# Towards a Unified View of Modeling and Programming (Track Summary)

Manfred Broy[1], Klaus Havelund[2⋆], Rahul Kumar[3], and Bernhard Steffen[4]

[1] Technische Universität München, Germany
[2] Jet Propulsion Laboratory, California Inst. of Technology, USA
[3] Microsoft Research, USA
[4] TU Dortmund University, Germany.

## 1 Motivation and Goals

Since the 1960s we have seen tremendous amount of scientific and methodological work in the fields of specification, design, and programming languages. In spite of the very high value of this work, however, this effort has found its limitation by the fact that we do not have a sufficient integration of these languages, as well as tools that support the development engineer in applying the corresponding methods and techniques. A tighter integration between specification and verification logics, graphical modeling notations, and programming languages is needed.

In a (possibly over) simplified view, as an attempt to impose some structure on this work, we can distinguish between three lines of work: formal methods, model-based engineering, and programming languages. Formal methods include, usually textual, formalisms such as VDM, CIP, Z, B, Event-B, ASM, TLA+, Alloy, and RAISE, but also more or less automated theorem proving systems such as Coq, Isabelle, and PVS. Such formalisms are usually based on mathematical concepts, such as functions, relations, set theory, etc. A specification typically consists of a signature, i.e. a collection of names and their types, and axioms over the signature, constraining the values that the names can denote. A specification as such denotes a set of models, each providing a binding of values to the names, satisfying the axioms. Such formal methods usually come equipped with proof systems, such that one can prove properties of the specifications, for example consistency of axioms, or that certain theorems are consequences of the axioms. A common characteristic of these formalisms is their representation as text, defined by context-free grammars, and their formalization in terms of semantics and/or logical proof systems. In parallel one has seen several model checkers appearing, such as SPIN, SMV, FDR, and UPPAAL. These usually

⋆ The research performed by this author was carried out at Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

prioritize efficient verification algorithms over expressive and convenient specification languages. Exceptions are more recent model checkers for programming languages, including for example Java PathFinder (JPF).

Starting later in the 1980s, the model-based engineering community developed graphical formalisms, most prominently represented by UML and later SysML. These formalisms offer graphical notation for defining data structures as "nodes and edge" diagrams, and behavioral diagrams such as state machines and message sequence diagrams. These formalisms specifically address the ease of adoption amongst engineers. It is clear that these techniques have become more popular in industry than formal methods, in part likely due to the graphical nature of these languages. However, these formalisms are complex (the standard defining UML is much larger than the definition of any formal method or programming language), are incomplete (the UML standard for example has no expression-language, although OCL is a recommended add-on), and they lack commonly agreed up semantics. This is not too surprising as UML has been designed on the basis of an intuitive understanding of the semantics of its individual parts and concepts, and not under the perspective of a potential formal semantics ideally covering the entire UML. This leaves users some freedom of interpretation, in particular concerning the conceptual interplay of individual model types and often leads to misunderstandings, but it has still been sufficient in practice in order to support tool-based system development, even by providing, e.g., partial code generation. On the other hand, it is also responsible for the only very partial successes of the decades of attempts to provide formal semantics to UML. One may, therefore, argue that (the abstract syntax and intuitive semantics of) UML, as it stands, is not adequately designed to support a foundation in terms of a formal semantics. It would therefore be interesting to reconsider the design of UML with the dedicated goal to provide a formal semantics and thereby reach a next level of maturity.

Finally, programming languages have evolved over time, starting with numerical machine code, then assembly languages, and transitioning to higher-level languages with FORTRAN in the late 1950s. Numerous programming languages have been developed since then. The C programming language has since its creation in the early 1970s conquered the embedded software world in an impressive manner. Later efforts, however, have attempted to create even higher-level languages. These include language such as Java and Python, in which collections such as sets, lists and maps are built-in, either as constructs or as systems libraries. Especially the academic community has experimented with functional programming languages, such as ML, OCaml, and Haskell, and more recently the integration of object-oriented programming and functional programming, as in for example Scala.

Each of the formalisms mentioned above have advantageous features not owned by other formalisms. However, what is perhaps more important is that these formalisms have many language constructs in common, and to such an extent that one can ask the controversial question: *Should we strive towards a unified view of modeling and programming?* It is the goal of the meeting to discuss the relationship between modeling and programming, with the possible objective of achieving an agreement of what a unification of these concepts would mean at an abstract level, and what it would bring as benefits on the practical level. What are the trends in the three domains: formal specification and verification, model-based engineering, and programming, which can be considered to support a unification of these domains. We want to discuss whether the time is ripe for another attempt to bring things closer together.

## 2 Contributions

The paper contributions in this track are introduced below. The papers are ordered according to the sessions of the track: (1) opinions, (2) more concrete, (3) meta-level considerations, (4) domain-specific approaches, (5) tools and frameworks view, and (6) panel. Within each track the papers are ordered to provide a natural flow of presentations.

### 2.1 Opinions

Selic [16] (*Programming* $\subset$ *Modeling* $\subset$ *Engineering*) takes the position that models and modeling have a much broader set of purposes than just programming. It points out that there is a direct conflict between modeling and programming, as modeling is based on abstracting away irrelevant details whereas programming requires full implementation oriented details. In the end, the paper investigates the question of the complex relationship between modeling and programming. Finally it comes up with the question of whether modelers can become programmers, and it concludes that it has to deal with the question whether high level modeling languages can be used as implementation languages.

Seidewitz [15] (*On a Unified View of Modeling and Programming – Position Paper*) considers a unified view of modeling and programming. Seidewitz takes the position that some software models specify behavior precisely enough that they can be executed, and that all programs can be considered models, at least of the execution they specify. He concludes that modeling and programming are actually not so different after all, and there might be conversions. He claims that the language design legacy of UML is largely grounded on the old view of sharply separating models and programs, complicating their future convergence, and that it is perhaps time to move forward in the direction of new generations of unified modeling and programming languages.

Haxthausen and Peleska [5] (*On the Feasibility of a Unified Modelling and Programming Paradigm*) argue that we should not expect there to be a single "best" unified modeling, programming and verification paradigm in the future. They, on the contrary, argue that a multi-formalism approach is more realistic and useful. Amongst the reasons mentioned is that multiple stake holders will not be able to agree on a formalism. The multi-formalism approach requires to translate verification artifacts between different representations. It is illustrated by means of a case study from the railway domain, how this can be achieved, using concepts from the theory of institutions, formalized in category theory.

### 2.2 More Concrete

Elaasar and Badreddin [3] (*Modeling Meets Programming: A Comparative Study in Model Driven Engineering Action Languages*) compare two approaches, Alf and Umple, where modeling meets programming. They start from the remark that modeling and programming have often been considered two different domains. They point out, that this is true when modeling is primarily meant for human communication, but is not the case when modeling is meant for execution. In their paper they discuss two approaches that specifically address execution. In particular they consider the language Alf, that has evolved from the modeling community to make models executable, and Umple, that has evolved from the academic community to introduce abstractions of modeling into programming languages. The paper discusses critical differences, and ideas for future evolution of model oriented programming languages.

Lattmann, Kecskés, Meijer, Karsai, Völgyesi, and Lédeczi [8] (*Abstractions for Modeling Complex Systems*) present three abstraction methods for improving the scalability of the modeling process and the system models themselves. The abstractions, crosscuts, model libraries, and mixins, are part of the WebGME framework, and the paper describes how these abstractions are incorporated into this framework.

Leavens, Naumann, Rajan, and Aotani [9] (*Specifying and Verifying Advanced Control Features*) discuss the problem of verifying programs that are written with design patterns such as higher-order functions, advice, and context dependence. Such concepts allow for greater modularity and programming convenience, but tend to be harder to verify. They propose the use of Greybox specifications and techniques for verification of such programs.

### 2.3 Meta-Level Considerations

Rouquette [13] (*Simplifying OMG MOF-based Metamodeling*) discusses the complexity of UML, the meta-model MOF used to define it, and the XML Metadata Interchange (XMI) standard used for serializing UML models. He alternatively suggests to define the abstract syntax of a modeling language as a normalized relational schema, and to consider a particular model as tabular instance of that schema. He finally suggests leveraging recent advances in functional programming languages, such as Scala, in order to modernize the traditional practice of model-based programming with the Object Constraint Language (OCL) and the Query/View/Transformation (QVT) standards.

Prinz, Møller-Pedersen, and Joachim Fischer [12] (*Modelling and Testing of Real Systems*) elaborate on OMG-style modeling conventions, in particular by introducing the distinction between description and prescription in order to deal with partially realized systems. Whereas the former is intended for capturing already existing parts of a foreseen system, the latter specifies to be realized parts. This distinction is also used in their testing approach which reminds of hardware in the loop or back-to-back testing, where real and simulated parts are simultaneously used. The power of this approach depends on the executability level of the underlying (modeling) languages, which ranges from mere presentation to dual executability as required for fully exploiting the presented testing approach. In this sense, the corresponding 5-level hierarchy can be regarded as a specific top-level view for merging the modeling and programming landscapes.

Kugler [6] (*Unifying Modelling and Programming: A Systems Biology Perspective*) suggests to explore the topic of unifying modeling and programming in the context of computational systems biology, which is a field in its early stages, and therefore potentially more receptive to new ideas. Here, for example cells can be effectively described as biological programs. Software and system development share several of the main challenges that computational systems biology is facing : making models amendable to formal and scalable reasoning, using visual languages, combining different programming languages, rapid prototyping, and program synthesis.

### 2.4 Domain-Specific Approaches

Berry [1] (*Formally Unifying Modeling and Design for Embedded Systems - a Personal View*) shows the direct and formal connection between model-based design and programming in the synchronous languages framework, which is tailored for the embedded system domain. In this setting higher-level models can comfortably be designed, verified, and subsequently transformed to high quality system code. That this actually works as described has been witnessed in the past, and it is the foundation for the Kieler framework, where the design of hardware circuits is targeted. On the one hand, Berry's work impressively demonstrates the synergies between modeling and programming. On the other hand, the author explicitly admits that the success of his overall framework is domain-specific, and cannot easily be generalized.

Rybicki, Smyth, Motika, Schulz-Rosengarten, and Hanxleden [14] (*Interactive Model-Based Compilation Continued Incremental Hardware Synthesis for SCCharts*) present an extension of the Kieler development framework, which further strengthens its meta-modeling based approach for stepwise translating high level descriptions to hardware. Remarkable is its user-orientation: not only can the effect of the (M2M) transformations steps be controlled via sophisticated graphical visualization, but also subsequent simulation runs can be followed at each of the intermediate levels. Thus Kieler can be regarded as a framework where models are "morphed" to programs and even hardware by automatic transformation. At the practical side this illustrates the maturity of Eclipse's meta-modeling facilities, which, in the meantime, can effectively be used to integrate domain-specific languages into a development framework.

Larsen, Fitzgerald, Woodcock, Nilsson, Gamble, and Foster [7] (*Towards Semantically Integrated Models and Tools for Cyber-Physical Systems Design*) argue that the modeling of specifically embedded cyber-physical systems best can be done using multiple formalisms. A case study of a small unmanned aerial vehicle is used to demonstrate the need for multiple formalisms, namely a formalism for defining control, in this case VDM-RT, and a formalism for continuous behavior defined by differential equations, in this case 20-sim. The integration is founded in the semantic framework of Unifying Theories of Programming (UTP). Combined systems are suggested simulated via a co-simulation framework.

### 2.5 Tools and Frameworks View

Lethbridge, Abdelzad, Orabi, Orabi, and Adesina [10] (*Merging Modeling and Programming using Umple*) present Umple, a programming and modeling language that has been created by introducing modeling constructs in programming and vice-versa. Umple aims at maintaining model-code and text-diagram duality. Several examples and uses of Umple are provided with a broad discussion.

Elmqvist, Henningsson, and Otter [4] (*Systems Modeling and Programming in a Unified Environment based on Julia*) present a new methodology for modeling cyber physical systems using a Modelica-like extension of the powerful Julia programming language – a language extension they call Modia. A good discussion is provided by the authors regarding the needs of a Model Based Systems Engineering approach, along with a strong description of the features and implementation of the Modia language. Examples and illustrations are also provided in great detail.

Naujokat, Neubauer, Margaria, and Steffen [11] (*Meta-Level Reuse for Mastering Domain Specialization*) reflect on the distinction between modeling and programming in terms of WHAT and HOW, and emphasize the importance of perspectives: what is a model (a WHAT) for the one, may well be a program (a HOW) for the other. In fact, attempts to pinpoint technical criteria like executability or abstraction for clearly separating modeling from programming seem not to survive modern technical developments. Rather, the underlying conceptual cores continuously converge. What remains is the distinction of WHAT and HOW, separating true purpose from its realization, i.e. providing the possibility of formulating the primary intent without being forced to over-specify. The paper argues that no unified general-purpose language can adequately support this distinction in general, and propose a meta-level framework for mastering the wealth of required domain-specific languages.

### 2.6 Panel

The panel section started with a presentation by Broy, Havelund, and Kumar [2] (*Towards a Unified View of Modeling and Programming*), who present an argument for unifying modeling and programming in one formalism. They highlight relevant developments in the fields of formal methods, model-based engineering, and programming languages. They subsequently illustrate how modeling can be perceived as programing via examples in the Scala programming language. The paper concludes with a summary of issues considered important to reflect on in any attempt to unify modeling and programming. They specifically highlight the need to combine textual and visual languages, the need for allowing definition of domain-specific languages, and the need for analysis support.

## References

1. G. Berry. Formally unifying modeling and design for embedded systems – a personal view. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
2. M. Broy, K. Havelund, and R. Kumar. Towards a unified view of modeling and programming. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
3. M. Elaasar and O. Badreddin. Modeling meets programming: A comparative study in model driven engineering action languages. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
4. H. Elmqvist, T. Henningsson, and M. Otter. Systems modeling and programming in a unified environment based on Julia. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
5. A. E. Haxthausen and J. Peleska. On the feasibility of a unified modelling and programming paradigm. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
6. H. Kugler. Unifying modelling and programming: A systems biology perspective. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
7. P. G. Larsen, J. Fitzgerald, J. Woodcock, R. Nilsson, C. Gamble, and S. Foster. Towards semantically integrated models and tools for cyber-physical systems design. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
8. Z. Lattmann, T. Kecskés, P. Meijer, G. Karsai, P. Völgyesi, and Ákos Lédeczi. Abstractions for modeling complex systems. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
9. G. T. Leavens, D. Naumann, H. Rajan, and T. Aotani. Specifying and verifying advanced control features. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
10. T. C. Lethbridge, V. Abdelzad, M. H. Orabi, A. H. Orabi, and O. Adesina. Merging modeling and programming using Umple. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.

11. S. Naujokat, J. Neubauer, T. Margaria, and B. Steffen. Meta-level reuse for mastering domain specialization. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
12. A. Prinz, B. Møller-Pedersen, and J. Fischer. Modelling and testing of real systems. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
13. N. F. Rouquette. Simplifying OMG MOF-based metamodeling. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
14. F. Rybicki, S. Smyth, C. Motika, A. Schulz-Rosengarten, and R. von Hanxleden. Interactive model-based compilation continued – incremental hardware synthesis for SCCharts. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
15. E. Seidewitz. On a unified view of modeling and programming – position paper. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.
16. B. Selic. Programming $\subset$ modeling $\subset$ engineering. In T. Margaria and B. Steffen, editors, *7th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2016, Corfu, Greece, October 10-14*, LNCS. Springer, 2016. These proceedings.