EAF): the Zachman's framework, ArchiMate, DoDAF, TOGAF, FEAF or the RM-ODP. Each viewpoint addresses a particular concern and uses its own specific (viewpoint) *language*, which is defined in terms of the set of concepts specific that concern, their relationships and their well-formed rules.

Although separately specified, developed and maintained to simplify reasoning about the complete system specifications, viewpoints are not completely independent: elements in each viewpoint need to be related to elements in the other viewpoints in order to ensure the *consistency* and *completeness* of the global specifications. The questions are: how can it be assured that indeed *one* system is specified? And, how can it be assured that no views impose contradictory requirements? The first problem concerns the conceptual *integration* of viewpoints, while the second one concerns their *consistency*. There are many approaches that try to tackle the problem of consistency between viewpoints, many of them coming from the ADL community (see, e.g., [3] for a list of such works). However, many of the current viewpoint modeling approaches to system specification used in industry (including the IEEE Std. 1471 itself and the majority of the existing EAFs) do not address these problems [18].

There are several ways to check viewpoint consistency. In some approaches such as the OpenViews framework [19], two views are consistent if a design can be found that is a refinement of both views. Other approaches, such as Viewpoints [20], consistency requirements are defined in terms of rules, which are specified as queries on the database that contains the viewpoints. The database performs then the consistency checks, using first-order logic. But the most general approach to viewpoint consistency is based on the definition of *correspondences* between viewpoint elements.

Correspondences do not form part of any of the viewpoints, but provide statements that relate the various different viewpoint specifications—expressing their semantic relationships. The problem is that current proposals and EAFs do not consider correspondences between viewpoints, or assume they are trivially based on name equality between correspondent elements, and are implicitly defined. Furthermore, the majority of approaches that deal with viewpoint inconsistencies assume that we can build an underlying metamodel containing all the views, which is not normally true. For instance, should such a metamodel consist of the intersection or of the union of all viewpoints elements? Besides, the granularity and level of abstraction of the viewpoints can be arbitrarily different, and they may have very different semantics, which greatly complicates the definition of the common metamodel.

Our efforts are currently focused on the development of a generic framework and a set of tools to represent viewpoints, views and correspondences, which are able to manage and maintain viewpoint synchronization in evolution scenarios, as reported in [21], and that can be used with the most popular existing EAFs.

6

References

1. Kleppe, A.G.: A language description is more than a metamodel. In: Proc. of ATEM 2007. (2007) `http://megaplanet.org/atem2007/ATEM2007-18.pdf`.
2. Harel, D., Rumpe, B.: Meaningful modeling: What's the semantics of "semantics"? Computer **37**(10) (2004) 64–72
3. Vallecillo, A.: A Journey through the Secret Life of Models. Position paper at the Dagstuhl seminar on Model Engineering of Complex Systems (MECS) (2008) `http://drops.dagstuhl.de/opus/volltexte/2008/1601`.
4. Czarnecki, K., Helsen, S.: Classification of model transformation approaches. In: OOPSLA'03 Workshop on Generative Techniques in the context of MDA. (2003)
5. de Lara, J., Vangheluwe, H.: Translating model simulators to analysis models. In: Proc. of FASE 2008. Number 4961 in LNCS, Springer (2008) 77–92
6. Kastenberg, H., Kleppe, A.G., Rensink, A.: Defining object-oriented execution semantics using graph transformations. In: Proc. of FMOODS 2006. Number 4037 in LNCS, Springer Verlag (2006) 186–201
7. Marković, S., Baar, T.: Semantics of OCL Specified with QVT. Software and Systems Modeling (SoSyM) (2008)
8. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story diagrams: A new graph rewrite language based on the unified modeling language. In: Proc. of the VI International Workshop on Theory and Application of Graph Transformation. (1998)
9. Engels, G., Hausmann, J.H., Heckel, R., Sauer, S.: Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In: Proc. of UML 2000. Number 1939 in LNCS, Springer (2000) 323–337
10. de Lara, J., Vangheluwe, H.: Defining visual notations and their manipulation through meta-modelling and graph transformation. Journal of Visual Languages and Computing **15**(3–4) (2006) 309–330
11. Ehrig, H., Karsten, Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer (2006)
12. Rivera, J.E., Guerra, E., de Lara, J., Vallecillo, A.: Analyzing rule-based behavioral semantics of visual modeling languages with maude. In: Proc. of SLE 2008. LNCS, Toulouse, France, Springer (2008)
13. Romero, J.R., Rivera, J.E., Durán, F., Vallecillo, A.: Formal and tool support for model driven engineering with Maude. **JOT 6**(9) (2007) 187–207
14. Rivera, J.E., Vallecillo, A.: Representing and operating with model differences. In: Proc. of TOOLS Europe 2008. Number 11 in LNBIP, Springer (2008) 141–160
15. Rivera, J.E., Vallecillo, A.: Adding behavioral semantics to models. In: Proc. of EDOC 2007, IEEE Computer Society (2007) 169–180
16. Gyapay, S., Heckel, R., Varró, D.: Graph transformation with time: Causality and logical clocks. In: ICGT. (2002) 120–134
17. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation **20**(1-2) (2007) 161–196
18. Romero, J.R., Vallecillo, A.: Well-formed rules for viewpoint correspondences specification. In: Proc. of WODPEC 2008. (2008)
19. Boiten, E.A., Bowman, H., Derrick, J., Linington, P., Steen, M.W.: Viewpoint consistency in ODP. Computer Networks **34**(3) (2000) 503–537
20. Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M.: Viewpoints: a framework for integrating multiple prespectives in systems development. SEKE journal **2**(1) (1992) 31–58
21. Eramo, R., Pierantonio, A., Romero, J.R., Vallecillo, A.: Change management in multi-viewpoint systems using ASP. In: Proc. of WODPEC 2008. (2008)