Software & Systems Modeling (2018) 17:713–714
https://doi.org/10.1007/s10270-018-0685-2

EDITORIAL

UML customization versus domain-specific languages

Jeff Gray[1] · **Bernhard Rumpe**[2]

Published online: 9 June 2018

We recently collaborated again with 40 developers on a large automotive architecture project. In addition, we discussed some of the characteristics of the project with other modelers and consultants. Several observations emerged that we would like to share.

First, modeling is a complex skill that requires much practice and cannot simply be adopted by drawing diagrams. Even experienced programmers have challenges when developing models that are significantly more abstract than just a repetition of code structures and algorithms. When using explicit modeling languages for software in a mechanical engineering company, we observed that most of the modelers with domain knowledge are not experienced with even basic fundamentals of software engineering. The lack of foundational software engineering knowledge leads to many different problems that can be found in the models produced, such as:

- The models are either overly abstract or much too fine-grained.
- The models have an ill-balanced structure and therefore cannot be used for obtaining an overview of the available information easily.
- Many model elements are used incorrectly, i.e., the intended semantics does not correspond to the actual language semantics, which reduces the value of a model to individual interpretation.
- Sometimes, much of the relevant information is encoded as comments and very little structure and especially behavioral information is given in the model itself.
- Engineers with domain expertise often focus on creating models for their own specific situation and are usually not concerned with structuring their models into reusable, modular artifacts. This occurs far too often across many different industries. Even though some engineers miss opportunities to generalize their solutions to other situations or products (even within their same domain), the modeling languages need to also provide more explicit support for modularity to encourage best practices that incorporate concepts such as interfaces and encapsulation at the model level.

Second, many of the problems that we observed from this project are not new. The standard way of improving the process is to provide detailed guidelines at each level and phase in the modeling process (e.g., which model elements should be used, how they should be connected, how complete and accurate the model should be). Such guidelines have been helpful with new software engineers focused on coding practice. For example, object-oriented programming languages provide language support and structures for separating concerns. Guidelines have helped to make code readable (e.g., layout, names, structural nesting), design patterns document experiential reuse opportunities supporting best practices in code structure, encapsulation and clear object-oriented interfaces improve understandability and reuse.

To many researchers and practitioners, it is clear that similar guidelines are necessary when using a general-purpose modeling language, such as UML for software or SysML for embedded software systems that control mechanical devices. Both languages are designed for general-purpose use. The tool developers that adopt these languages strive to make their tools applicable for a broad set of domains and projects. As a consequence, the tool support is often generic and lacks methodical assistance, if provided at all. Thus, the situation is very similar to implementation needs with programming language usage: guidelines are beneficial and sometimes necessary to manage the inexperience of engineers who have domain expertise, but little foundational software engineering knowledge.

However, there is an unfortunate difference between programmers of general-purpose programming languages and modelers using general-purpose modeling languages (e.g., SysML). The latter usually know very much about the

✉ Bernhard Rumpe
bernhard.rumpe@sosym.org

Jeff Gray
jeff.gray@sosym.org

[1] University of Alabama, Tuscaloosa, AL, USA

[2] RWTH Aachen University, Aachen, Germany

domain, but lack knowledge and experience with the supporting modeling language. Domain experts often do not really internalize the guidelines and therefore have to look up (or ignore) the guidelines every time they start modeling. But more importantly, domain experts often have many other responsibilities and models are a smaller part of their daily routine. This restricted time is a detriment toward growing domain experts into modeling experts. This situation adds to the awkwardness of modeling the structure and behavior in a general-purpose modeling language, such as UML or SysML.

As a third observation and consequence of the first two observations, domain experts who create models in domains that are not related to information technology concerns need methodical assistance from their tools. Tools should constructively apply the methodical guidelines, thus preventing modelers from having to lookup the guidelines repetitively. For example, tools need to be tailored in such a way that only those modeling elements appropriate in a specific situation are presented as options for the domain experts constructing the model. Potentially new views need to be defined, with encapsulation mechanisms made more explicit in the tools, and assistance in tracing the connections across the abstractions represented in the models. Modeling tools need to offer multiple opportunities for customization, extension, and in particular, restriction. After such an extensive customization, the general-purpose language may emerge as a domain-specific language (DSL) based on UML or SysML.

Unfortunately, it is not always clear whether the underlying DSLs can be generalized to the domain or the company, or if they are highly specific to each project. However—and this is the fourth observation—an experienced "toolsmith" or "method developer" may be needed to assist the domain experts in customizing the tools. For large system-development companies, it might be useful to have a group of toolsmiths that concentrate primarily on this form of customization.

Finally, as a fifth and more general observation for the modeling community, it seems to be an interesting question whether it is better to develop a new DSL from scratch by adding modeling elements iteratively, or start with a general-purpose modeling language and restricting it until it fits the specific use.

We might see a yo–yo effect here: in the 1990s, many methods and modeling languages were popularized. Then, for a while, unification based on UML was very helpful. Then, DSLs that were developed from scratch began to emerge. The next trend may be a repository of UML/SysML-based DSLs that actually unify DSL and UML/SysML thinking.

Thanks for the input from Stefan Kriebel (BMW) and Nikolaus Regnat (Siemens).

Content of this Issue

This volume contains the MODELS 2015 Special Issue with Jordi Cabot and Alexander Egyed as guest editors. Two regular papers are also included:

- "Case-based exploration of bidirectional transformations in QVT Relations" by Bernhard Westfechtel.
- "Visual modeling of RESTful conversations with RESTalk" by Ana Ivanchikj, Cesare Pautasso, and Silvia Schreier.