Software and Systems Modeling (2021) 20:591–593
https://doi.org/10.1007/s10270-021-00887-w

EDITORIAL


# Modeling in the large: model libraries

**Jeff Gray**[1] · **Bernhard Rumpe**[2]

Published online: 14 May 2021


*Modeling in the Large* is a key concept that is vital toward addressing the growing complexity and organizational requirements that are faced by developers when applying modeling techniques to real-world problems. For the most simple products, it is usually not necessary to define and follow a complicated formal development process. Modeling is particularly beneficial if the product is complex, comes in many different variants, or if the product is for a highly regulated domain (e.g., safety and security regulations). In these cases, one model cannot describe the whole product, but many models are needed to define multiple interacting concerns, often requiring several different languages to describe various aspects and viewpoints of the products or parts of the system under development.

A modeling approach for software and systems engineering naturally needs to cope with many models that are related to each other. Research at the intersection of programming languages and software engineering has articulated the benefits of a clear and sound *modularity concept* to contain the complexity of various decompositions that are essential for programming of large systems. A key insight from these findings is that programming modules (e.g., modules, classes, packages) are more reusable if they are self-contained with clear boundaries (i.e., interfaces). This idea was even promoted in the earliest discussions of software reuse when Doug McIlroy introduced the concept of software components and reuse in the late 1960s. The concept of modular reasoning also facilitates a discussion about the correctness of a module without having to understand every detail specified in other parts of the system. Reusability and effectiveness of interface design have allowed us to design software more quickly and with higher quality. In fact, at the past annual MODELS and AOSD conferences, several SoSyM editors organized over 15 instances of the "Workshop on Aspect-Oriented Modeling" to address these core issues, even though aspects have not proven to be as modular for all modeling approaches.

Thus, a good modeling language should also offer techniques to define models in a modular and reusable way. If the reuse of models can be standardized in specific domains, then model-based software and systems engineering will progress with even more breakthroughs. The advent of *model libraries* will allow us to come up with good, model-based designs in early phases of development.

It cannot be stated more emphatically:

> Modeling languages need to encourage modularity of their artifacts. Only then can reusable models be organized in model libraries.

However, this goal is difficult to achieve. It took a number of years to understand how to design programming languages with a good modularization concept. With the heterogeneity of models and modeling languages, this concept of modularization is even more complicated.

Technically, there are advantages in connecting models in the same way as programs: There is usually an explicit "import" or "include" statement that references other dependencies. When expressing such dependencies, the symbols and resources defined in the imported models are available in the importing model. Java has a precise understanding of similar concepts; namely, classes, interfaces, methods, and attributes are exported symbols. As an example comparison, state machines can export and reuse states and triggers as symbols. However, many aspects in the interplay between state machines and other types of models are unclear in current modeling languages, such as UML and SysML. For example, how are the symbols in one model represented and imported as states in a state machine? Or, are the states defined in the state machine and exported to other models (and programmed pieces of code)? Or a third option, are state names defined locally and not exported at

✉ 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

all, as a form of information hiding (i.e., hidden, like private resources in a Java class)?

The situation becomes even more problematic when considering the fact that different modeling languages have different kinds of symbols, and the interfaces between modeling languages (and even programming languages) may not have a precise mapping definition of each symbol between the languages. For example, if a state machine exports its state names, how can they be used and integrated in a Java program that needs to access information that is within the model? Java does not know the concept of "state"—thus, mapping a specific state could have multiple options, such as mapping to a Boolean variable of the same name, or an enumeration value, or even a class name (like the State design pattern suggests).

These questions are not easy to resolve, and for various different kinds of projects, the solution approach could be different (e.g., the state machine example suggests that various options are possible). An integration of modeling languages under a concept like *development viewpoints* may allow a viewpoint or even project-specific forms of integration.

Future Modeling in the Large research must clarify these issues. We hope that in the future, the modeling community can move from a dominant focus on single models and single language research to more improved approaches for model and language integration.

Currently, several SysML tool vendors and their standardization efforts are considering another approach. The UML and the SysML standards do not have many concepts that encourage the development of independently reusable and storable models. Instead, many of the commercial tools offer a "one size fits all approach" that requires a modeling artifact to consist of many connected and related models that are stored together and usually cannot be reused independently without some difficulty. We assert that more organizations are defining their own domain-specific modeling techniques (i.e., their own versions of how to organize model elements) in a database-oriented way. This leads to a "model warehouse," where everything can be related to everything else, and additional effort is necessary to define appropriate modularity boundaries. Of course, another problem with this kind of model storage strategy is versioning of the model instances, problems with building variants, and ensuring undisturbed development in a local subspace (which developers usually favor before they commit a consolidated update in a developer-friendly version control system). In contrast, we note that the majority of attempts to store program source code in a database has been abandoned, with file-based management of programs preferred as the standard for most modern projects.

It will be interesting to see which forms of model modularity will be more successful and more beneficial for developers in the long term.

## Content of this issue

1. **Expert Voice**
   - "Specifying dynamic software system architectures" by Bran Selic
2. **Theme Section on Multi-paradigm Modeling for Cyber-Physical Systems**
   Guest Editors: Eugene Syriani and Manuel Wimmer.
3. **Regular Papers**
   - "Implementing QVT-R via semantic interpretation in UML-RSDS" by Kevin Lano and Shekoufeh Kolahdouz-Rahimi
   - "Wodel-Test: A model-based framework for language-independent mutation testing" by Pablo Gómez-Abajo, Esther Guerra, Juan de Lara, and Mercedes Merayo
   - "Live modeling in the context of state machine models and code generation" by Mojtaba Bagherzadeh, Karim Jahed, Benoit Combemale, and Juergen Dingel
   - "Graphical composite modeling and simulation for multi-aircraft collision avoidance" by Feng Zhu and Jun Tang
   - "Pragmatic reuse for DSML development—Composing a DSL for hybrid CPS modeling" by Stefan Klikovits and Didier Buchs
   - "CEViNEdit: improving the process of creating cognitively effective graphical editors with GMF" by David Granada, Juan Manuel Vara, Mercedes Merayo, and Esperanza Marcos
   - "A systematic literature review of cross-domain model consistency checking by model management tools" by Weslley Torres, Mark van den Brand, and Alexander Serebrenik.

**Funding** Open Access funding enabled and organized by Projekt DEAL.