View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Sheffield Hallam University Research Archive

Meta-Modeling Model-Based Engineering Tools (Dagstuhl Seminar 13182)

CLARK, Tony, FRANCE, Robert B, GOGOLLA, Martin and SELIC, Bran V

Available from Sheffield Hallam University Research Archive (SHURA) at:

http://shura.shu.ac.uk/12044/

This document is the author deposited version. You are advised to consult the publisher's version if you wish to cite from it.

Published version

CLARK, Tony, FRANCE, Robert B, GOGOLLA, Martin and SELIC, Bran V (2013). Meta-Modeling Model-Based Engineering Tools (Dagstuhl Seminar 13182). Dagstuhl Reports, 3 (4).

Repository use policy

Copyright © and Moral Rights for the papers on this site are retained by the individual authors and/or other copyright owners. Users may download and/or print one copy of any article(s) in SHURA to facilitate their private study or for non-commercial research. You may not engage in further distribution of the material or use it for any profit-making activities or any commercial gain.

Sheffield Hallam University Research Archive
http://shura.shu.ac.uk

Report from Dagstuhl Seminar 13182

Meta-Modeling Model-Based Engineering Tools

Edited by

Tony Clark[1], Robert B. France[2], Martin Gogolla[3], and Bran V. Selic[4]

1 **Middlesex University, GB,** t.n.clark@mdx.ac.uk
2 **Colorado State University, US,** france@cs.colostate.edu
3 **Universität Bremen, DE,** gogolla@informatik.uni-bremen.de
4 **Malina Software Corp. – Nepean, CA,** selic@acm.org

Abstract

Model-based engineering (MBE) is a software development approach in which abstraction via modeling is used as the primary mechanism for managing the complexity of software-based systems. An effective approach to software development must be supported by effective technologies (i.e., languages, methods, processes, tools). The wide range of development tasks that effective MBE approaches must support leads to two possible tooling scenarios. In the first scenario a federated collection of tools is used to support system development. Each tool in the collection provides specialized services. Tool interoperability and consistency of information across the tools are major concerns in this scenario. These concerns are typically addressed using transformations and exposed tool interfaces. Defining and evolving the transformations and interfaces requires detailed low-level knowledge of the tools and thus leads to complex tooling environments that are difficult to configure, learn, use, and evolve. In the second scenario, a single tool is used to support the complete modeling lifecycle. This avoids the inter-tool transformation and consistency problems, but the resulting multi-featured tool is a monolithic entity that is costly to develop and evolve. Furthermore, the large number of non-trivial features can make learning and using such tools difficult.

Successful uptake of MDE in industry requires supporting tools to be, at least, useful and usable. From a tool developer's perspective, there is also a need to significantly reduce the cost and effort required to develop and evolve complex MBE tools. This seminar brings together experts in the areas of MBE, meta-modeling, tool development, and human-computer interactions to map out a research agenda that lays a foundation for the development of effective MBE tools. Such a foundation will need to support not only interoperability of tools or tool features, but also the implementation of high quality MBE tools. The long-term objective is to foster a research community that will work on a foundation that can be expressed in the form of standard tool (meta-)models that capture and leverage high quality reusable MBE tool development experience.

Seminar 28. April–03. May, 2013 – www.dagstuhl.de/13182
1998 ACM Subject Classification D.2 Software Engineering, H.1 Models and Principles
Keywords and phrases meta-modeling, model-based engineering, models, tools, domain specific modeling languages
Digital Object Identifier 10.4230/DagRep.3.4.188
Edited in cooperation with Dustin Wüest

Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

1 Executive Summary

Tony Clark
Robert B. France
Martin Gogolla
Bran V. Selic

License Ⓒ Creative Commons BY 3.0 Unported license
© Tony Clark, Robert B. France, Martin Gogolla, and Bran V. Selic

The 33 participants at the Meta-Modeling Model-Based Engineering Tools (M^{3}BET) Dagstuhl Seminar were brought together to explore how model-based engineering (MBE) techniques can be used to improve the quality of software modeling tools. The participants included expert researchers, practitioners and tool developers in the software/system modeling and the human computer interaction communities. The discussions aimed to answer the following question: Can MBE techniques be used to produce more effective MBE tools, and, if so, how should it be done?

The vision underlying the seminar is one in which technologists create tool models that specify desired tool features, and tool modeling frameworks that are used to analyze, compose, transform, simulate and otherwise manipulate the tool models. In the vision, tool developers will use tool models and frameworks to produce useful, usable and cost-effective software modeling tools.

Seminar Organization

Day/Session	Monday	Tuesday	Wednesday	Thursday	Friday
Morning	Presentation and discussion of seminar objectives and outcomes; Introductions	Short presentations; Formation of working groups	Plenary session; Working group sessions	Working group sessions	Drafting seminar conclusions and plans
Afternoon	5 minute presentations by participants	Working group sessions	Demos; Social event	Demos; Plenary session	Seminar closure

Figure 1 The Final Seminar Program.

The final seminar program is given in Figure 1. On the first day the seminar objective and outcomes were presented. The seminar objectives, as presented on that day, was to better understand the "what, why, and how" of tool models, and initiate work on (1) languages for tool modeling, (2) MBE methods and technologies for tool development, and (3) tool modeling frameworks.

The planned outcomes were (1) reports on the results of group discussions, (2) a research roadmap for achieving the tool modeling vision, (3) potential solutions for achieving the vision, and (4) initiation of new research collaborations among participants.

To help shape and initiate the discussions, the organizers proposed the following as an initial set of breakout group topics:

Tool capabilities – The intent was that discussions in this group would focus on identifying the software tool capabilities that should be captured in tool models, and on

how these capabilities could be captured in tool metamodels. This covers discussions on (1) how metamodels can be used to describe tool capabilities in a manner that supports generation of high-quality tool components, (2) the utility and feasibility of defining tool metamodels, (3) potential benefits associated with and purposes served by a tool metamodel, and (4) challenges associated with developing an effective metamodel (i.e., a metamodel that is fit-for-purpose).

Tool qualities – Discussions in this group would aim to answer questions about desirable tool qualities (e.g., What issues determine tool adoption and why?). This includes key questions related to, for example, usability/human factors, scalability, interoperability, as well as non-technical but important considerations related to organization goals, culture, and processes.

Tool ecosystems – A tool framework can be thought of as a technological ecosystem that involves both tools as well as tool users. Discussions in this group would seek answers to questions such as: What are the features of a good tools framework? Are there candidate frameworks available? If so, are they sufficient or do they need to be extended?

Tool development methods – Discussions in this group would focus on answering the following questions: How can MBE be applied to the development of MBE tools? What types of languages are suitable for describing tools? How can tool quality issues be addressed by such methods?

Working Groups

During the discussions on group topics it was decided to base the groups on tool concerns and issues that the participants had some stake in. It turned out that the concerns and issues that arose from the discussions were mostly derived from those underlying the groups proposed by the organizers.

The concerns and issues were then clustered into two groups based on participant interests. Group A consisted of usability, utility, and broader non-technical concerns (e.g., designing tools that support how developers work and think, designing and performing usability studies, adapting tool features to user expertise and desired level of formality, marketing/business/cultural concerns). Group B consisted of the more technical concerns, for example, concerns related to tool development methods, scalability, support for separation of concerns, tool quality assessment and benchmarking.

Group B concerns were further grouped into two categories: Composition, and Methods and Quality concerns. The Composition concerns included issues related to tool, language, and model composition, and the use of multi-models with heterogeneous semantics.

Three Working Groups, each focusing on one of the above concern groups, were formed on the seminar's first day. A summary of the discussions that took place in each group is included in the report.

Summary and Future Work

One of the major insights gained during the seminar was that a good understanding of the utility of tool models and the identification of appropriate forms of tool models/metamodels requires one to first address more fundamental tool development and assessment concerns. On hindsight, this should not have been a surprising result; effective tool models would have to capture significant tool development and assessment experience and knowledge and thus such experience and knowledge needs to be distilled first. The seminar provided a good forum for discussing and organizing the experience and knowledge of the participants.

Post-seminar collaborations that will utilize these results to develop an initial set of tool models/metamodels were initiated at the seminar.

In addition to the planned collaborations on tool models, the participants also agreed to engage in the following post-seminar activities:

- Publications: The following publications are planned
  - A special issue of the Software and System Modeling (SoSyM) journal that will include articles that focus on the concerns and issues discussed at the seminar.
  - A paper that discusses problems associated with current software modeling tools.
- Workshops: Workshops in which participants will discuss and develop tool models/metamodels will be held at conferences such as MODELS 2014 and ICSE 2014.

2 Table of Contents

Executive Summary
Tony Clark, Robert B. France, Martin Gogolla, and Bran V. Selic 189

Overview of Talks

Towards Truly View-Based Software Engineering Environments
Colin Atkinson 195

Meta-Models for Model Based Engineering Tools
Tony Clark 196

MDE and SLE Cross Fertilization
Benoit Combemale 197

Human Activity Modeling of Human Modeling Activities
Larry Constantine 197

Abstract Modeling for Interaction Design
Larry Constantine 197

Lessons from Human Tool Use
Larry Constantine 198

Towards More Reliable Tools
Catherine Dubois 198

Megamodels in Model-Based Engineering Tools
Michalis Famelis 198

Demo Abstract: Typed Megamodeling with the Model Management Tool Framework
Michalis Famelis 199

Towards Empathetic Tool Design
Robert B. France 199

Flexible Modeling
Martin Glinz 200

Some Challenges and Expectations on Tool Support for Model-Based Engineering
Martin Gogolla 201

MBE Tools and Evolving Environments
Lars Hamann 201

Make Good Languages
Øystein Haugen 202

Engineering Tool Integration: Patterns
Gabor Karsai 202

MetaEdit+: Faster Meta-Modeling by Design
Steven Kelly 203

Demo Abstract for MetaEdit+
Steven Kelly 203

Finding Tool Paradigms
Thomas Kühne 204

Towards Greater Adoption of MDE by Industry Practice
Vinay Kulkarni 204

Servicing IT Needs of Future Enterprises
Vinay Kulkarni 208

Methods, not Models
Stephen J. Mellor 209

Computer Automated Multiparadigm Modeling for the Design of Software Systems in a Physical World
Pieter J. Mosterman 209

The Original Sin – Using Current Computers for Modeling Software
Pierre-Alain Muller 210

The Other Face of Models
Leonel Domingos Telo Nóbrega 211

The MetaSketch Workbench
Leonel Domingos Telo Nóbrega 211

Improving the Usability of Formal Verification Techniques Through Customizable Visualizations
Ileana Ober 212

Empirical Studies of Expert Software Design (Cuckoo or Canary?)
Marian Petre 213

Bridging the Gap Between Software Models and Analysis (Performance, Reliability, etc.) Models
Dorina C. Petriu 213

The Need for Scalable and Flexible Model-Based Engineering
Louis Rose 214

EuGENia Live: A Flexible Graphical Modelling Tool
Louis Rose 215

Compositional Model Based Software Development
Bernhard Rumpe 215

Formal Verfication and Model-Based Engineering
Martina Seidl 216

MBE Tools: Effective Automation Where It Helps
Bran V. Selic 217

Models in Software Engineering
Perdita Stevens 218

Tool Building Using Language Composition
Laurence Tratt 218

Generic Tools, Specific Languages
Markus Völter 218

The Tool's The Thing
Jon Whittle 219

Flexible, Lightweight Metamodeling
Dustin Wüest 219

Liberating Software Engineers from the Tyranny of a Strict Modeling Language
Dustin Wüest 220

Design and Design Thinking
André van der Hoek 221

Overview of Working Groups

Working Group A: Summary of Discussions and Conclusions
Bran V. Selic 221

Working Group B.1: Composition Issues in MBE Languages and Tools
Tony Clark 222

Working Group B.2: Research Questions for Validation and Verification in the Context of Model-Based Engineering
Martin Gogolla 225

Participants 227

3 Overview of Talks

3.1 Towards Truly View-Based Software Engineering Environments

Colin Atkinson (Universität Mannheim, DE)

License Creative Commons BY 3.0 Unported license
© Colin Atkinson
Main reference C. Atkinson, D. Stoll, C. Tunjic, and J. Robin, "A Prototype Implementation of an Orthographic Software Modeling Environment", VAO 2013, Montpellier 2013.
URL http://swt.informatik.uni-mannheim.de

Ensuring the coherence of all the data, artifacts and relationships generated during a software development project has always been an important aspect of software engineering, even in the days when software systems were typically written in a single language, but with modern development languages, paradigms and processes maintaining artifact coherence has arguably become one of the biggest challenges in software engineering. Without a continual campaign of "management" activities (e.g. configuration management, change management, version management, release management, traceability management etc.) the coherence of software engineering artifacts inevitably degrades over time. At the concrete level of individual projects this coherence challenge is often experienced as an integration problem – more specifically, as the problem of integrating data, tools, and processes within a single environment. The software industry therefore has tended to tackle this challenge with "integration" or "interoperability" solutions (e.g. the Open Services for Lifecycle Collaboration (OSLC)). While such initiatives undoubtedly alleviate some of the immediate problems faced in using traditional tool suites, they do not really address the underlying problems. In particular, the number of pairwise consistency relationships that need to be maintained grows rapidly with the number of tool/views involved (n(n-1)/2) where n is the number of tools).

This talk takes the position that in order to address the root causes of the aforementioned problems, the next generation of software engineering environments and method should be based on the following three principles. The first principle is that the information edited and portrayed within the "artifacts" should be regarded as "views" derived from a single centralized description of the system a "Single Underling Model (SUM)". The most important consequence of this shift in perspective from artifact/tool centric handling of information to view/SUM centric handling of information is that no lifecycle management activities must be carried out at the artifact level (as is typically the case today). In other words, there must be no versions/variants of artifacts (e.g. code modules, PIMs etc), there must only be versions/variants of the system (and its components) as described in the SUM.

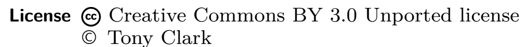
The second principle is that the best modeling architecture to realize such an approach is a multi-level-model approach that allows languages and services to be applied uniformly across multiple classification levels. Such an architecture is often referred to as an Orthogonal Classification Architecture (OCA). It is important to stress that the term "model" here is intended in a more general sense than the typical types of diagram used in model-driven development (i.e. class diagram or state diagrams etc.). In this context the term "model" refers to any form of information representation including all traditional software engineering artifacts such code and models of the kind commonly used in model-driven development.

The final principle is that a software engineering environment based on such an infrastructure should be "view" agnostic in the sense that it should be based on a method or development "paradigm" that is not driven by, or biased towards, any particular kind of view. In particular, this means that no particular kind of view such as "code" or "PIMs" should dominate the development process. This will make it possible to merge today's

advanced software engineering paradigms such as model- driven development, aspect-oriented development, product-line engineering and agile development in a much more fundamental way than has so far been possible.

3.2 Meta-Models for Model Based Engineering Tools

Tony Clark (Middlesex University, GB)


License Creative Commons BY 3.0 Unported license
© Tony Clark

Model Based Engineering claims to address issues including efficiency and quality of the Software Engineering process through the use of abstraction in terms of models. Models represent an aspect of a system and allow the engineer to design, analyze and manipulate the system aspect without needing to deal with implementation details that might otherwise impede progress.

Successful Model Based Engineering relies on tools that construct, analyze and transform models. There are many such tools, some free and come commercial, that are currently available. Unfortunately, the usability of such tools is generally viewed to be less than ideal, and has been claimed to be part of the reason why MBE has not become more widespread within the SE industry.

The lack of MBE tool usability may be attributed to various factors. These include the sheer size and complexity of typical MBE tool functionality where menus proliferate and reach many levels of nesting. It is often the case that a tool tries to support many (in some cases all) the actors in the software development life-cycle leading to tools that, for each individual user, offer a vast array of redundant features. Furthermore, such tools are often developed by software engineers who are not experts in design issues, leading to awkward or unintuitive interactions.

A proposal to address this problem is to apply MBE principles to the development of MBE tools. A meta-language for tool development would allow features such as design principles to be expressed and captured for families of tools. The resulting tool models could be processed by tool engines that make the development of smaller tools more economically attractive. MBE techniques such as slicing, transformation and merging could be applied to tool models thereby allowing tool fragments to be reused so that building larger tools is achievable by smaller groups, and can benefit from earlier tool verification efforts.

Over time, it may be possible to produce a standard model for expressing MBE tools leading to a marketplace in terms of meta-frameworks that process tool-models and in terms of tool fragments. Companies can have proprietary implementations of a meta-framework that produce a product-line of MBE tools tailored for a specific domain.

3.3 MDE and SLE Cross Fertilization

Benoit Combemale (IRISA – Rennes, FR)

License Creative Commons BY 3.0 Unported license
© Benoit Combemale
URL http://www.combemale.fr

Domain Specific Modeling Languages (DSMLs) are widely adopted to capitalize on business domain experiences. Consequently, DSML implementation is becoming a recurring activity, whether newly defined language or by specialization of a more general language (e.g., UML). To be effective, a DSML requires an associate dedicated modeling environment including tools such as editors, simulators, and code generators. In practice, the implementation of such a modeling environment is a complex and time consuming task. However, it is commonly realized from scratch for each DSML, even for languages which share some concepts and could share bits of tool support. A major obstacle is that there is no opportunity to capitalize the work, even if the different modeling languages and their respective tools are very closed or should be coordinated. In this talk, we explore the challenges associated to the cross-fertilization of Software Language Engineering, Global Software Engineering and Model Driven Engineering to support a Coordinated Model-Driven Language engineering. In particular, we focus on a uniform approach for language reuse, variability, composability and coordination.

3.4 Human Activity Modeling of Human Modeling Activities

Larry Constantine (University of Madeira – Funchal, PT)

License Creative Commons BY 3.0 Unported license
© Larry Constantine

Human Activity Modeling (hAM) is a systematic formalism based on activity theory. Activity theory is a well-established framework for describing and understanding human activities of all kinds. The systematic notation of hAM provides simplified conventions that support user experience and interaction design and builds a shared vocabulary that bridges to software engineering. Recent work extends the range of applications to include social sciences research and organization analysis. In this presentation, hAM is applied recursively to the activities that analysts, designers, and engineers carry out when using models.

3.5 Abstract Modeling for Interaction Design

Larry Constantine (University of Madeira – Funchal, PT)

License Creative Commons BY 3.0 Unported license
© Larry Constantine

Recent work has extended Canonical Abstract Prototypes (CAPs) to support the design of large, complex multi-modal, multi-media, multi-channel systems and services. The resulting dynamic, distributed CAPs (ddCAPs) accomplish a separation of concerns that facilitates fuller specification of dynamic and behavioral aspects of user interfaces at an abstract level. The notation of ddCAP models provides a simpler, more effective platform for communication between interaction designers and software engineers.

3.6 Lessons from Human Tool Use

Larry Constantine (University of Madeira – Funchal, PT)

License Creative Commons BY 3.0 Unported license
© Larry Constantine

Findings from ethnographic and social science investigations of tool use are reviewed for concrete guidelines for design and construction of better modeling tools. Informed by activity theory inquiries into how humans organize and use tools in complex, skilled activities, a dozen specific recommendations are presented for enhancing user performance in modeling and use of modeling tools.

3.7 Towards More Reliable Tools

Catherine Dubois (ENSIIE – Evry, FR)

License Creative Commons BY 3.0 Unported license
© Catherine Dubois

Tools are fundamental when using formal methods and when applying a model based engineering approach. And they are many MBE tools. However a question arises: do you trust your tools ? The position I try to defend is that we can adopt a formal approach to verify and/or develop the MBE tools. In particular a formalized semantics of a modeling language is necessary to build tools in order to provide them with a basis as solid as possible. Several researchers have tackled this problem, using Coq, Isabelle or Maude, either to provide sound mathematical foundations for the study and the validation of MDE technologies or to verify some model transformations. However it is a challenging task (and hard work) to develop such formalizations in proof assistants like Coq or Isabelle. We can notice that modeling languages often share some notions or components. Consequently a research perspective could be to develop a framework allowing to define the semantics of these common component together with a way of combining them in order to define formally the mechanized semantics of a modeling language and thus verify tools and model transformations. DSL, aspects, variability etc. are, according to me, ingredients of such a framework.

3.8 Megamodels in Model-Based Engineering Tools

Michalis Famelis (University of Toronto, CA)

License Creative Commons BY 3.0 Unported license
© Michalis Famelis

I briefly introduced the ideas of my research group regarding the use of typed megamodeling to improve model management in model-based engineering tools. Through the explicit management of models, their metamodels, and relations and transformations between them, we aim to create support for ecosystems of modular, reusable tools. Typed megamodels can be used at tool runtime for online tool extension and reconfiguration, while providing semantic consistency for tasks such as language extension and adaptation. Tooling tasks can also be better supported by leveraging the megamodel type system, by taking into account type casting, polymorphism, type safety, etc. I illustrated these ideas using the Model

Management Tool Framework (MMTF), an Eclipse-based pluggable infrastructure developed at the University of Toronto. Megamodels can also be used to support collaboration and knowledge transfer among multiple users of tools. I illustrated this using the example of Modelepedia, a semantic-wiki companion to MMTF.

3.9 Demo Abstract: Typed Megamodeling with the Model Management Tool Framework

Michalis Famelis (University of Toronto, CA)

License Creative Commons BY 3.0 Unported license
© Michalis Famelis

Model management has emerged as an important approach to addressing the complexity introduced by the use of many interrelated models in Software Engineering. In this tool demonstration, I highlighted the capabilities of the Model Management Tool Framework (MMTF) – an Eclipse-based pluggable infrastructure for rapidly developing model management tools. MMTF differs from other model management frameworks through its emphasis on interactive model management, graphical megamodeling and strong typing. Using examples from automotive software development, I illustrated how MMTF can be used as a runtime typed megamodel to enhance the capabilities of modeling tools.

3.10 Towards Empathetic Tool Design

Robert B. France (Colorado State University, US)

License Creative Commons BY 3.0 Unported license
© Robert B. France

There is no doubt that research in model driven (software) development (MDD) has advanced the state-of-art in software modeling and yielded revealing insights into the challenging problems that developers of complex software-based systems face. Yet, sustainable use of MDD technologies in industry is still elusive. While many practitioners have some appreciation of the benefits associated with proper use of MDD approaches, technologists (tool architects and developers) have fallen short of providing the types of tools needed to fully support MDD practices. Specifically, a pesky point of friction is the limited number of usable MDD tools that are available today.

Practitioners understandably judge the effectiveness of MDD approaches in terms of tool quality: If the tool is bad then the approach it supports is perceived as being bad. While this perception may be logically flawed, it is understandable; tools are at the front-end of the battle to manage software complexity through MDD approaches, and thus their failure to support MDD practices and the work styles of MDD practitioners has a significant impact on how well MDD practices can be supported in a sustainable manner. In other words, tools should make the power of MDD readily accessible to developers.

Poor tool support can contribute to accidental complexity. This gives rise to the perception that MDD adds complexity to the software development process. In many cases, tools are too heavyweight, that is, they have many features that support a broad spectrum of development activities, in addition to a complex infrastructure for managing the features and for extending the tool with new features. The problem with such broad-spectrum tools is that it can take considerable effort to, for example, (1) identify and get to the subset of features needed to support a particular activity, (2) learn how to effectively use a desired feature, and (3)

learn how to use a combination of features to support development workflows that cut across different development phases (e.g., to support coordinated use of models at the requirements, architecture, and detailed design stages). This effort is a significant contributor to the accidental complexity associated with many heavyweight MDD tool environments.

The above problems stem from a lack of attention to tool usability. Rather than build tools that fit the working style of MDD practitioners, it seems that tool developers/technologists arbitrarily impose particular working styles on tool users. The problem stems from a failure on the part of the tool developers to perform usability studies in which MDD practitioners at different skill levels perform their work using the tools and provide feedback on how well tools support their work. More specifically, tool developers need to practice what is sometimes called empathetic design, where tool design activities are centered on the practitioners that technologists seek to influence. MDD technologists need to put practitioners at the "front and center" of their tool design activities. Currently, technologists work with a model of practitioners that is often not validated, that is, they may be working with flawed models of practitioners. It is thus not surprising that tools often do not fit the working styles of practitioners and thus are perceived as adding to the complexity of developing software-based systems.

There is a need for MDD tools that amplify a modeler's skills and good working habits. The antithesis is that a tool should not force a practitioner to change good working styles, nor get in the way of skillful use of modeling concepts. If a tool imposes a particular working style on practitioners it should be done in a manner that makes the benefits clearly apparent to practitioners (i.e., the effort required to use a tool should be significantly offset by gains in productivity or product quality). Both researchers and technologists are needed to realize this vision of usable tools. Researchers need to conduct studies of how practitioners at various skill levels work. Such empirical results can help technologists build more accurate models of their target audience. Technologists also need to continually evaluate the usability of their tools using representative sets of practitioners, and to evolve their tools accordingly.

In closing, I would also like encourage technologists to consider incorporating features that practitioners can use to develop their modeling skills (i.e., not just support current practices or amplify skills). This can take the form of, for example, suggestions for improving models based on modeling "bad smells" and on the detection of anti-patterns in models. Such tools would be particularly useful in classrooms where next-generation modelers are molded.

3.11 Flexible Modeling

Martin Glinz (Universität Zürich, CH)

License Creative Commons BY 3.0 Unported license
© Martin Glinz
Joint work of Glinz, Martin; Wüest, Dustin
URL http://www.ifi.uzh.ch/rerg/research/flexiblemodeling.html

In my research group at the University of Zurich, we are working on flexible modeling, developing the FlexiSketch tool as a major constituent of our research. We try to understand what's the essence of a flexible modeling tool that makes it a must-have thing. We also investigate the problem of metamodeling by end- users, empowering them to metamodel without knowing it. Future work will investigate how to organize and navigate large collections of sketches, text fragments and model fragments.

3.12 Some Challenges and Expectations on Tool Support for Model-Based Engineering

Martin Gogolla (Universität Bremen, DE)

License Creative Commons BY 3.0 Unported license
© Martin Gogolla

This position statement is formulated from the viewpoint of having some experience in various fields like algebraic specification, temporal logic, entity-relationship modeling, graph transformation, and syntax and semantics of modeling languages. Furthermore, our considerations are influenced by a current work focus on OCL (i.e., a textual expression language) as part of UML (i.e., a modeling language with graphical and textual syntax being able to be used as a general purpose and domain specific language and allowing the characterization of design time and runtime phenomena). Our previous work concentrated on validation and verification techniques for assuring model properties and profits from the development of an academic UML and OCL tool during recent years which is currently used within a larger German project.

Key challenges for MBE tool support include (a) bridging the gap between modeling and programming, (b) efficiently and automatically transforming descriptive models into efficient programs (which may be viewed as prescriptive models) respecting properties assured for source models, (c) enabling communication between modeling tools which have different capability focus, and (d) developing benchmarks (test suites) for models and for modeling tools in order to compare functionality, efficiency, usability, exchangeability, learnability (and other criteria) of modeling techniques and tools. As possible outcomes of future work we identify (a) formal comparison criteria for and modeling features of validation and verification modeling tools and their quality assurance, (b) connecting validation and verification modeling tools with modeling tools having different focus (e.g., transformation tools, code generation tools, performance analysis tools, process management tools, teaching tools), and (c) formal criteria for rating the capabilities of modeling tools.

3.13 MBE Tools and Evolving Environments

Lars Hamann (Universität Bremen, DE)

License Creative Commons BY 3.0 Unported license
© Lars Hamann

Years back I worked as a “non-MBE” developer, but also in this past, models were all around. However, they were only used for specific domains during different development phases, like for example, the relational model of an information system. Unfortunately, these models containing so much information ended up as a historical artifact pinned at a wall. The modeling tools used to generate these artifacts were all well suited to create models in their particular domain, but considered a small part of their surrounding world only. Either, they transformed (called export those days) their model into something with lesser abstraction loosing all the semantic of the higher abstraction or they simply could export it to another (established) tool for the same domain (to be honest, to import data from an established tool was the more common case, because the user should use the importing tool).

In present days, import and export are called model transformation and common data formats like EMF, support transferring models. However, tools still consider only a small part

of their surrounding world (call it ecosystem if you like). They define their requirements on the ecosystem in a self-serving way ignoring changes to their outside or, in my opinion, much more important they require modification to the outside. This makes it nearly impossible to compose a tool-set of specialized tools. One future challenge in making MBE-based tools more accepted is to build tools in a more solidary way. To relate the aforementioned observations to a possible meta-model of model-based engineering tool, a possible future meta-model for MBE-tools should support this composition of evolving and not evolving tools. A simple, but important feature would be to integrate versioning of tools and their used meta-models. In a strong sense, it could require a tool to provide model-transformations between versions.

3.14 Make Good Languages

Øystein Haugen (SINTEF – Oslo, NO)

License Creative Commons BY 3.0 Unported license
© Øystein Haugen

If Model-Based Engineering Tools should be meta-modeled, then their success will be dependent on the means for meta-modeling. Meta- modeling is dependent on meta-languages such as MOF (standardized by OMG), but experience shows that even extremely well qualified and competent people fail to be able to express their intent in MOF. MOF is not a good domain-specific language for describing languages which is what it should have been to make meta-modeling simpler. We need a better standardized meta-language such that those tools and techniques that are built on the meta-models will have better quality, be more precise and unambiguously interpreted. Let good people make good languages for good tooling.

3.15 Engineering Tool Integration: Patterns

Gabor Karsai (Vanderbilt University, US)

License Creative Commons BY 3.0 Unported license
© Gabor Karsai

Model-driven development of systems and software implies that many, different types of models are used. Models can represent anything and everything about the system being constructed, and arguably there is no single model-based engineering tool that can cover all aspects of the process. The MDE community has developed various techniques, standards, and tools for model representation, manipulation, management, transformation, etc. and the problem of model integration has been recognized and appreciated. Model-based engineering tools support an engineering process, where different types of models are used for different purposes. Unless the seamless integration of such models is solved, model-based processes and tools will always be difficult to use and will meet resistance from the practitioners of the traditional, document-oriented processes. A couple of requirements for model integration are straightforward. For model integration we need well-defined *interfaces* between modeling languages, although the precise nature of such interfaces is a subject of research at this point. Arguably mapping between model elements in different modeling languages is a part of the solution, but the interface is probably more complex. A specific model element (*unit of*

knowledge) must be entered during the engineering process only once, and all derived and dependent elements must be related to that single instance. Models must undergo semantics-preserving transformations – for analysis, code generation, verification, testing, etc., purposes. Frequently transformations must be bi-directional: for instance, analysis models should be derived from design models, but the results of the analysis must be translated back into the language of the design. There has been several model and tool integration patterns developed since model-driven development started. Hub-and-spoke, point-to-point translation, and fine-grain linking of model elements across models are the three major examples. It is a major research challenge to make such model integration approaches usable and scalable. Arguable, the most challenging aspect is the tool semantics: if models are to be translated (or linked) between two tools, there has to be some semantic mapping between the metamodels of these tools. The precise, yet actionable specification and efficient implementation of this semantic mapping remains a continuing research problem. Another challenging aspect is the specific process support: engineering processes are often designed and *tooled* for a specific product family. In other words, the model integration and its executable manifestation should be customizable to specific processes and tools – we need reusable model integration frameworks that allow this. Finally, there are several implementation-related issues: efficient and robust synchronization of models among the tools, distributed version control, support for collaborative work, and usability are the main issues in this area.

3.16 MetaEdit+: Faster Meta-Modeling by Design

Steven Kelly (MetaCase – Jyväskylä, FI)

License Creative Commons BY 3.0 Unported license
© Steven Kelly

Empirical research has shown that using UML does not lead to a significant increase in productivity over coding: plus or minus 15%. MDA tool manufacturers claim 20-40% productivity increases with their tools. Using Domain-Specific Modeling languages in MetaEdit+ has shown productivity improving to 500-1000% of that for coding, consistently across a number of different cases and domains. The design and implementation of the DSM language and generators has been found to take only 2-3 person weeks with MetaEdit+. Recent research by Eclipse modelling project committers (tinyurl.com/gerard12) showed that implementing a modelling language is 10-50 times faster with MetaEdit+ than with other graphical language workbenches, both commercial (RSA, Obeo) and open source (GMF, GME). As yet, it is an open research question as to which features in MetaEdit+ contribute most to this difference.

3.17 Demo Abstract for MetaEdit+

Steven Kelly (MetaCase – Jyväskylä, FI)

License Creative Commons BY 3.0 Unported license
© Steven Kelly

MetaEdit+ is a modeling, meta-modeling and generation tool targeting the creation and use of Domain-Specific Modeling languages. MetaEdit+ applies DSM to itself: the GOPRR language used for defining languages is itself a Domain-Specific Language designed from

scratch for this task. In the demo we showed how with MetaEdit+ you can quickly and easily incrementally build a language – including abstract syntax, concrete syntax and transformational semantics: draw models in the language, and automatically generate full code for the modeled system.

3.18 Finding Tool Paradigms

Thomas Kühne (Victoria University – Wellington, NZ)

License Creative Commons BY 3.0 Unported license
© Thomas Kühne

In tool design there is a tension between serving users in ways they ask for and prescribing paradigms that the tool designers believe are in the best interest of the user.

While it may take considerable effort on behalf of users to adapt to imposed paradigms, it may be an investment that will produce significant pay-off. An unsatisfied tool user should therefore not only ask "What is wrong with the tool?" but also "What is wrong with me?". Conversely, tool builders need to ask themselves not only "What can I build?" but also "What should I build?" trying to answer the latter question by thoroughly analysing what users need rather than what they want.

3.19 Towards Greater Adoption of MDE by Industry Practice

Vinay Kulkarni (Tata Consultancy Services – Pune, IN)

License Creative Commons BY 3.0 Unported license
© Vinay Kulkarni

Introduction. Model-driven software engineering has been around since mid-90s. Launch of OMG's MDA in 2000 generated widespread interest in MBSE. Today it can justifiably be said that model-driven development has proved beneficial in certain niche domains if not all. There is ample evidence of models being used in many ways viz., as pictures, as documentation aids, as jump-start SDLC artefacts, as primary SDLC artefacts etc [1, 2]. When used as primary SDLC artefacts, models shift the focus of software development from coding (in terms of the desired implementation technology primitives) to specifications at a higher level of abstraction [3]. These specifications can be kept independent of implementation technology platforms and hence can be targeted for delivery into multiple technology platforms through platform-specific code generation. Being closer to the problem domain, models present a more intuitive mechanism for domain experts to specify requirements. Being more abstract, size of application specification in terms of models is significantly smaller than its implementation and hence easier to review manually. Being rich in structure, models are amenable for constraints to be specified such that certain errors can be altogether eliminated and certain errors can be detected at modeling stage itself. Model-based code generators can be made to encode best coding practices and standards so as to deliver uniformly high code quality in a person-independent manner. Moreover, it is possible to bootstrap model- driven development so that model-based code generator implementation can be generated from its specification in model form [4]. Intuitiveness demand on models dictate they be domain-specific. Since there can be infinitely many domains with each domain possibly ever- expanding, it is impossible to think of a universal modeling language that can effectively cater to them all. Furthermore,

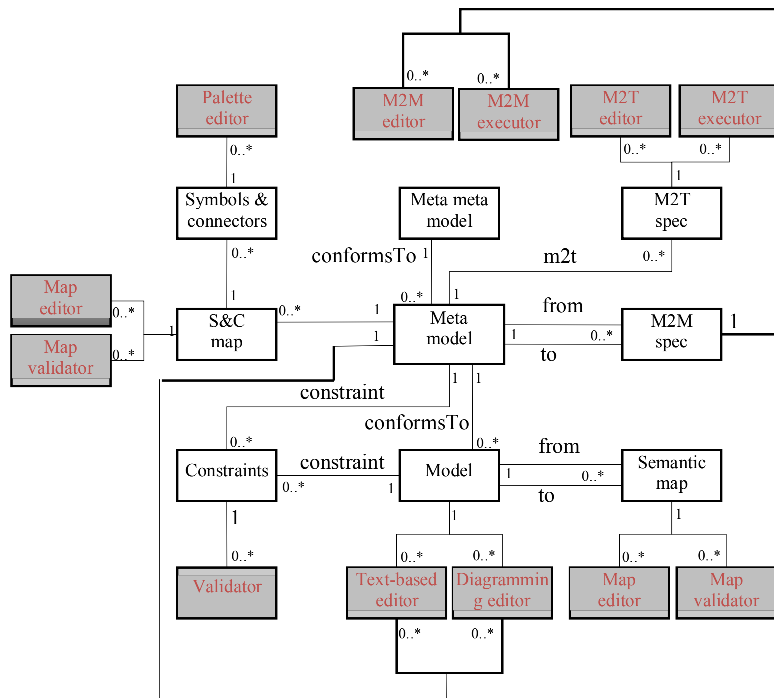


Figure 2 Meta-Model of Modeling Language Engineering Platform.

models are purposive and hence it is impossible to conceive a single modeling language that can cater to all possible purposes. Therefore, multiplicity of modeling languages is a reality. Separation of concerns principle makes the need for a cluster of related modeling languages (one for each concern in a domain) and a mechanism to relate the separately modeled concerns (say to compose a unified model) apparent. The need to relate otherwise separate models demands expressibility of one in terms of the other. Thus emerges the need for a common language capable of defining all possible modeling languages of interest. There are multiple stakeholders for a model each possibly having a limited view being presented in the form a suitable diagramming notation. From the above discussion, it follows there could be as many diagramming notations as there are modeling languages. And thus emerges the need for a language to define all possible visualizations of a model. For models to be used as primary SDLC artefacts, there needs to be an execution engine for the models – say an interpreter or a transformer to (say) text format that is executable e.g. a programming language. Plus, separation of concerns leading to a cluster of relatable models indicates the need for transforming one model into another and another and so on. Therefore, it can be justifiably claimed that minimal capabilities to comprehensively address the development aspect of software engineering using model driven techniques are: A language to define all possible modeling languages, A language to define all possible visualizations of a model, A language to specify transformation of one model into another, and A language to specify transformation of a model into text artefacts. As a result, with models being the primary SDLC artefacts, software development gets transformed into language engineering endeavour wherein the focus is on defining the most intuitive and expressive modeling language[s] for a given purpose and the necessary execution machinery.

Modeling language engineering platform. Figure 1 describes a meta model for configurable extensible modeling language platform. Meta objects coloured grey can be viewed as tool implementations that can be plugged into the platform through a well- defined handshake. The platform should come with inbuilt meta meta model (which is read-only) and model-editing primitives such as InstantiateMetaObject, ConnectTwoMetaObjectsWithAnAssociation, AnnotateMetaObjectWithProperties and corresponding primitives for modification and deletion for the meta objects and associations in the meta meta model. The platforms should also provide a symbols and connectors palette which users can extend further. Primitives for mapping symbols and connectors with user-defined meta model entities should be available out- of-the-box. Platform should enable users specify constraints both at meta model and model levels. It should be possible to plug in suitable validators through well-defined handshake. To ensure easy extensibility, the platform should not make any assumptions about implementation of the tool being plugged in. The platform-to-tool handshake protocol should be externalized, well-defined and stateless. Much of the core technology to implement such a platform is already available. For instance, Eclipse [5] can provide the backbone plug-in architecture for the platform. Eclipse's eCore is a good starting point for the reflexive meta meta model. Text-based [meta] model editors can be realized with little modification, if at all, to the various model editors available. OMG QVT [16] and OMG MOFM2T [7] should suffice as specification languages for model-to-model and model-to-text transformation respectively. Both have many implementations available – licensed as well as freeware variety. In OCL [8], there exists a sophisticated declarative mechanism to specify model constraints. However, it is possible to imagine a situation where a new constraint specification language seems appropriate. Therefore, platform should have the capability to define another constraint specification and execution mechanisms. Enabling diagramming based model editors is a relatively more involved task in absence of externally stated semantics. Plus, there is dependence on presentation manager for rendering the graphics. Ideal solution, that avoids technology lock-in, is to generate platform-specific implementation of the modeling language engineering platform from its specification in model form. This may not be as far out in the future as it seems. A step in this direction has already been accomplished through model-based generation of model-based code generators [4]. Though to be used essentially by knowledgeable users the platform needs to pass minimal usability standards. A graphical user interface providing access to the relevant information through minimal 'clicks' and presenting it in uncluttered manner is the minimal requirement. Ideally, model content accompanying the platform should come organized in the form of a searchable repository. It should be possible to organize the content in the form of components ensuring modularity, high internal cohesion, and explicitly stated coupling. It would be desirable for the component abstraction to support family or software product line concept leading to compact models that can be configured easily [9, 10, 11]. However, the proposed architecture has a significant limitation in absence of a mechanism for specifying semantics (of the models), as a result, the onus of ensuring semantic correctness would be entirely on implementers of the model processing infrastructure. Thus, modeling language engineering platform of Fig. 1 can be viewed as the minimal tooling infrastructure needed for improving productivity of current MDD practitioners. Also, its existence is likely to make MDD enthusiasts to 'take the plunge' so to say. International standards do exist for the key constituents namely i) meta meta model in MOF [12], ii) model to model transformation in QVT [6], and iii) model to text transformation in MOFM2T [7]. Eclipse's plug-in architecture [5] has become defacto standard. The high level of standardization should help develop MDD community for and around the proposed platform. Development (and continuous maintenance) of the proposed platform using open source community model seems the best approach.

Need to shift the focus. Focus of modeling community as regards software-intensive systems for enterprises has so far been restricted to achieve platform independence and uniform code quality through model-based code generation. As a result, what gets modeled can at best be said as abstract description of the desired implementation of application under consideration. The very purpose of these models is automatic derivation of the desired implementation through code generation wherein the details regarding design strategies, implementation architecture and technology platform are filled in [3]. The models can also help in computing impact of a change and reflect the change into implementation with minimal side-effects thus optimizing code generation as well as testing effort. Thus, current model- driven development infrastructure is at best equivalent to a language compiler in code-centric development. However, support available there in terms of analyzers, debuggers, profilers etc is missing. The higher level of abstraction makes these shortcomings even more striking in case of model driven development. Ever-increasing penetration of internet and rapid advance of technology are subjecting enterprises to increased dynamics. As a result, enterprise IT systems need to be agile in adapting to the changes in their operating environment. For instance, business applications need to conform to new regulatory compliances such as Sarbane-Oxley [13], HiPAA [14] etc; global banks need to cater to recently opened up developing economies; insurance companies need to introduce right-fit products from time to time; financial services organization decides to cut down total IT cost through infrastructure virtualization; and so on. Only very few of these changes are crisply evident e.g. Sarbane-Oxley compliance. Majority of the changes need deep understanding of the current state of the enterprise IT systems and analysis capabilities to decide what change to introduce where and when. Having decided what needs to be done, comes the how part. However, modeling community has so far focused solely on building the right kind of business applications from their high level specifications – a subset of how part above. The harder problem of identifying what change needs to be introduced where and what would be the likely benefits of this adaptation is largely ignored. In absence of relevant information, management is left to resort to ill-informed decision making. Business- critical nature of enterprise systems means heavy price to pay for wrong decisions. It is not surprising to find that vast majority of enterprise transformation projects are either abandoned early in project life cycle or greatly exceed estimated budget and hardly ever deliver the desired returns on investment [15]. Therefore, modeling community needs to focus on what to model and not how to model. For instance, enterprise wide consistent data views is an important requirement of all enterprises. Current solution is to formulate this as a schema integration problem which adequately addresses static (or structural) aspect but forces one to rely on the program control flows for dynamic aspect. The latter is principally due to lack of suitable enterprise models. The models should help capture core properties such as 'Every credit entry must have a debit entry' in double book accounting. Then these models can be analyzed for such properties. Ability to come up with the right models and the ability to analyze them for properties of interest, we think, will provide a better handle on identifying what needs to change where. Simulation (or interpretive execution) capability of models, say on a representative input data, will help, say, get a more realistic feel of likely benefits. Thus, use of models and model-based techniques can bring more certainty to hard problems such as enterprise transformation [15].

References

1 John Hutchinson, Mark Rouncefield and Jon Whittle. Model-Driven Engineering Practices in Industry. ICSE'11 pp 633–642.
2 Hailpern, B. and Tarr, P. Model driven development: the good, the bad and the ugly. IBM Systems Journal, 2006, Vol 45, Issue 3, pp 451–461.

3 Vinay Kulkarni, R. Venkatesh, Sreedhar Reddy. Generating Enterprise Applications from Models. OOIS Workshops 2002: 270–279.
4 Vinay Kulkarni, Sreedhar Reddy. An abstraction for reusable MDD components: model-based generation of model-based code generators. GPCE 2008: 181–184.
5 Eclipse – http://www.eclipse.org
6 Query, View and Transformation – http://www.omg.org/spec/qvt
7 MOF Models to Text Transformation language – http://www.omg.org/spec/MOFM2T
8 Object Constraint Lanaguage – http://www.omg.org/spec/OCL/2.0/
9 D E Parnas. Designing software for ease of extension and contraction. ICSE 1978: 264–277.
10 Vinay Kulkarni. Raising family is a good practice. FOSD 2010: 72–79.
11 Vinay Kulkarni, Souvik Barat. Business Process Families Using Model-Driven Techniques. Business Process Management Workshops 2010: 314–325.
12 Meta Object Facility – http://www.uml.org/mof
13 The Sarbanes-Oxley act – http://www.soxlaw.com
14 HiPAA – http://www.hipaa.com
15 W B Rouse. Enterprise Transformation: understanding and enabling fundamental change. John Wiley and sons. 2006.
16 Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) – http://www.omg.org/spec/QVT/

3.20 Servicing IT Needs of Future Enterprises

Vinay Kulkarni (Tata Consultancy Services – Pune, IN)

License Creative Commons BY 3.0 Unported license
© Vinay Kulkarni

We have delivered large business applications using model driven development approach supported by home-grown standards compliant MDD toolset. In our view the basic technological pieces for supporting model-driven development are in place. Many tools with a varying degree of sophistication exist. Other important aspects such as usability, learnability, performance need to be improved which in essence is a continuous process. However, full potential of model-driven development cannot be realized in absence of ready-to-use models supported by domain ontologies providing the semantic and reasoning basis. This aspect is poorly addressed at present. Inability to translate benefits accruable due to MDE in quantitative terms seems to be a serious stumbling block for adoption of model-driven approach.

Focus of MDE community has been on developing technologies that address how to model. Barring the domain of safety-critical systems, these models are used only for generating a system implementation. Rather, modelling language design/definition is influenced very heavily by its ability to be transformed into an implementation that can be executed on some platform. Modern enterprises face wicked problems most of which are addressed in ad hoc manner. Use of modelling can provide a more scientific and tractable alternative. For which, modelling community needs to shift the focus on analysis and simulation of models. Results from random graphs, probabilistic graphical models, belief propagation and statistics seem applicable here. We believe, it is possible to model at least a small subset of modern complex enterprises so as to demonstrate that model is the organization.

3.21 Methods, not Models

Stephen J. Mellor (Evesham, UK)

License Creative Commons BY 3.0 Unported license
© Stephen J. Mellor
Main reference Pick any "methods" book you like from the Eighties and Nineties.

Model-based engineering depends, unsurprisingly, on models. What tends to be emphasized less is how we go about building those models, and how we go about determining the target into which we shall transform those models. In other words, MBE depends on methods.

A *method* is a defined, but flexible, definition of an approach to building models and their subsequent translation into something else, most importantly, executables. However, since the advent of the UML, methods have taken a back seat to notation. Latterly, with increased interest in DSLs and technology for transforming (any) models, methods have been deprecated further to the point that they have all but disappeared. (Save, perhaps, agile "methods", though these can mostly be distilled into social techniques and coding strategies.)

As a consequence, we have tools (MBE all its forms), but little or no guidance on how to use them. Improving the tools so that they produce better code or have better interfaces is clearly of value–and may even induce more people to use them–but without ways to use the tools properly so as to grind though a customer problem and truly understand it, the tools are doomed to fail. We need methods, not models.

3.22 Computer Automated Multiparadigm Modeling for the Design of Software Systems in a Physical World

Pieter J. Mosterman (The MathWorks Inc. – Natick, US)

License Creative Commons BY 3.0 Unported license
© Pieter J. Mosterman
Joint work of Mosterman, Pieter J.; Biswas, Gautam; Vangheluwe, Hans; Zander, Justyna; Denckla, Ben; Zhao, Feng; Nicolescu, Gabriela; Hamon, Gregoire; Bouldin, Don; Rucinski, Andrzej; Ghidella, Jason; Friedman, Jon
URL http://msdl.cs.mcgill.ca/people/mosterman/publications.html

To provide computational support for multiple paradigms in engineering of software systems it is essential that the semantics are well defined in a computational sense, for which semantic anchoring holds great promise. Further value derives from a theoretical formulation vs. a computational implementation. Models of a physical world often rely on abstractions and so it is essential to have these approximations well defined and analyzable so that they can be consistent within a design stage as well as between design stages. Moreover, model exchange such as via model repositories derives much benefit from computationally formulated semantics. With graphical models, the transformations between design stages as well as for semantic anchoring requires efficient graph transformation methods which in addition enables domain-specific languages. Models are very promising to (re)solve system integration issues. For system-level analyses, domain-specific languages for modeling the physics requires consideration of (i) how the physics domain constraints can be formalized, (ii) what good semantic domains are, and (iii) how the information domain interplays with the physics.

References

1. Ben Denckla and Pieter J. Mosterman. Stream- and state-based semantics of hierarchy in block diagrams. In *Proceedings of the 17th IFAC World Congress*, pages 7955–7960, Seoul, Korea, July 2008.
2. Pieter J. Mosterman. Implicit Modeling and Simulation of Discontinuities in Physical System Models. In S. Engell, S. Kowalewski, and J. Zaytoon, editors, *The 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems*, pages 35–40, Dortmund, Germany, September 2000.
3. Pieter J. Mosterman and Gautam Biswas. Formal Specifications from Hybrid Bond Graph Models. In *Proceedings of the Qualitative Reasoning Workshop*, pages 131–142, Cortona, Italy, June 1997.
4. Pieter J. Mosterman, Don Bouldin, and Andrzej Rucinski. A peer reviewed online computational modeling framework. In *Proceedings of the CDEN 2008 Conference*, CD– ROM. paper ID pmo131763, Halifax, Canada, July 2008.
5. Pieter J. Mosterman, Feng Zhao, and Gautam Biswas. An ontology for transitions in physical dynamic systems. In: *Proceedings of the National Conference on Artificial Intelligence* (AAAI-98), pages 219–224, Madison, WI, July 1998.
6. Pieter J. Mosterman, Jason Ghidella, and Jon Friedman. Model-based design for system integration. In *Proceedings of The Second CDEN International Conference on Design Education, Innovation, and Practice*, pages CD– ROM: TB–3–1 through TB–3–10, Kananaskis, Alberta, Canada, July 2005.
7. Pieter J. Mosterman and Hans Vangheluwe. Guest editorial: Special issue on computer automated multi- paradigm modeling. *ACM Transactions on Modeling and Computer Simulation*, 12(4):249–255, 2002.
8. Pieter J. Mosterman and Justyna Zander. Advancing model-based design by modeling approximations of computational semantics. In *Proceedings of the 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 3–7, Zürich, Switzerland, September 2011.
9. Gabriela Nicolescu and Pieter J. Mosterman, editors. *Model-Based Design for Embedded Systems*. CRC Press, Boca Raton, FL, 2009. ISBN: 9781420067842.
10. Justyna Zander, Pieter J. Mosterman, Gréegoire Hamon, and Ben Denckla. On the structure of time in computational semantics of a variable-step solver for hybrid behavior analysis. In *Proceedings of the 18th IFAC World Congress*, Milan, Italy, September 2011.

3.23 The Original Sin – Using Current Computers for Modeling Software

Pierre-Alain Muller (University of Mulhouse, FR)

License Creative Commons BY 3.0 Unported license
© Pierre-Alain Muller

Computers are supposed to be able to do anything, I guess, and this is why computer scientists one day got the idea that computers could be used for developing software as well. And this might be wrong.

My new thesis is that current computers are not capable of supporting the informal, unstructured, emerging, global, creative process that goes on when people think about the software which should be developed for a given (often partially unknown) purpose.

This is also the reason why programmers do not model. They are using wonderfull computerized tools for programming, and naturally stop to use computers when they "feel" that computers are not appropriate anymore.

If modeling could be automated by computers with the same kind of efficiency than programming, I'm sure that they would use computers (for modeling).

So the real issue is: "can we come up with that ideal computerized system for modeling"? (in other words, the meta-model for model-based engineering tools, which this seminar is all about).

3.24 The Other Face of Models

Leonel Domingos Telo Nóbrega (University of Madeira – Funchal, PT)

License Creative Commons BY 3.0 Unported license
© Leonel Domingos Telo Nóbrega

TModels are traditionally used in software engineering for documenting aspects of design and implementation, and, in some cases, to generate the code for the system. Though there remains debate about this type of approaches and the role, qualities and properties that models should have, the existence of models that conforms to well-defined modeling languages allows other types of uses that goes beyond the aforementioned. Visualization techniques can enable the mining of information about the models and give an innovative perspective that will contributes positively to a better understanding, analysis and validation of models.

3.25 The MetaSketch Workbench

Leonel Domingos Telo Nóbrega (University of Madeira – Funchal, PT)

License Creative Commons BY 3.0 Unported license
© Leonel Domingos Telo Nóbrega
Main reference L. Nóbrega, N.J. Nunes, H. Coelho, "The Meta Sketch Editor," in G. Calvary, C. Pribeanu, G. Santucci, J. Vanderdonckt (Eds.), Computer-Aided Design of User Interfaces V, ISBN 978-1-4020-5819-6, pp. 201–214, Springer Netherlands.
URL http://dx.doi.org/10.1007/978-1-4020-5820-2_17

The MetaSketch is a modeling language workbench that allows the rapid development of modeling languages, creating new possibilities and opportunities in the context of the Model-Based Development approach. The technology behind the MetaSketch is based on OMG standards like MOF 2.0, OCL 2.0 and XMI 2.1, and is specially tailored for creating new members of the UML family of languages. Currently, this workbench comprises a set of tools – the Editor, the Merger, the Comparer, the Visualizer and the Designer – and all of them are in a pre-beta release stage:

- MetaSketch Editor – The editor is the heart of the workbench and is simultaneously a modeling and a metamodeling editor. In a practical sense, this means that the same editor can be used to create the metamodel and the models that conforms to the created metamodel.
- MetaSketch Merger – This tool is only applicable on a metamodel and basically resolves all the Package Merge's used on the definition of the metamodel. In addition, all hierarchy of package is flattened, resulting in a metamodel with only one package that contains all language constructs definition merged.

- MetaSketch Comparer – Tool to compare easily two models described in XMI.
- MetaSketch Visualizer – This tool was originally developed for improve the navigability of modeling Editor. The base idea is to use any kind of relationship within model's elements to create views different than existing diagrams. Further, it is possible to visualize quantitative data extracted forma models through charts. To query engine to generate the data for the visualization use an Imperative OCL interpreter.
- MetaSketch Designer – This is a work in progress and will allow graphically create the definition of the concrete syntax for each language constructs and type of diagram. Currently, these definitions have to be done directly in the XML files.

3.26 Improving the Usability of Formal Verification Techniques Through Customizable Visualizations

Ileana Ober (Paul Sabatier University – Toulouse, FR)

License Creative Commons BY 3.0 Unported license
© Ileana Ober
Joint work of Ober, Ileana; Ober, Iulian; Aboussoror, El Arbi
Main reference E.A. Aboussoror, I. Ober, I. Ober, "Significantly Increasing the Usability of Model Analysis Tools Through Visual Feedback," in Proc. of the 16th Int'l SDL Forum on Model-Driven Dependability Engineering (SDL'13), LNCS, Vol. 7916, pp. 107–123, Springer, 2013.
URL http://dx.doi.org/10.1007/978-3-642-38911-5_7

One of the in hallmarks of the massive and rigorous introduction of the use of models in software development life cycles, is the openings they offer towards early model based verification and validation. During the last decades, the formal methods communities have developed interesting and powerful theoretical results that allow for advanced model checking and proving capabilities. Their usage is hindered by the complexity of information processing demanded from the modeler in order to apply them and to effectively exploit their results.

One of the research directions that we are about to explore in my research group concerns the definition of mechanisms that decrease the cognitive effort demanded from the user in order to use traditional model checking tools. This consists in defining flexible and carefully selected visual presentations of the feed-backs provided by classic model checking tools used in the context of high- level (UML and SysML) models. This ranges from simple presentation adjustments (highlighting new information or coloring things that change their values) to defining, based on feed-backs from domain experts, new diagrams (such as message exchange diagrams). Offering flexible visualization customizations opens the way to new types of scenario visualizations, improving scenario understanding and exploration. This approach was implemented in our UML/SysML analyzer and was validated in a controlled experiment that shows a significant increase in the usability of our tool, both in terms of task performance speed and in terms of user satisfaction.

Our thesis is that still a lot of effort is needed in order to make available tools and techniques allowing the user to take benefit of the use of modeling and abstraction without requiring. The results we obtained [1], in terms of tool functionalities and user evaluation, make us confident in the feasibility of this objective.

References

1 El Arbi Aboussoror, Ileana Ober and Iulian Ober. *Seeing Errors: Model Driven Simulation Trace Visualization.* In Proceedings of the 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30-October 5, 2012, LNCS 7590

3.27 Empirical Studies of Expert Software Design (Cuckoo or Canary?)

Marian Petre (The Open University – Milton Keynes, GB)

License Creative Commons BY 3.0 Unported license
© Marian Petre

Based on empirical studies of professional software developers, this talk identifies key features of the notations developers actually use for early design, and remarks, from a cognitive perspective, on fundamental challenges arising from the complex nature of 'modeling'. What software designers produce when they're unconstrained has some commonalities and emphasises movement between perspectives, between representations, between levels of abstraction, and between alternatives. Their sketches also preserve indications of provisionality. Similar characteristics – fluidity, ease of annotation, ability to change representation, expressive juxtaposition, escape from formalism, provisionality, indications of history – facilitate design discussions. One of the things that has struck me over time is that we keep finding new contexts, without quite answering the deep questions that cut across them. How can we support: reasoning across levels of abstraction; understanding the consequences of design decisions; deriving operational models; managing the tradeoff between what information a representation makes accessible and what it obscures; reasoning about interactions and emergent behaviours; preserving rationale, provenance and provisionality? From a cognitive perspective, modeling is not one activity, and addressing the needs of one constituent may be problematic for another. Moreover, possibly the most reliable result in the psychology of programming is the impact of individual differences. One size does not fit all.

3.28 Bridging the Gap Between Software Models and Analysis (Performance, Reliability, etc.) Models

Dorina C. Petriu (Carleton University – Ottawa, CA)

License Creative Commons BY 3.0 Unported license
© Dorina C. Petriu

Joint work of Petriu, Dorina C.; Alhaj, Mohammad; Tawhid, Rasha
Main reference D.C. Petriu, M. Alhaj, R. Tawhid, "Software Performance Modeling," in Proc. of the 12th Int'l School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'12), LNCS, Vol. 7320, pp. 219–262, Springer, 2012.
URL http://dx.doi.org/10.1007/978-3-642-30982-3_7

Many formalisms and tools (such as queueing networks, stochastic Petri nets, stochastic process algebras, fault trees, probabilistic time automata, etc.) for the analysis of different nonfunctional properties (NFPs) of systems and software (such as performance, reliability, availability, scalability, security, etc.) have been developed over the years. Traditionally, analysis models used to be built "by hand" by the analyst. More recently, in the MBE context, the following approach for the analysis of different NFPs has emerged in literature: a) extend the software model to be analyzed with annotations specific to the respective NFP; b) apply a model transformation to derive an analysis model expressed in the selected formalism from the annotated software model; c) analyze the NFP model using existing solvers; and d) interpret the results and give feedback to the software developers.

In the case of UML-based software development, the extensions required for NFP-specific annotations are defined as UML profiles, which have the advantage to be processed by standard UML tools without any change in the tool support. For instance, two standard UML profiles provide, among other features, the ability to define performance annotations:

the UML Profile for Schedulability, Performance and Time (SPT) defined for UML 1.X versions (OMG, 2005) and the UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) defined for UML2.X versions (OMG, 2009).

An important research challenge is to hide, as much as possible, the analysis formalism and the complexity of the analysis models from the software developers, while at the same time providing analysis results and advice for improvement in the context of the software model. A list of research questions is as follows:

- Model transformation from software model to analysis model: keep the separation of concerns between the platform-independent model (PIM) of the software under development and the underlying platform model. Platform modeling requirements are: provide support for reusable, generic, parametric platform models; provide support for flexible deployment models (because resource allocation is important for many NFPs).
- Bridging the gap between the software model and the analysis model trough trace-links between the elements of the two models. Such trace-links will be used, for instance, for providing analysis feedback to software developers (analysis results, advice).
- Integrate multiple NFP analyses in the software development process (e.g., performance, reliability and security): a) For each NFP, explore the parameter space for different design alternatives, configurations, workload parameters, etc., in order to find the "best solution"; b) Toolset should provide support for an experiment controller for exploring the parameter space; c) Software process issue: how integrate the evaluation of multiple NFP in the software development process.
- Tool interoperability: Input and output format of the analysis tools (many created a long time ago) are tool specific and often unfriendly to automated processing.
- Impact of software model changes on the NFP analysis: it is desirable to be able to propagate changes incrementally, rather than re-building the analysis models from scratch every time the software model is changed.

3.29 The Need for Scalable and Flexible Model-Based Engineering

Louis Rose (University of York, GB)

License Creative Commons BY 3.0 Unported license
© Louis Rose

I briefly summarised recent work in my research group on improving the scalability of modelling tools and techniques, and on increasing the degrees of flexibility in modelling tools and techniques. I argued in favour of better supporting iterative and incremental development in model-based engineering tools by, for example, favouring dynamic and weakly typed environments. Finally, I spoke about the way in which I believe that model-based engineering does and does not address issues of software maintenance, which remains one of the most costly elements of software engineering.

3.30 EuGENia Live: A Flexible Graphical Modelling Tool

Louis Rose (University of York, GB)

License Creative Commons BY 3.0 Unported license
© Louis Rose
Joint work of Rose, Louis; Kolovos, Dimitrios S.; Paige, Richard F.
Main reference L.M. Rose, D.S. Kolovos, R.F. Paige, "EuGENia Live: A Flexible Graphical Modelling Tool," in Proc. of the Extreme Modeling Workshop (XM'12), pp. 15–20, ACM, 2012.
URL http://dx.doi.org/10.1145/2467307.2467311
URL http://www.di.univaq.it/diruscio/sites/XM2012/xm2012_submission_6.pdf

Designing a domain-specific language is a collaborative, iterative and incremental process that involves both domain experts and software engineers. Existing tools for implementing DSLs produce powerful and interoperable domain-specific editors, but are resistant to language change and require considerable technical expertise to use. I presented EuGENia Live, a tool for rapidly prototyping and designing graphical DSLs. EuGENia Live runs in a web browser, supports on-the-fly metamodel editing, and produces DSLs that can be exported and used to produce an initial implementation in the Eclipse Modeling Framework. EuGENia Live draws on existing work from the fields of flexible modelling and model-driven engineering.

3.31 Compositional Model Based Software Development

Bernhard Rumpe (RWTH Aachen, DE)

License Creative Commons BY 3.0 Unported license
© Bernhard Rumpe
URL http://monticore.de

Model based software development promises to strongly improve efficiency and quality in software development projects. However, MBSE has not yet delivered it's promises yet.

In the talk, we examine the current state and problems of MBSE and discuss a number of approaches to tackle those. In particular, we discuss how to make use of models in large development projects, where a set of heterogenous models of different languages needs is developed and needs to fit together.

A model based development process (both with UML as well as a domain specific modeling language (DSML)) heavily relies on modeling core parts of a system in a redundant free form, having compositional generators to early and repeatedly cut code and tests from these models.

We in detail discuss compositionality on models and heterogeneous modeling languages and how it supports agile development as well as reuse of language and tooling infrastructures.

And we show what the current status of compositionality is vs. a bunch of interesting languages given.

We finally demonstrate what has already been achieved in the language workbench MontiCore developed in our group over the recent years.

3.32 Formal Verfication and Model-Based Engineering

Martina Seidl (University of Linz, AT)

License Creative Commons BY 3.0 Unported license
© Martina Seidl

As in traditional hardware and software system development, also in model-based engineering formal techniques find many applications within the development process in order to verify that certain properties of the system under development hold. In the context of the Austrian research project FAME (supported by the Vienna Science Fund under grant ICT10-018; www.modelevolution.org), we currently investigate how different verification techniques may be used to support the evolution process of software models. Therefore, we use various reasoning engines like SAT solvers and model checkers not only to verify that evolution does not violate any constraints but also for calculating correctly evolved models. For example, the satisfiability problem of propositional logic is used to encode consistent merging of concurrently evolved versions of one sequence diagram with respect to a given set of state machines [1]. The SAT solver is used to suggest the modeler a set of correct solutions from which the most convenient approach may be selected. During this work, we encountered several challenges. First of all, we need a concise semantics in order to specify the verification problem. Having an exact formulation of the verification problem, the encoding in the respective formalism almost directly derivable. Second, there is a huge gap between the representation in the modeling language and the input of the verification tool. For example, in the case of a SAT solver, the problem is encoded as a sequence of numbers. In order to make the output interpretable for humans, it has to be translated back to the model representation. In order to highlight the solutions found by the SAT solver, we annotate the solution with colors for directing the human modeler's attention to the particular point in the model where modifications have been performed in order to ensure the consistency. The third major challenge involves the testing of our approach. On the one hand we took manually engineered models from the literature. These models allowed us to test the basic functionality of our approach, but they were too small to evaluate the performance of our approach. Therefore, we implemented a random generator which allows us to generate suitable input models of arbitrary size. These models showed not only valuable to improve the performance of our approach, but it helped us also to find bugs in our implementation which were not triggered by the manually engineered test models. Besides investigating how formal verification techniques support various evolution tasks in model-based engineering, we are also investigating how models can support the development of verification backends like SAT solvers. A model-based testing approach [2] shows to be be extremely valuable for building modern SAT solvers, which consists of strongly optimized code (usually in C) in order to be able to tackle verification problems of practical size. Such systems are too complex to be verified and therefore, much emphasize has to spend on testing their correctness. We propose to describe correct sequences of API calls, input data, and option configurations in terms of state machines which may then be used for grammar-based blackbox fuzzing as well as for delta debugging. Whereas the former technique generates the test cases, the latter reduces an error triggering test case such that manual debugging is feasible. By this means an automatic test chain is realized. In future work, we plan to extend this approach to other solving systems like SMT solvers. To sum up, both model-based engineering techniques and formal verification techniques can benefit from each other and an exchange of ideas and approaches of the different fields may bring valuable insights and advances to both research fields.

References

1. Magdalena Widl, Armin Biere, Petra Brosch, Uwe Egly, Marijn Heule, Gerti Kappel, Martina Seidl, Hans Tompits. Guided Merging of Sequence Diagrams. SLE 2012. 164–183.
2. Cyrille Artho, Armin Biere and Martina Seidl. Model-Based Testing for Verification Backends. Accepted for Tests and Proofs (TAP) 2013.

3.33 MBE Tools: Effective Automation Where It Helps

Bran V. Selic (Malina Software Corp. – Nepean, CA)

License Creative Commons BY 3.0 Unported license
© Bran V. Selic
Main reference B. Selic, "What will it take? A view on adoption of model-based methods in practice," Journal of Software and System Modeling, October 2012, Volume 11, Issue 4, pp. 513–526.
URL http://dx.doi.org/10.1007/s10270-012-0261-0

Increased levels of computer-supported automation are considered as one of the key enablers to the higher levels of productivity and product quality promised by model-based engineering (MBE) . However, practical experience with present-day MBE tools indicates that we are still far from this ideal. In fact, if anything, these tools are proving to be an impediment to productivity; they are complex, difficult to learn, and difficult to use. Users of these tools often find themselves in frustrating situations where the tools are blocking their progress, forcing them into workarounds and constrained operating modes and procedures which are not conducive to free expression of ideas.

Much of this can be attributed to the tool designers' poor understanding of their users, how they work and what motivates them. There is often a naïve and implicit image of a "typical user", based on very shallow analysis and a distressing lack of awareness of the complexities involved in human-machine interaction. (This is a problem even when tool developers are users of their own tools – something that, unfortunately, does not happen often enough for MBE tools – since in those cases developers often lose sight of the fact that their users do not have the same deep understanding of a tool's architecture as they do.) In such settings, usability is rarely considered an architectural-level concern, but as something that is merely a presentation issue to be solved by a 'suitable" user interface.

To get the full benefits that we expect from the automation potential of MBE tools, we must not only understand how and where it should be applied, but equally importantly, knowing where it is inappropriate. (Just because something can be automated does not mean it should be.) A key research issue related to this is finding ways to support the transition between an informal and provisional mode of operation, which is conducive to design exploration, and the formal mechanistic world required for computer-based design capture. Ideally, this should not be a one-time one-way transition, which could lead to premature commitment, but a continuous back-and-forth interplay of these modes as design progresses.

3.34 Models in Software Engineering

Perdita Stevens (University of Edinburgh, GB)

License Creative Commons BY 3.0 Unported license
© Perdita Stevens

I gave an abbreviated version of a talk I have given in Edinburgh in the series of Hamming Seminars, whose remit is to discuss what were the major problems in a given field. Here is the abstract for the original talk.

A model is (for purposes of this talk) an abstract, usually graphical, representation of some aspect of a software-intensive system. Software engineering has, since its invention, involved the use of models. In recent decades, a perceived need for greater automation of processes in software engineering, the wish to develop software faster and more cheaply, and the drive for higher quality have motivated ever more reliance on models. Languages for modelling and techniques for working with models have struggled to keep pace, hindered in some cases by conflicting requirements arising from the different motivations. We are currently in a dangerous state in which models sometimes endanger rather than support the developers' aims. I will explain some of the problems, and discuss the progress that I think we will/may/should/must/cannot expect to see in the coming years

3.35 Tool Building Using Language Composition

Laurence Tratt (King's College London, GB)

License Creative Commons BY 3.0 Unported license
© Laurence Tratt
URL http://soft-dev.org/

Language composition allows users to mix languages together in a fine-grained manner. It can complement modelling by giving the possibility to mix together modelling and non-modelling languages. We presented a new language composition editor that allows arbitrary language composition to feel like using a normal text editor. It also allows textual and non-textual languages to be composed and edited together.

3.36 Generic Tools, Specific Languages

Markus Völter (Völter Ingenieurbüro – Heidenheim, DE)

License Creative Commons BY 3.0 Unported license
© Markus Völter

In my talk, and in the mbeddr tool demo, I introduced an new approach to developing domain-specific software engineering tools. It promises a significant increase in the fidelity of adapting generic tools to specific domains, while significantly reducing the effort of the adaptation. The approach, called Generic Tools, Specific Languages, shifts the focus from the engineering *tool* to the underlying *languages*. Recent advances in language engineering – and the language workbenches that support these advances – enable the modular, incremental extension of languages. Projectional editing supports increased notational freedom, such as tables or mathematical symbols. These two ingredients enable the shift in focus for developing domain-specific engineering tools: a *generic tool*, the language workbench, hosts arbitrary

languages and provides IDE support such as syntax coloring, code completion, go-to-definition and find-references as well as model search, refactoring, debugging or visualization. On top of this platform, a set of very *specific languages* adapt the tool to a given domain. Some actual tool adaptations (windows, buttons, context menus) are sometimes necessary, but they are few and far between and are always related to languages or language extensions. mbeddr (http://mbeddr.com) is an example implementation of this approach for embedded software engineering. It builds on top of the JetBrains MPS language workbench (http://jetbrains.com/mps)

3.37 The Tool's The Thing

Jon Whittle (Lancaster University, UK)

License Creative Commons BY 3.0 Unported license
© Jon Whittle

In this talk, I review literature on tools from psychology, sociology, philosophy, management studies and computer science. Tool use has been studied for centuries and MDE researchers should try, as much as possible, to take such literature into account when developing new MDE tools. The talk also presents a recent study by Whittle, Hutchinson, Rouncefield, Burden and Heldal, which carried out 40 interviews will companies applying MDE tools in practice. Interview data was analyzed using a grounded approach to identify emergent themes related to MDE tool use. The result is a taxonomy of 30 technical, social and organizational factors of MDE tools which affect their adoption in practice.

3.38 Flexible, Lightweight Metamodeling

Dustin Wüest (Universität Zürich, CH)

License Creative Commons BY 3.0 Unported license
© Dustin Wüest
Joint work of Wüest, Dustin; Seyff, Norbert; Glinz, Martin
Main reference D. Wüest, N. Seyff, M. Glinz, "FlexiSketch: A Mobile Sketching Tool for Software Modeling," in D. Uhler, K. Mehta, J.L. Wong, (Eds.), Mobile Computing, Applications, and Services, LNICST, Vol. 110, pp. 225–244, Springer, 2013.
URL http://dx.doi.org/10.1007/978-3-642-36632-1_13

Current metamodeling tools provide the technical means for constructing all kinds of domain specific languages (DSLs), but they have two disadvantages. First, they are hard to learn and require experts in both the problem domain and metamodeling. Second, they neglect the processes used in practice to come up with good DSLs. Tools are not adjusted to the methods how practitioners create DSLs. Creators of metamodeling tools assume that the users know what they are doing and have the completed DSLs already in their minds when they start to use the tools. In contrast, creating good DSLs is often an iterative process, where engineers interleave between constructing a DSL and creating model examples of it. These models can help to show weak spots in a DSL, and the DSL evolves over time.

We argue for the need of flexible metamodeling tools that enable users with only basic metamodeling knowledge to incrementally build DSLs based on concrete model examples. Domain experts should be able to create modeling languages step by step and repeatedly test them by drawing example models. This also implies that tools must support different

level of details for language definitions and do not require a complete language description before it can be tested in a productive environment. At the beginning, it must be possible to define individual elements of a modeling language at different abstraction levels, depending on how much detail is needed or already known. Metamodeling tools should foster creativity and allow trying out different design ideas by providing fast feedback cycles. To achieve this, it must be possible to quickly change a metamodel and try out the new version. It helps if a model editor is generated or updated on the fly.

For being able to come up with better metamodeling support in practice, we have to know i) who the typical users are, ii) how much they know about metamodeling, iii) what methods and processes they use to perform metamodeling in practice, and iv) what their requirements for metamodeling tools are. In addition, we should exploit the possibilities that a tool-supported approach gives for guiding less experienced users in performing metamodeling.

3.39 Liberating Software Engineers from the Tyranny of a Strict Modeling Language

Dustin Wüest (Universität Zürich, CH)

License Creative Commons BY 3.0 Unported license
© Dustin Wüest
Joint work of Wüest, Dustin; Seyff, Norbert; Glinz, Martin
Main reference D. Wüest, N. Seyff, M. Glinz, "Flexible, lightweight requirements modeling with FlexiSketch," in Proc. of the 20th IEEE Int'l Requirements Engineering Conf., pp. 323–324, IEEE, 2012.
URL http://dx.doi.org/10.1109/RE.2012.6345826

Creativity and discussing different design alternatives play central roles in early phases of software development. Many modeling tools do not provide adequate support for these phases, because they restrict users in utilizing a particular modeling language and level of detail. Having to think about how to express ideas in a particular language while coming up with the ideas hinders the creative flow. Also, different ideas may be expressed at different levels of detail, while a modeling language typically only supports one level of detail. Therefore, whiteboards and flip charts are still dominant tools for early software design phases. Engineers have to accept the drawback of manually re-creating the information from a whiteboard in a software modeling tool if they want to re-use and refine the information.

We presented FlexiSketch, a flexible and lightweight modeling tool for mobile devices. The tool allows freeform sketching and the creation of nodes-and-edges diagrams. Modeling and metamodeling can be performed in a single environment. Engineers do not have to adhere to a predefined modeling language. Instead, they can quickly sketch different design ideas. Metamodel information can be added to individual diagram elements on demand, which enables a step-wise formalization and refinement of model sketches. This also means that domain specific languages can be built in an iterative, incremental way. Short feedback cycles foster creativity and invite to try out different design alternatives.

3.40 Design and Design Thinking

André van der Hoek (University of California – Irvine, US)

License Creative Commons BY 3.0 Unported license
© André van der Hoek
Main reference M. Petre, A. van der Hoek, (Eds.), "Software Designers in Action: A Human-Centric Look at Design Work," Chapman and Hall/CRC, forthcoming Sep 2013.
URL http://sdcl.ics.uci.edu/research/calico

Model-driven engineering requires software developers to design. A large body of literature concerns itself with design thinking – ways in which designers work through a design problem. To date, this literature has not been examined from the perspective of software design. This talk makes the case for taking a design thinking look at software design, and highlights some of the dimensions of what such an exploration would look like.

4 Overview of Working Groups

4.1 Working Group A: Summary of Discussions and Conclusions

Bran V. Selic (Malina Software Corp. – Nepean, CA)

License Creative Commons BY 3.0 Unported license
© Bran V. Selic
Joint work of Larry Constantine, Robert B. France, Martin Glinz, Lars Hamann, Steven Kelly, Thomas Kühne, Vinay Kulkarni, Stephen J. Mellor, Pierre-Alain Muller, Leonel Domingos Telo Nóbrega, Marian Petre, Dorina C. Petriu, Louis Rose, Bran V. Selic, Perdita Stevens, André van der Hoek, Jon Whittle, Dustin Wüest

The participants agreed to separate the issues to be addressed into two broad categories. One category dealt primarily with technical issues related to modeling and modeling tools. The second category addressed topics related to the use of modeling tools in context. This included a wide spectrum of concerns, from recognizing end-user business needs, to understanding how domain experts do design, to identifying and overcoming tool usability issues. Fortunately, among the participants were a number of researchers who specialized in studying software design practices and usability, who helped provide the necessary multi-disciplinary perspective.

A general consensus quickly emerged that the flaws of the current generation of rigidly conceived modeling tools were standing in the way of agile design, and thereby impeding broader adoption of models and modeling in industrial practice. One of the root causes of this seems to be lack of support for provisionality in design, that is, the ability not only to tolerate but to actively support the type of informal processes and ambiguous artifacts produced during design exploration. Case studies of industrial projects have shown that design involves a myriad of aspects that are not easily replicated by computer-based automation, including non-verbal communications, analogies, and the use of ad hoc informal notations and terms. Collaboration is a key ingredient in such processes that should be supported by tools, but without imposing undue structure that blocks active discourse. This raises the issue of understanding what should (and what should not) be automated in this process and, if so, how? A closely related challenge is determining how best to transition back and forth between the informal and highly flexible world of creative thinking and the much more formal world of design capture and implementation, or between different purposes of models, such as descriptive and prescriptive uses of models.

From these considerations, the notion of tools that are "fit for purpose" emerged as a useful formulation and desirable objective. This, of course, implies a highly human-centric

approach to the design of such tools. There are many different uses of and many different perspectives on models that must be accommodated. One way of achieving this is to focus less on users per se and more on their activities, purposes, and expertise levels (expert users work in very different ways from novices). This means highly adaptable and customizable tools, allowing users to organize them according to the task and situation at hand, rather than based on some preconceived hardwired workflow or, worse yet, on some automated inference strategy built into the tool (experience has shown that such inferences are often wrong and annoying). Workflows should not be interrupted by petty and inflexible formal rules.

Finally, an additional topic discussed in this context was how to create and popularize a modeling culture among software practitioners by teaching the value and use of models early in software engineering curricula.

At the end of the workshop, a number of key research questions were identified that related to the issues described above (NB: many of these questions are closely related and even overlap):

1. Investigate the transitions that occur in model-based development (e.g., between the informal and formal domains, between models for different purposes, between models at different levels of abstraction) and how this can be supported by tools.
2. Identify the various modeling purposes and what would be required to construct corresponding "fit for purpose" tools.
3. Study how to specify and manage the relationships between different models intended for different purposes.
4. Understand how a model intended to support one purpose can be retargeted for a different purpose.
5. Understand the actual and potential end-user value propositions and if and how they can be supported by models and modeling.
6. Collecting evidence of the value of models in practice (industry and research).
7. Study what to (and what not to) automate in modeling. (When? Why?)
8. How can tools help us capture and exploit the provisionality inherent in design.
9. Investigate ways of fostering new generations of model-aware practitioners. (How should we teach modeling? How do we make it relevant?)
10. Understand how model-based engineering fits into modern agile development processes.
11. Investigate and clarify the differences (social, cultural, technical, etc.) between practitioners of "traditional" software development methods and those using models and modeling; Based on that understand why so many practitioners resist model-based methods.

4.2 Working Group B.1: Composition Issues in MBE Languages and Tools

Tony Clark (Middlesex University, GB)

License Creative Commons BY 3.0 Unported license
© Tony Clark
Joint work of Colin Atkinson, Tony Clark, Benoit Combemale, Lukas Diekmann, Øystein Haugen, Gabor Karsai, Steven Kelly, Vinay Kulkarni, Stephen J. Mellor, Pieter J. Mosterman, Dorina Petriu, Bernhard Rumpe, Laurence Tratt, Markus Völter.

The hypothesis of group B.1 is that the problem of complexity (and therefore usability) of Model Based Engineering tools can be addressed through *composition*. Methods and technologies for modularity and composition can help by providing a basis for reuse of tool elements and a basis for understanding otherwise highly complex technologies.

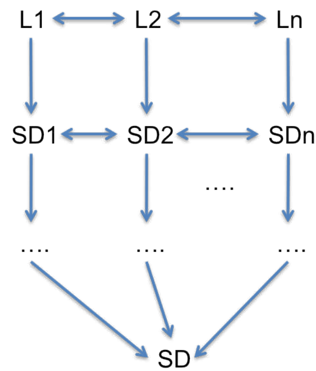


Figure 3 Finding a Common Semantic Domain.

Language Dependencies

Syntactic Mixing

	YES	NO (requires glue)
YES	Full Language Interleaving	Embedding One Language In Another.
NO	References	Reuse through a third party (e.g. a mapping model)

Figure 4 Categories of Language Composition.

Syntax	Includes the abstract and concrete syntax, graphical, textual, mixed.
Semantics	Includes models of computation, denotation, translational, mathematical, logic,
Implementations	Code generators, interpreters.
Analytics	Includes the type system, well-formedness rules, language properties and their definitions, relationships to analysis tools.
Tools	Editors, debuggers, version control, life-cycle management.
Usage	Guidelines, development process, documentation, life-cycle.

Figure 5 A Proposal for a Composable Language Module.

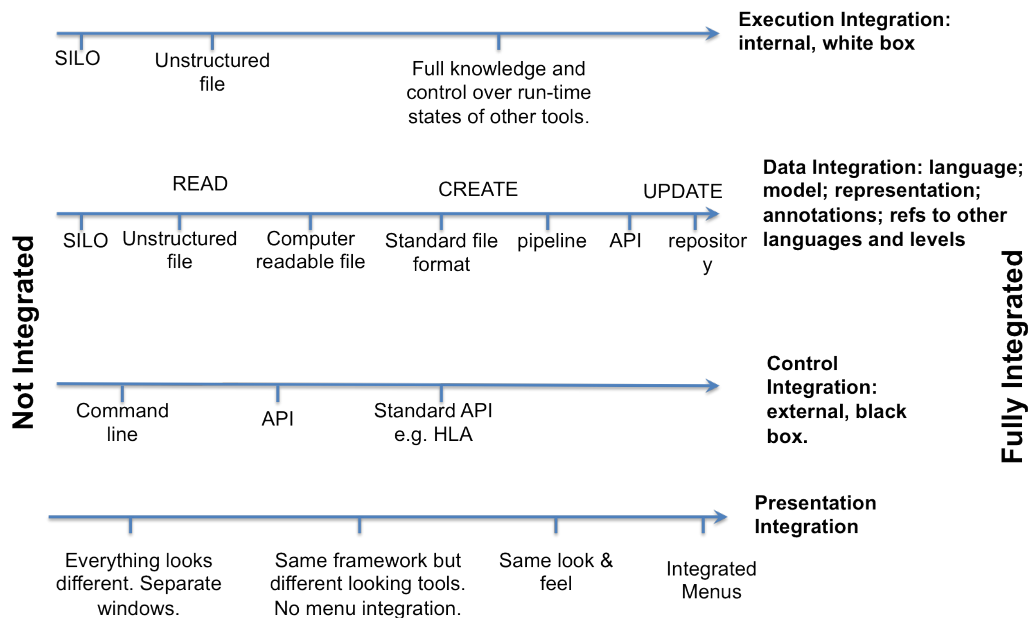


Figure 6 A Spectrum of Tool Composition.

The group identified four different facets of composition that were discussed over several sessions leading to the identification of a series of key research questions:

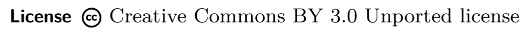
Semantic Heterogeneity arises because MBE languages and tools often consist of multiple sub-components each of which has a specific semantics all of which must be made consistent in order for the tools to work together. Figure 3 (proposed by Bernhard Rumpe) shows an approach to composition which seeks to identify a single common semantic domain `SD` through multiple refinement transformations of individual language semantic domains. The group also discussed whether it would be possible to produce a small number of agreed models of computation that are reused as a semantic basis for modelling languages. By agreeing such a set of domains, maturity and well-understood relationships between them would evolve over time.

Language Composition often occurs when a tool offers the ability to model multiple system aspects, and is key to language reuse. The group spent time trying to identify different types of language composition and produced the categorization shown in Figure 4 (proposed by Markus Völter) where languages can be integrated to different levels. The group also discussed what an ideal composable language module might look like as shown in Figure 5 together with a likely set of composition operators including union and embedding.

Tool Composition occurs when building tool-chains or reusing modular tool components. Several aspects of tool composition were identified and described as a spectrum of integration from *not integrated* to *fully integrated* as shown in Figure 6 (proposed by Steven Kelly).

4.3 Working Group B.2: Research Questions for Validation and Verification in the Context of Model-Based Engineering

Martin Gogolla (Universität Bremen, DE)

License  Creative Commons BY 3.0 Unported license
© Martin Gogolla
Joint work of Catherine Dubois, Michalis Famelis, Martin Gogolla, Leonel Nóbrega, Ileana Ober, Martina Seidl, Markus Völter

Notions 'Validation' and 'Verification' (V&V): The goal of the using validation and verification techniques is to better understand models and to uncover model properties which are stated implicitly in the model (see Fig. 7). Validation and verification serves to inspect models and to explore modeling alternatives. Model validation answers the question 'Are we building the right product' whereas model verification answers 'Are we building the product right'. Validation is mainly an activity by the developer demonstrating model properties to the client, whereas in verification the developer uncovers properties relevant within the development process. Verification covers testing as well as automatic and semi-automatic proving techniques.

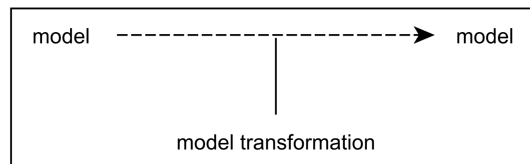


Figure 7 Validation and Verification of Models and Transformations.

Relationships between Client, Developer and V&V Engine: How do we express properties at the level of models in a way understandable to clients? How do we formulate models and properties in a single language transparent to clients? How do we report the validation and verification results and diagnostics in an appropriate form to clients? How do we bridge the gap between formally expressed and verified properties on one side and client attention on the other side? Can modeling language extensions help in making explicit the needs of validation and verification machines?

Design VS Time-Runtime: How do we obtain during the validation and verification phase an initial model instanciation on the model runtime level which is determined by the model design time description? How do we obtain consequent runtime instanciations? How do we connect design time and runtime artifacts? How do we deal with the scalability issue in the context of validation and verification tools? How do we handle time and space concerns wrt design time and runtime artifacts? How do we automatically or semi-automatically manage the validation and verification machine configuration?

Model Transformation: What verification techniques are meaningful for verifying model transformations? How do we analyse properties like confluence and termination for transformations which are composed from transformation units? How do we analyse

correctness of model transformations wrt a transformation contract? How do we infer a transformation contract from a model transformation?

Methods: How do we integrate validation and verification in the overall development and modeling process? On the technical level of tool exchange? On the methodological level of using the right technique at the right time for the right task? When are techniques like animation, execution, symbolic evaluation, testing, simulation, proving or test case generation used efficiently during development (see Fig.8)? For which model and model transformation properties can they be employed?

Applicability of V&V techniques for stakefolder (client, developer) understanding

	Client-Developer	Developer-Developer
Animation	+	
Testing	+	+
Execution	+	+
Simulation	+	+
Symbolic execution	(+)	+
Proving	(+)	+
Test case generation		+

+: applicable, (+): restricted applicable for domain-specific, explainable properties

Figure 8 V&V Techniques and their Relationship to Stakeholders.

Informal VS Formal VS Incomplete Modeling: How do we leverage informal assumptions found in sketches for exploratory validation and verification? Are informal sketches close enough to validation and verification at all? What are appropriate relaxation mechanisms for different degrees of formality? How do we handle incomplete or partial models wrt validation and verification? How do we deactivate and activate model units? How do we handle the exploration of model properties and alternatives?

Comparison and Benchmarking: How do we compare existing validation and verification tools employed for modeling wrt functionality, coverage, scalability, expressiveness, executing system (i.e., for models at runtime)? Which criteria are appropriate for comparison? Can the broad and diverse spectrum of validation and verification machines (like B, Coq, HOL/Isabelle, SAT, SMT, CSP solvers, Relational logic and enumerative techniques) be globally compared in a fair way at all?

Properties: How do we handle model and model transformation properties relevant in validation and verification like consistency, reachability, dependence, minimality, conformance, safety, liveness, deadlock freeness, termination, confluence, correctness? How do we search for such properties in models and model transformations? What are the benefits and tradeoffs between expressing these properties on more abstract modeling levels in contrast to expressing them on more concrete levels? How do we find the right techniques for uncovering static and dynamic model properties? Which techniques are appropriate for uncovering static modeling language inherent properties, which for static model-specific properties? Which techniques are appropriate for uncovering dynamic generic properties, which for dynamic model-specific properties? Which high-level features are needed in the property description language in order to query and to determine modeling level concepts?

Participants

- Colin Atkinson
Universität Mannheim, DE
- Tony Clark
Middlesex University, GB
- Benoit Combemale
IRISA – Rennes, FR
- Larry Constantine
University of Madeira – Funchal, PT
- Lukas Diekmann
King's College London, GB & University of Konstanz, DE
- Catherine Dubois
ENSIIE – Evry, FR
- Michalis Famelis
University of Toronto, CA
- Robert B. France
Colorado State University, US
- Martin Glinz
Universität Zürich, CH
- Martin Gogolla
Universität Bremen, DE
- Lars Hamann
Universität Bremen, DE
- Øystein Haugen
SINTEF – Oslo, NO
- Gabor Karsai
Vanderbilt University, US
- Steven Kelly
MetaCase – Jyväskylä, FI
- Thomas Kühne
Victoria Univ. – Wellington, NZ
- Vinay Kulkarni
Tata Consultancy Services – Pune, IN
- Stephen J. Mellor
Evesham, UK
- Pieter J. Mosterman
The MathWorks Inc. – Natick, US
- Pierre-Alain Muller
University of Mulhouse, FR
- Leonel Domingos Telo Nóbrega
Univ. of Madeira – Funchal, PT
- Ileana Ober
Paul Sabatier University – Toulouse, FR
- Marian Petre
The Open University – Milton Keynes, GB
- Dorina C. Petriu
Carleton Univ. – Ottawa, CA
- Louis Rose
University of York, GB
- Bernhard Rumpe
RWTH Aachen, DE
- Martina Seidl
University of Linz, AT
- Bran V. Selic
Malina Software Corp. – Nepean, CA
- Perdita Stevens
University of Edinburgh, GB
- Laurence Tratt
King's College London, GB
- André van der Hoek
Univ. of California – Irvine, US
- Markus Völter
Völter Ingenieurbüro – Heidenheim, DE
- Jon Whittle
Lancaster University, UK
- Dustin Wüest
Universität Zürich, CH