Softw Syst Model (2005) 4: 1–3 / Digital Object Identifier (DOI) 10.1007/s10270-005-0078-1

Editorial

Domain specific modeling

Robert France, Bernhard Rumpe

Published online: 25 January 2005 –

Looking at other engineering disciplines, it is evident that modeling is a vital and important part of the development of complex artifacts. Good modeling techniques provide support for the separation of concerns principle, rigorous analysis of designs, and for structuring construction activities. Models can play a similar role in the development of software-based systems. Furthermore, the software development activity has a characteristic not present in physical engineering: deliverable products (software systems) can be generated from models (an observation made by Bran Selic, IBM/Rational at an MDA summer school). This characteristic can and should be exploited in our quest to make software development an engineering activity.

The current landscape of modeling languages is highly diverse. Graphical languages, such as Petri Nets and Statecharts have successfully been used for years. Standardized languages such as the SDL have had a good base of tool support but have seen their use in industry diminish over time. Languages can be specific to an application domain (e.g., SDL was developed to support modeling of telecommunication systems), to a development phase (e.g., SCR method uses a language designed specifically for modeling requirements of reactive systems), or they can be general-purpose (e.g., the UML). Recently, much attention is on the development of domain-specific languages (DSLs). Proponents of DSLs claim that their use can help bridge the gap between a domain expert's view of a software system and its implementtion. The domains covered by DSLs can range from highly individual application areas, such as "railroad planning applications" to broader domains, such as the "embedded system domain".

The UML was at first an attempt to unify various object-oriented modeling languages, and it seemed that its target applications were primarily business systems. The UML is now being used to model applications and concepts in a variety of domains, including embedded systems and business workflows. While this broadened the scope of the UML it has made it difficult to develop semantics that can be used to support its application in a number of domains. This has led to the realization that a single, consistent semantics that supports the use of the UML may not be possible, and to the view of the UML as a family of languages. There are currently a number of semantic variation points in the UML to support this notion. Developing a semantic framework for the UML that takes into consideration its numerous variation points is proving to be an extremely difficult task – and the important issues are not all technical!

It should not be surprising then that in specific domains, the use of existing "domain specific languages" persists. A DSL, let it be a programming or a modeling language, has several advantages. Among them:

- Domain specific constructs are better suited for communication with users in the domain. The users can better understand the models if they are presented in domain-specific terms. This leads to the claim that domain specific models are better suited for requirements engineering than the UML.
- DSLs have restricted semantic scope (i.e., the number of semantic variations they have to deal with is small compared with general-purpose languages), thus developing a semantic framework is less challenging.
- Restricting semantic scope can lead to better support for generating implementations from models. Given an appropriate component-framework that implements individual constructs of the DSL, the composition of these components can lead to powerful implementations.
- DSL's increase domain specific reuse of components, which can lead to improved the quality of systems and order-of-magnitude improvements in time-to-

market and developer productivity. Examples from the telecommunication indicate that a speedup factor of ten is possible.

So far DSL's have been developed in an ad-hoc manner, that is tooling was implemented without much reuse of existing technology. Reuse was limited in some cases to use of graphical framework for developing editors and the use of parser generators for textual languages. More recent work has focused on making the development of DSLs more systematic. Among the DSL development environments that are being created are Meta-Editors and graphical editor environments that allow developers to quickly create new DSLs and associated editors. In the MDA/MOF context, general data structures for the abstract syntax of models and techniques for their transformation are under development. Software companies such as Microsoft have identified this as an important area of development, and Microsoft and have released early versions of DSL development environments.

The advent of frameworks for developing DSLs will raise new challenges, some of which are identified below:

- If building or improving a DSL is a vital part of a project, the development process needs to incorporate these new activities. = It is not yet clear if a new breed of software development specialists (domain engineers?) will be needed to effectively carry out these new activities.
- Two questions that will have to be addressed can be stated as follows: Is there a need for company or industry wide standard DSLs? If yes, who will develop the standards?
- If the burden of defining a new DSL becomes considerably lower, we will probably see a large number of DSLs created and used. The situation will be similar to the one faced with the use of XML-Dialects: Many Dialects will appear and their interoperability or translation between them will be a major problem. This could give rise to a modeling Tower of Babel.
- To prevent the Tower of Babel, it might be necessary to build DSLs on a common syntactical and semantic base that reaches far beyond a common infrastructure for defining abstract syntax. The UML might provide such a base, but how can one derive a structural or behavioral DSL from a sublanguage of the UML?
- A language should have a semantics (in its original meaning of "meaning"). Will a DSL development framework support practical definition of DSL semantics of DSLs? How can this be done?

The editors look forward to an interesting discussion on the above and other challenges related to the development of DSLs.

The MoDELS conference series

As some readers might know, the Software and Systems Modeling Journal was born out of a stated need for a premier communication medium for topics related to modeling. In this area of research and practice, The ¡¡UML¿¿ conferences have played a significant role in pusing the state of the art in modeling and in evaluating modeling experience. Readers of the first set of SoSyM special issues will recall that they consisted of the best papers in these conferences.

When the SoSyM journal started, it was already clear that the UML will be an important but by not the only modeling language. For this reason, SoSyM's scope was defined to be inclusive of all other software-based system modeling languages.In the UML/modeling community this was also clear:

45% of the papers at the ≪UML≫ conferences were MDA-related papers, and many of those were not related to UML. Keynotes of the ≪UML'2004≫ addressed topics on software composition (Oscar Nierstrasz) and generative software development (by Krzysztof Czarnecki).

In 2005 the widening of the conference scope to all modeling related topics will occur. The UML-conference series will be renamed the

MoDELS-conference series,

where MoDELS is an acronym for "Model Driven Engineering Languages and Systems". The successful UML-series of conferences will continue and keep its focus on models of software and software-related systems. However, it will have tracks for papers on UML, Model Driven Architecture (MDA) and Domain Specific Languages (DSL). For more details about the conference see the following website: `http://www.modelsconference.org`.

Papers in this issue

In the expert voice "**UML – the Good, the Bad or the Ugly?**" Brian Henderson-Sellers compiled a set of position papers on the new UML 2.0 standard. The papers were written by experts who were directly involved in, or influenced the UML development process. Each contributor had the opportunity to read and respond to other contributions in the compilation.The contributors, namely, Steve Cook, Steve Mellor, Joaquin Miller and Bran Selic, highlight strengths and weaknesses of the UML 2.0, discuss improvements over previous UML versions, and gives their perspectives on the future of the UML.

In the regular paper "**Formal verification of software source code through semi-automatic modelling**" Cindy Eisner describes experience related to the verification of an implemented caching mechanism. The verification involved generating a model from the implementation and running a model checker on the model to verify desired properties . Experience reports can significantly contribute to the development of the field and we strongly encourage authors to submit high quality experience reports.

The regular paper "**The KeY tool? Integrating object oriented design and formal verification**"

by Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Richard Bubel, Martin Giese, Reiner Hähnle, Wolfram Menzel, Wojciech Mostowski, Andreas Roth, Steffen Schlager and Peter H. Schmitt describes a formal verification tool that uses the OCL as formal specification language. The tool supports verification of programs writtebn in a subset of the Java language.

In the regular paper "**An approach for reverse engineering of design patterns**", Ilka Philippow, Detlef Streitferdt, Matthias Riebisch and Sebastian Naumann describe a novel approach to detecting design patterns in existing software products. The approach uses minimal key structures and positive as well as negative search criteria for its search and is able to detect all GOF design patterns.

Jon Whittle, Richard Kwan and Jyoti Saboo describe their experiences in applying sequence diagrams as a primary model notation in their regular paper "**From scenarios to code: An air traffic control case study**". They define a mapping algorithm that produces statecharts from sequence diagrams, and they show how code can be generated from the models.

In the final regular paper "**A reference framework for process-oriented software development organizations**", the authors João M. Fernandes and Francisco J. Duarte describe techniques for modeling development processes. Instead of the usual application of models to the software under development or to the environment that uses the software, these models are used to structure the development process itself. The Rational Unified Process was one of the first prominent approaches to explicitly use models and the authors embed this process into their framework.

We hope you enjoy reading the articles in this issue,

Robert France, Bernhard Rumpe
Editors in Chief