Software and Systems Modeling (2021) 20:287–289
https://doi.org/10.1007/s10270-021-00881-2

EDITORIAL

Reflections on the standardization of SysML 2

Jeff Gray[1] · Bernhard Rumpe[2]

Published online: 25 March 2021

In 2018, we wrote an editorial entitled "Agile Model-Based System Development" [GR18c], in which we made several observations:

1. "In fact, modeling is at the heart of almost any engineering discipline. Thus, it is not surprising that our engineering colleagues have developed a detailed portfolio of modeling techniques to describe their systems in various perspectives, viewpoints, and abstractions."
2. "However, the state-of-the-art in systems modeling has several challenges, where each modeling aspect and view is often assisted by an individual modeling and analysis tool. Data exchange between the tools is complicated, even though mostly automated, but suffers from concerns about robustness, completeness of the mappings between the models, as well as regular version upgrades of tools. A second problem is that these mappings between models are not easy to standardize, because different projects use the same modeling languages in different forms (semantics), which enforces configurable mappings or individually developed translations per project."
3. "Traditional engineers use system models to design and understand the product through analysis and automation of engineering tasks. Therefore, models should be more than just documentation artifacts. Each model that captures a perspective of the project should either become a part of the product construction or should be used for automated validation and test quality management."

We also discussed that traditional systems development has not yet reached the maturity of agile software development for a number of reasons, including the far too long, complex, and dependent tool chain.

Currently, the development of the updated standard SysML 2.0 is a larger effort, where the stakeholders are trying to accommodate a number of these problems:

1. The language itself has received a renovation in its syntactical shape, with the expectation that a consolidated set of syntactic constructs leads to a better methodical use of the various modeling sub-languages.
2. SysML 1 was defined as a profile based on the UML, which on first consideration was a good idea, but in reality and practice led to a number of complications. Therefore, SysML 2 will be a standalone language independent from UML. SysMLs relation to other modeling languages, like UML, CAD, PLM, etc. is therefore not part of the SysML standardization. (But because SysML and UML are at least partially overlapping there is potential to create inconsistencies and confusion).
3. SysML 2 will cover an even wider range of views that can be described, even embedding geometrical models, such as CAD and physical simulation models of various forms.

It will be interesting to see the results of the new SysML standardization effort.

On the one hand, the community seems to pretty much agree that a standardization that addresses many systems modeling needs is necessary; to quote Wittgenstein, "The limits of my language mean the limits of my world." On the other hand, we know from UML, that trying to cover every aspect and viewpoint leads to a complex language, which in turn leads to reluctance to learn and adopt the language. Possible solutions to this challenge will not be to reduce the complexity of the language, but to modularize the language in such ways that it can be used and learned in pieces, without having to deal with the rest of the language in total. Both natural language, as well as typical electrical and mechanical engineering norms, are often built that way. Such modularization should not only include the syntax, but

✉ 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

also the internal tool representations and semantics. Therefore, we suggest:

Make SysML modular in all relevant dimensions.

It would be very helpful if SysML is not presented as a big blob, but rather as a set of "packages" or "layers" of individually usable sub-languages. We suggest:

SysML should become a library of independent, individually usable languages, and with a clear understanding of how they relate.

Standards need to be produced collaboratively by individuals. Frequently, these collaborators seek compromise to arrive at the best possible effort and results, but they are sometimes biased by their own personal background and by their own company's interests that are supporting their participation in the standardization committee. In the case of SysML (and formerly UML), stakeholders are often tool vendors who have their own specific goals and objectives, and want to protect their own intellectual property (e.g., algorithms, tools, and processes) against becoming outdated because of different interpretations of the language in the standard. This may be even more prevalent for large user groups that have adopted specific tools and the embedded version of the SysML language that is manifest in such tools in certain forms. This is a typical phenomenon, where the tool developers originally create clever ideas, which might really be an optimal realization based on their target domain and their own background. But once these algorithms are implemented, they become a legacy asset and contribute toward the complexity of a standardization effort. As a result, compromises among the committee stakeholders often leave the semantics of the language open to interpretation in some subtle nuances, allowing several different tool-realizations to be valid. This has advantages (e.g., domain-specific specializations are possible), but also creates disadvantages (e.g., the less precise a language is defined, the more difficult for users to understand it). Therefore, we also suggest:

Make SysML a language with precise syntax and semantics.

And to overcome the tool vendor and other stakeholder bias:

Ask researchers to identify optimal language syntax and semantics design.

This recommendation, however, raises another important issue, which can be observed in typical standardization processes. If a goal is to make a language precise, then a "meta-language" is needed that supports the description of both syntax and semantics. For the syntax, we have mainly two approaches, namely grammars for the textual models and metamodels (often expressed as class diagrams) for visual models. The desire for a precise semantics is not surprising, because the majority of stakeholders for a standardization process must be able to understand the standard. The semantics of a modeling language, however, cannot be defined easily using a class diagram. Many research papers, including some from the authors of this editorial, have been investigating approaches that produce a precise and understandable semantics. Please note that we use the term semantics as "meaning of a model" as opposed to "behavior of a program", because each modeling language (including class diagrams) has semantics, but some languages are focused on structure instead of behavior. Because class diagrams are too restricted in their expressivity, we suggest:

Ask researchers to define a formally precise semantics.

We also recommend that tool developers use the results and findings of this formally precise semantics to verify their context condition checkers, tool analyses and code syntheses against this semantics. This will result in a significant improvement on quality.

Because only certain aspects of the language semantics can be described using a class diagram on the "meta-level", another possibility is to specify the semantics of a model by describing the effects during its execution in the form of interpretation. This is a kind of operational semantics based on an abstract machine. This type of semantics definition is suitable for generation of code for prototypes, simulation and also for product code. But it also reduces a modeling language to a mere programming language, because developers then will no longer model properties in a compact form, but will "program" their models to become efficiently executable. A goal of SysML 2 is to cover all aspects of development and, therefore, requires it also to be a real modeling and specification language. This means that SysML 2 must be capable of allowing underspecification. If decisions are to be postponed, requirements will describe a bandwidth or range of possible realizations. Thus, variants of the system shall be possible and need to be specified, and uncertainty in development and during system operations must be captured. Therefore, we suggest: While it is ok to provide an executable semantics for a specific subset of the SysML, please also

Develop a true specification language that allows to capture underspecification in various forms.

Last year in [GR20a], we also argued that analysis is very important. In addition to traditional static analysis that considers the known context conditions, as well as the analysis of a simulated execution, there are additional possibilities for performing advanced analysis of models. In particular, symbolic analysis is of deep interest for analyzing underspecified models because such analysis allows the capture of all possible behaviors, through mimicking all (or at least the major) execution paths in parallel. Thus, we finally suggest:

> Besides a standardization of the language, it would be very beneficial to also standardize the kinds of analyses desired for SysML.

Let us closely watch how the SysML standardization will proceed. For scientists, the typical way to evaluate a standardization effort is to (1) run experiments with the expected or then defined standard, (2) develop and argue for improved language variants, (3) develop smart synthesis and analysis algorithms, (4) design well-calibrated "model smell" techniques, and (5) publish all findings in an appropriate conference or journal. Both the MODELS conference and the SoSyM journal were originally founded to accompany the UML 1 standardization effort. They will of course also serve to help the SysML standards become the best possible definition that will serve the needs of multiple stakeholders.

We thank Tim Weilkiens for a critical assessment of an earlier version of this editorial.

Cited editorials (available at https://www.sosym.org/editorials):

[GR18c] J. Gray and B. Rumpe: Agile model-based system development. In: *Journal of Software and Systems Modeling (SoSyM)*. Springer Berlin/Heidelberg. Volume 17(4), Pages 1053–1054, 2018.

[GR20a] J. Gray and B. Rumpe: Compositional model analysis. In: *Journal of Software and Systems Modeling (SoSyM)*. Springer Berlin/Heidelberg. Volume 19(2), Pages 261–262, 2020.

Contents of this Issue

1. **SEFM 2019 Special Section**
   Guest Editors: Gwen Salaün and Peter Csaba Ölveczky
2. **Regular Papers**

   - "An improved approach on the model checking for an agent-based simulation system" by Yinling Liu, Tao Wang, Haiqing Zhang, and Vincent Cheutet
   - "Modeling cultures of the embedded software industry: feedback from the field" by Deniz Akdur, Bilge Say, and Onur Demirörs
   - **SoSyM First Paper at MODELS 2020**: "Claimed advantages and disadvantages of (dedicated) model transformation languages: a systematic literature review" by Stefan Götz, Matthias Tichy, and Raffaela Groner
   - "Modeling and simulation of the IEEE 802.11e wireless protocol with hidden nodes using Colored Petri Nets" by Estefania Coronado, Valentín Valero, Luis Orozco Bargosa, Maria Emilia Cambronero, and Fernando L. Pelayo
   - "Consistent change propagation within models" by Roland Kretschmer, Djamel Eddine Khelladi, Roberto Lopez-Herrejon, and Alexander Egyed
   - **SoSyM First Paper at MODELS 2020**: "Specification and automated verification of atomic concurrent real-time transactions" by Simin Cai, Barbara Gallina, Dag Nyström, and Cristina Seceleanu.

Funding Open Access funding enabled and organized by Projekt DEAL.