Software and Systems Modeling (2025) 24:1621–1622
https://doi.org/10.1007/s10270-025-01339-5

EDITORIAL

Pragmatic specification of software behavior, configuration, and orchestration: the precision and usability of domain-specific modeling

Marsha Chechik[1] · Benoit Combemale[2] · Jeff Gray[3] · Bernhard Rumpe[4]

Published online: 7 November 2025

In recent discussions, a recurring theme has been the evolving role of specification languages in industrial practice. While formal methods and modeling languages have long aspired to provide unified frameworks for reasoning about software systems, contemporary usage patterns—particularly in large-scale industrial settings such as big tech companies—paint a different picture.

In several big tech companies, and potentially in other software-intensive businesses, specification exists almost entirely in the service of **verification**. If a specification does not directly enable the formal verification of an artifact of practical importance—such as code, APIs, configurations, protocols, or policies—it is deemed irrelevant. This perspective frames specification not as an abstraction layer or design blueprint, but as a tightly coupled tool for reasoning about concrete, existing system artifacts.

The implications are far-reaching. First, languages are chosen *not* for their generality or unification power, but rather for their precision and fitness for a specific verification task. They specify specific artifacts, e.g., configuration or orchestration of specific tasks. Task plans are then integrated using sophisticated and consistency-verifying tools. Often, this means designing small domain-specific languages (DSLs) that restrict certain forms of expressiveness, but in turn allow developers to employ SMT solvers or other reasoning tools as core reasoning engines. Therefore, the DSL merely serves as syntactic sugar. This tightly scoped and tool-driven approach diverges sharply from traditional ambitions of modeling languages like the UML.

In contrast, the UML was envisioned as a unifying paradigm—bringing together a variety of notations (e.g., class diagrams, sequence diagrams, state machines) under a shared semantic umbrella, which allows for modeling structure, behavior, interactions, and more under a common framework. This unification was one of UML's significant achievements: enabling (ideally, semantically consistent) co-expression of different views of a system. Yet, in practice, UML's lack of high-quality verification tools and the inherent complexity of supporting semantic variations across its sublanguages have limited its practical uptake in domains where correctness is paramount. The lack of high-quality verification tools is due to theoretical barriers (such as undecidability) and practical barriers (such as language complexity or usage variability), and it has not been seriously tackled by the community so far.

This raises a sobering question: **Did the pursuit of universal modeling dilute the focus on semantic precision and tool support?** In many safety–critical domains (e.g., avionics, automotive), UML has been eclipsed by specialized tools or executable, programming language-like tools (e.g., SCADE/Lustre, Simulink, or Spectra)—each optimized for narrow slices of the system development process, and each equipped with robust verification pipelines.

Still, we must acknowledge UML's enduring legacy: it showed that different modeling paradigms can, in principle, coexist under a shared semantic structure. This vision remains crucial—even if not fully realized—because the current "zoo of languages" in practical use leads to a new kind of fragmentation. As different teams or domains adopt their own fit-for-purpose languages, the question of **cross-language reasoning and translation** becomes both more important and more difficult. Obviously, a manual translation is far

✉ Bernhard Rumpe
bernhard.rumpe@sosym.org

Marsha Chechik
marsha.chechik@sosym.org

Benoit Combemale
benoit.combemale@sosym.org

Jeff Gray
jeff.gray@sosym.org

[1] University of Toronto, Toronto, ON, Canada

[2] University of Rennes, Rennes, France

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

[4] RWTH Aachen University, Aachen, Germany

too error-prone and, in evolutionary repetitions, also costly and tedious. The UML attempted to develop an integrated, translation-free approach, but so far has not been successful.

So what can we take from UML's journey? Perhaps the lesson is not that unification was a mistake, but that unification without *tool support for verification* lacked staying power. The human effort required to define semantic mappings between languages remains immense. No solver will do it for us. But the need for such mappings (or the return of an integrated approach) is growing—especially if we want specifications written in one language to be reused, verified, or composed with others.

Moving forward, we see the current twofold challenge for the modeling community:

1. To embrace the **diversity of domain-specific specifications** and build bridges—semantically and tool-wise—between them.
2. To ensure that **specifications remain actionable**, meaning verifiable, analyzable, and directly connected to the artifacts that practitioners care about.

This is not a retreat from the ambitions of modeling, but rather a solid next step in its evolution. The work done by the UML community in defining cross-language semantics is an integrating foundation we should build on and adapt—bringing those lessons into new contexts where reasoning and other verification tools are now taking center stage. This is especially relevant as new and "automated" junior developers, such as LLMs, will give us additional assistance in creating domain- and purpose-specific models.

As we look ahead, the future of specification may lie not in seeking a single universal language, but in enabling many languages to speak to each other—with operationally shared semantics.

Funding Open Access funding enabled and organized by Projekt DEAL.