Towards a Configurable Verification and Validation Framework for Critical Cyber-Physical Systems

Ármin Zavada
IncQuery Labs cPlc.
Budapest University of Technology and Economics
Budapest, Hungary
armin.zavada@incquerylabs.com
zavadaarmin@edu.bme.hu

Géza Kulcsár
IncQuery Labs cPlc.
Budapest, Hungary
geza.kulcsar@incquerylabs.com

Vince Molnár
Budapest University of Technology and Economics
Budapest, Hungary
molnar.vince@vik.bme.hu

Ákos Horváth
IncQuery Labs cPlc.
Budapest, Hungary
akos.horvath@incquerylabs.com

***Abstract*—Cyber-physical systems (CPS) are increasingly deployed in critical domains where safety and reliability are paramount. Model-Based Systems Engineering (MBSE) has emerged as a crucial methodology for managing the complexity of critical systems, leveraging models to enhance precision, understandability, and reusability. Verification and validation (V&V) techniques are central to MBSE, ensuring structural correctness and behavioral adherence to specifications. However, the proliferation of domain-specific languages (DSLs) with unique semantics often renders existing V&V solutions non-reusable and difficult to integrate into existing workflows. In this paper, we propose a configurable V&V framework that integrates static and dynamic analysis by leveraging semantic libraries to encapsulate language structure, constraints, and semantics. By relying upon semantic libraries rather than hardcoded transformation chains, supporting new high-level languages becomes a modeling task rather than a transformation task, allowing the refinement and reuse of existing language semantics. To validate the feasibility of the proposed framework, we demonstrate the approach using an open-source prototype and the OpenMBEE Space Mission model.**

***Index Terms*—cyber-physical systems, critical systems, mbse, model validation, model verification, domain-specific languages**

I. Introduction

Cyber-physical systems (CPS) are frequently employed in critical domains where safety and reliability are top priority. Critical systems integrate computational and physical components, creating complex environments where malfunctions can lead to severe consequences. Model-based Systems Engineering (MBSE) has emerged as a key methodology in these domains, leveraging models to enhance precision, understandability, and reusability [14]. One substantial benefit of using models is the opportunity for applying automatic reasoning techniques on the knowledge base describing the system, such as model validation and formal verification [9]. The expected benefit is a better understanding and, often, early problem detection throughout the design process, starting from the requirement engineering phase and extending to the implementation and operation phases (e.g., diagnostics [31]).

Model validation – often performed through static analysis – ensures that models adhere to certain well-formedness rules. This involves identifying structural and semantic inconsistencies using techniques such as graph-pattern matching [1]. Validation rules are typically derived from various sources, including language specifications (a syntactically valid model can be semantically invalid), design methodologies (such as the SYSMOD [33] rules for SysML models), or custom requirements tailored to specific projects. Given the diversity of validation needs, tools must allow flexible specification and customization of these rules to fit different modeling workflows effectively.

While validation focuses on structural correctness, verification ensures the system behaves as intended under specified conditions. Verification involves dynamic analysis techniques like model checking to evaluate the system's adherence to its specifications. However, a recurring challenge in formal verification is describing system designs using formal languages with sufficient precision. While there are existing tools that help in formalizing high-level languages [6], [18], [20], [32], they often require low-level formal knowledge, which is typically outside the expertise of systems engineers [19], or fail to integrate seamlessly into existing workflows. While these approaches definitely helped in increasing the adoption of formal methods in MBSE, there are many open questions and repeating challenges that prevent its widespread application [13], [17], [29].

Implementing verification and validation (V&V) tools is further complicated by the proliferation of domain-specific languages (DSLs), each with unique syntax and semantics. Even minor differences between DSLs often render existing V&V solutions non-reusable. Language workbench solutions commonly support model validation techniques but are usually limited in configurability during runtime. Similarly, existing model verification frameworks may lower implementation costs but often lack the flexibility required to support new or

Project no. 2024-2.1.2-EKÖP-KDP-2024-00005 has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation fund, financed under the EKÖP_KDP-24-1-BME-21 funding scheme.
This research has been partially funded by the Chips Joint Undertaking under grant agreement No 101111977, and by the National Development and Innovation Fund of Hungary under project no. 2022-1.2.8-KDT-2023-00003, within the frame of the Arrowhead fPVN project.

evolving DSLs. Maintaining consistency between constraints, semantics, and toolchains further increases the complexity, particularly when integrating these tools into existing workflows.

To address these challenges, we propose a configurable V&V framework that integrates static and dynamic analysis into a unified solution. The framework is designed to support the growing diversity of modeling languages – including domain-specific and newly developed languages – by reducing the complexity of V&V tool development.

Our approach employs two closely integrated validation and verification tools that process inputs specified in a dedicated modeling language. This design lets language designers formally specify language constraints, semantics, and composition rules within a reusable model. The framework leverages semantic libraries, first introduced in our previous work [34], to assign formal semantics to high-level languages by modeling their structure, behavior, and well-formedness rules. Consequently, the system automatically executes the required verification and validation tasks and maps the results back onto the original models. Supporting additional languages requires only the definition of their semantics (possibly reusing existing libraries) and the implementation of an appropriate frontend mapping the high-level language into the semantic representation.

To validate our approach, we present a proof-of-concept implementation built on open-source tools such as Semantifyr, Viatra, and Theta. We demonstrate the use of the framework by analysing parts of the OpenMBEE Space Mission SysML model.

II. Related Work

Managing the quality of models developed using MBSE is important, as high-quality models usually indicate high-quality resulting systems [8]. Model quality can be assessed using various quality metrics tailored towards specific aspects of the system [10]. One widely used technique is following existing system modeling methodologies, such as OOSEM [8] or SYSMOD [33]. Formal methods can also improve model quality by proving certain properties of the model. In a comprehensive review of domain-specific modeling methods (DSMM) engineering approaches, Ma et al. [19] detail validation and verification (V&V) techniques applicable at every phase of DSMM engineering. Their analysis outlines a clear roadmap for advancing V&V within this domain and advocates for the increased adoption of formal methods, which includes increasing the presence of formal methods in systems engineering curriculums and simplifying their use.

SysML [22] is a general-purpose systems engineering language built on top of the UML [23] language. As the finalization of SysML v2 [27] draws closer, we can expect to benefit from its many improvements. One substantial benefit of using SysML v2 will be its formal foundations through the use of KerML [26], which in contrast to UML, has well-defined, formal axiomatic semantics. In [21], Molnár et al. demonstrate the current status of the language and its semantics, and introduce some initial formal analysis tools that already can analyze SysMLv2 models. However, since the semantics of KerML is declarative, providing simulator and model-checker tools will be a challenge [36].

Bridging the gap between high-level engineering languages and low-level formal languages remains a longstanding challenge in formal methods. The so-called abstraction gap is typically addressed via model transformations or transpilations [6], [18], [20], [32], which require not only a syntactic mapping but also a semantic translation from a declarative specification to an operational representation. Preserving semantic equivalence – or at least ensuring conformance – between the two representations is crucial for maintaining the credibility of the analysis process. This task is particularly challenging when the semantics of the high-level language is under-specified. For example, Elekes et al. in [7] analyzed the specification of the PSSM standard [24] and reported various discrepancies between the documented semantics, its test set, and reference implementations.

In response to the inherent N×M transpilation challenge, several approaches have introduced intermediate languages (e.g., [2], [5], [15], [20]) to reduce the complexity to an N+M mappings. However, these intermediate representations frequently require modifications to accommodate the specific features of individual high-level languages, leading to increased maintenance overhead and limiting their scalability. The proliferation of DSLs also complicates matters, as the existing transpilations can rarely be reused between DSLs due to slight semantic differences requiring major changes in the intermediate representation. There are existing solutions that provide semantics to custom DSLs, such as the K Framework [4] and Spoofax [16]. Such tools usually work by specifying some compositional or pattern-based semantic rewrite rules that specify the execution semantics of the given language. However, they require the specification of the semantics during compile time and do not support their runtime configuration.

III. Dynamic Verification Toolkit

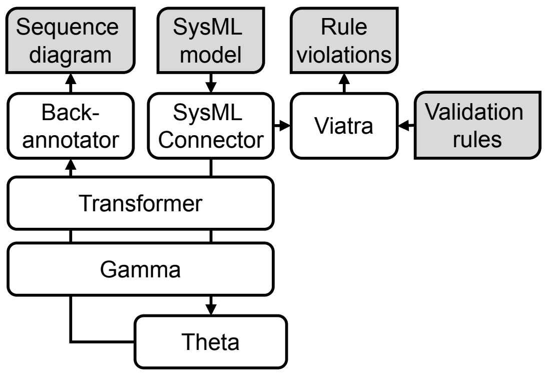

Fig. 1. High-level overview of the DVT transformation chain [15].

Dynamic Verification Toolkit (DVT), formerly known as Model Checking as a Service [15], is an integrated V&V

solution for a pragmatic subset of the Systems Modeling Language (SysML [22]). Figure 1 depicts the high-level overview of the DVT transformation chain. The tool builds upon the Gamma Statechart Composition Framework [20], which provides a semi-high-level formal language for the specification of statecharts and their various compositions and provides integration for numerous model checker backends, such as Theta [30]. Using the language it is possible to replicate the semantics of high-level languages. To verify parts of the SysML model, DVT transforms the input model into a semantically equivalent (given some semantic assumptions) Gamma model and delegates the verification problem to Gamma.

To increase the efficiency of the verification, DVT has certain pragmatic, semantic assumptions about the model – such as no fork and joins states in state machines – that reduce the state space of the model checking problem and simplify the transformation as well. Using the Viatra Query Language [1], validation rules ensure that these assumptions are respected before transforming the model into the Gamma intermediate language and verifying the dynamic aspects of the system.

The authors demonstrated the useability of DVT on various industrial SysML models, proving the applicability of the approach. While DVT demonstrates that such V&V solutions can be created for high-level languages, recreating similar tools for other languages – such as SysML v2 [27], or other DSLs – would possibly entail replicating the whole transformation chain.

Since Gamma uses semi-high-level languages, certain semantic assumptions must hold when using it, such as transition atomicity, the massage passing behavior between components, or the semantics of statecharts. In the case when a high-level language does not meet these assumptions – e.g., it uses activity models rather than statecharts or allows parallel transitions to be executed concurrently – additional non-trivial transformations are needed, or the Gamma language must be modified accordingly [36]. Similar problems arise with other toolchains using semi-high-level intermediate languages, such as [2], [5].

IV. Semantic Libraries

Semantic libraries [34] address this limitation by modeling the high-level language's semantics as a separate artifact rather than encoding it in the transformer implementation itself. This approach enables customizable verification tools with pluggable semantics, eliminating the need to recompile or repackage the tool for new configurations.

A. DVT as Semantic Libraries

Semantic libraries facilitate model derivations that reduce high-level models to suitable abstraction levels, directly mapping to formal languages with operational semantics. This approach eliminates the need for extensive manual transformations and focuses instead on leveraging model reasoners to handle semantic reductions guided by the rules encoded within the semantic libraries. Semantic libraries simplify the transformation process between languages to a syntactical mapping step and a model unfolding step; each syntactic language element has a corresponding type in the semantic library, unfolding which specifies the exact semantics of the given element. This integration allows for flexibility in defining and customizing low-level semantics tailored to specific use cases, semantic variants, or DSLs.

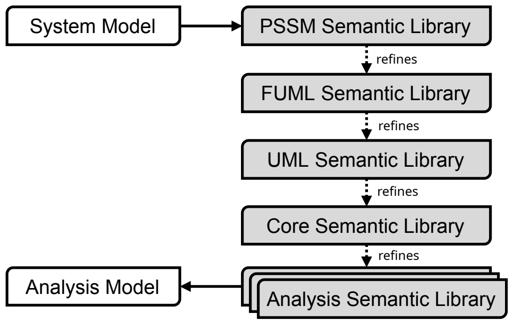

Fig. 2. An overview of a possible semantic library architecture of DVT.

Figure 2 illustrates a potential semantic library architecture for DVT. The Core library defines shared constructs (e.g., numbers, relations, expressions, successions), refined by the original UML [23] library according to the specification. UML has been refined by the fUML [25] and PSSM [24] specifications, further refining the UML semantic library, providing a more explicit semantics for the Activity and State Machine diagrams. Since the UML, fUML, and PSSM specifications do not provide a formal description of the language, the final PSSM Semantic Library has more refined, exact, formal semantics for the SysML language.

B. Semantic Constraints

To enforce semantic assumptions about the system model, we extend semantic libraries with semantic constraints embedded as first-class citizens into the library. These constraints are defined in terms of the library's declared elements, thereby reusing the same language structure. This integration promotes the reuse of these constructs and supports a design-by-contract methodology in the definition of semantic libraries. Each library explicitly specifies the types of models it supports, the constraints that must be satisfied, and how models can be interpreted in the context of the refined semantic library – which, in turn, asserts its own constraints and semantic assumptions.

As discussed in Chapter 4 of [34], refinements of a semantic library are required to produce execution traces that are stuttering equivalent. In this context, semantic constraints ensure stuttering equivalence by rejecting models that violate the underlying semantic assumptions of the library. For example, if a semantic library presupposes the absence of parallel regions or do-activities in the system model, it may simplify its operational semantics by omitting the portions associated with interleaving trace calculations. Under these conditions, the simplified operational model remains stuttering equivalent to the original, provided that the assumption is upheld.

This approach facilitates the creation of families of semantic libraries for a given language, each tailored to different levels of precision and supporting specific subsets of the language. Consequently, it enables the analysis of particular system aspects with an appropriately precise semantic library, hopefully reducing the verification process's overall complexity by reducing the state space of the problem.

V. Towards a Configurable V&V Framework

A high-level overview of our proposed V&V framework is illustrated in Figure 3. Only the components colored in blue need to be implemented for specific DSLs.

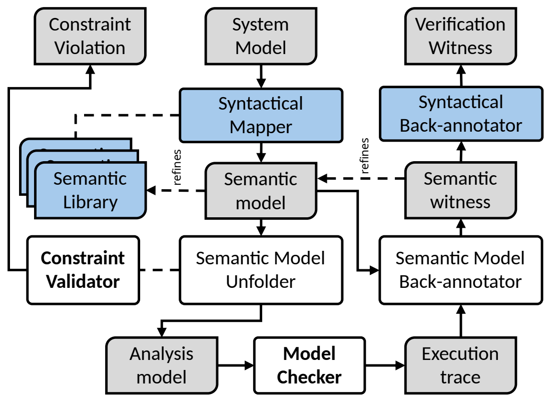

Fig. 3. An overview of the V&V Framework architecture.

A. System Model Verification

The framework leverages a layered architecture to achieve flexibility and modularity, ensuring that DSL-specific implementations require minimal effort. At its core, the system model is first mapped to a semantic model using a *Syntactical Mapper*. This step involves defining a 1-to-1 mapping between the constructs of the high-level modeling language and their corresponding pairs in the semantic library. The resulting semantic model is a refinement of the high-level constructs/types specified in the semantic library, inheriting all the intricate details of its semantics, as well as the semantic constraints associated with the various layers of used semantic libraries.

The *Semantic Model Unfolder* transforms the semantic model into an analysis model that can be directly processed by a model checker backend. During this process, semantic constraints are validated to ensure that the model adheres to the rules defined in the semantic library. Any violations halt the process, returning a *Constraint Violation* that provides detailed information about the issue.

The transformed analysis model is fed into a model checker backend, which verifies the model's adherence to the formal specifications. The model checker generates an execution trace or counterexample, which is then back-annotated to the semantic model level. The resulting *Semantic witness* is a refinement of the original Semantic model and serves as a witness to demonstrate the violation of the checked semantic property. Finally, the semantic witness is mapped back into the high-level modeling language representation by the *Syntactical Back-annotator*, which is a 1-to-1 mapping between the language and semantic library constructs, similarly to the syntactical mapper.

Note that both the syntactical mapper and the syntactical back-annotator are simple mapping layers on top of the semantic model; all the heavy lifting is done by the Semantic Model Unfolder and the Semantic Back-annotator, based on the modeled semantics in the used semantic library.

B. Constraint Validator

Ensuring none of the semantical constraints are violated is the task of the *Constraint Validator*. It talks directly to the *Semantic Model Unwrapper* since a constraint could be violated during the process of unfolding the model at any time, e.g., when using multiple layers of semantic libraries. When a constraint is violated, the unfolding process is halted, and a *Constraint Violation* is returned with the current state of the unwrapped model, containing all necessary information for understanding and fixing the issue.

C. Model Checker

Rather than explicitly depending on a single model checker, state-of-the-art verification tools use a portfolio of model checkers [3]. The benefit of the portfolio approach is that specific model checker algorithms may be better at some tasks but worse at others. Running a portfolio of them seems to lessen this effect without radical performance cost. For this reason, the specific model checker (or portfolio) is left as an implementation detail of the V&V framework, provided it supports the specific analysis representation returned by the Semantic Model Unfolder.

By creating model-checker-specific semantic libraries (at the bottom of the semantic library chain), and encoding certain model-checker-specific semantic constraints corresponding to the strengths and weaknesses of the given model checker (such as concurrent behavior or the use of clocks), the best backends can be selected based on the characteristics of the model under verification.

VI. Feasibility

To demonstrate the feasibility of the proposed V&V framework, we realized a proof-of-concept version using Semantifyr [34] and Viatra [1], and demonstrate its use with the OpenMBEE Space Mission model [28]. The implementation realizes specific parts of the proposed framework, Figure 4 depicts the exact components used in the implementation. OXSTS is the language used to model the Semantic libraries and the Semantic models. Semantifyr is used as the Semantic Unfolder and Backannotator components, Viatra Query Engine is used as the Constraint Validator, and Theta is used as the model checker backend.

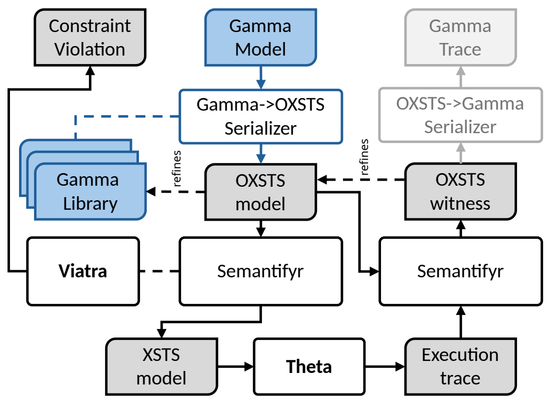

Fig. 4. A PoC realization of the proposed V&V Framework for the Gamma language.

A. The OXSTS Language

Ontological XSTS (OXSTS), introduced in [34], is a declarative-operational language built upon the XSTS [12] formalism, providing a meta-programming layer over the analysis formalism. OXSTS supports high-level modeling constructs, such as types and features, inheritance, and usage-oriented modeling, but also supports the formal operational constructs of the XSTS language. Listing 1 depicts the State type of the Gamma semantic library implemented in OXSTS. Every state has a parent state and region, and may contain entry and exit actions and inner regions.

```
type State {
 reference parent : Region[0..1]
 reference parentState : State[0..1]
 containment regions : Region[0..*]
 containment entryActions : Action[0..*]
 containment exitActions : Action[0..*] ... }
```

Listing 1. The State type in OXSTS.

OXSTS supports the definition of Viatra-like graph patterns, serving as semantic constraints. Listing 2 depicts a semantic constraint disallowing the use of parallel regions in statecharts. Any match of the *noParallelRegions* pattern indicates an ill-formed model since it is defined as a *negative constraint*.

```
neg constraint pattern noParallelRegions(
    state: State) { State.regions(state, r1)
  State.regions(state, r2); r1 != r2 }
```

Listing 2. Semantic constraint disallowing parallel regions in OXSTS.

B. Proof-of-concept Framework

Semantifyr is an open-source[1] proof-of-concept tool capable of instantiating OXSTS models and unfolding them into XSTS, mapping them directly to the operational analysis formalism, while validating the vql-like graph patterns defined in the semantic library. It can also back-annotate the execution traces returned by the Theta model checker into OXSTS representation.

[1]https://github.com/ftsrg/semantifyr

As part of the proof-of-concept, we implemented an initial Gamma semantic library and the corresponding Gamma-OXSTS serializer component, realizing the Syntactical mapping component of the proposed V&V framework.

C. Space Mission

We used the Gamma version of the Simple Space Mission model [28] (presented in [11]) since we have not yet implemented a SysML frontend, and the Gamma semantics provides a simpler illustration of the proposed framework.

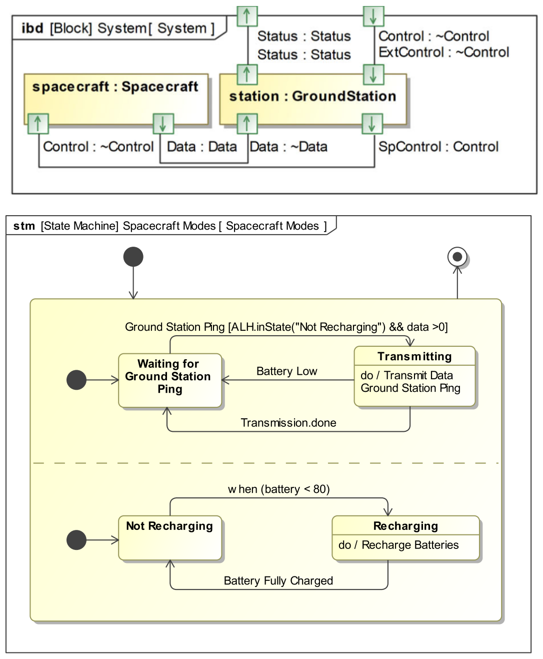

Fig. 5. The Space Mission system model and the State Machine of the Spacecraft component.

Figure 5 shows the Space Mission system and the Spacecraft component's State Machine implementation. The system composes the spacecraft and ground station components, connected via ports. The model realizes a limited-charge battery scenario, where the spacecraft component must send all observation data while maintaining an operational battery charge. With the implemented semantic library and Gamma-OXSTS mapper, we evaluated static and dynamic properties of the Space Mission model using the Viatra and Theta tool integrations of the Semantifyr framework. All the specified verification cases and constraints passed our tests, proving the feasibility of the proposed V&V framework. The relevant artifacts (Gamma, OXSTS, and test cases) are provided at [35].

VII. Conclusion

Critical cyber-physical systems are often modeled using the MBSE paradigm, as the use of models increases understanding, enables early fault detection, and increases quality through the use of various verification and validation techniques. However, creating appropriate V&V tools for custom domain-specific languages usually entails the recreation of the whole transformation chain, as the intricate semantic variances between languages render the existing V&V solutions non-reusable. In this paper, we proposed a configurable V&V framework that addresses these challenges by leveraging semantic libraries to define language semantics in a modular way that only requires the semantic library to be modeled and a frontend connecting it to the engineering language. By extending semantic libraries with constraints, our approach allows the specification of semantic assumptions as part of the semantic library, allowing the configuration of the language semantics by switching the used libraries. We demonstrated the feasibility of the framework through with a proof-of-concept implementation and an example model. In the future, we plan to realize and validate the approach by performing case studies on existing languages, such as SysML v2.

References

[1] Gábor Bergmann, István Dávid, Ábel Hegedüs, Ákos Horváth, István Ráth, Zoltán Ujhelyi, and Dániel Varró. Viatra 3: A Reactive Model Transformation Platform. In Dimitris Kolovos and Manuel Wimmer, editors, *Theory and Practice of Model Transformations*, pages 101–110, Cham, 2015. Springer International Publishing.

[2] Marco Bernardo, Lorenzo Donatiello, and Paolo Ciancarini. Stochastic Process Algebra: From an Algebraic Formalism to an Architectural Description Language. In Maria Carla Calzarossa and Salvatore Tucci, editors, *Performance Evaluation of Complex Systems: Techniques and Tools*, pages 236–260, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[3] Gianpiero Cabodi, Sergio Nocco, and Stefano Quer. Benchmarking a model checker for algorithmic improvements and tuning for performance. *Formal Methods in System Design*, 39(2):205–227, Oct 2011.

[4] Xiaohong Chen and Grigore Roşu. *K - A Semantic Framework for Programming Languages and Formal Analysis*, pages 122–158. Springer International Publishing, Cham, 2020.

[5] Ajay Chhokra, Sherif Abdelwahed, Abhishek Dubey, Sandeep Neema, and Gabor Karsai. From system modeling to formal verification. In *2015 Electronic System Level Synthesis Conference (ESLsyn)*, pages 41–46, 2015.

[6] Antonio Cicchetti, Federico Ciccozzi, Silvia Mazzini, Stefano Puri, Marco Panunzio, Alessandro Zovi, and Tullio Vardanega. CHESS: a model-driven engineering tool environment for aiding the development of complex industrial systems. In *Automated Software Engineering*, pages 362–365. ACM, 2012.

[7] Márton Elekes, Vince Molnár, and Zoltán Micskei. Assessing the specification of modelling language semantics: a study on UML PSSM. *Software Quality Journal*, 31(2):575–617, Jun 2023.

[8] Sanford Friedenthal and Roger Burkhart. Evolving SysML and the System Modeling Environment to Support MBSE. *INSIGHT*, 18(2):39–41, 2015.

[9] Mario Gleirscher and Diego Marmsoler. Formal methods in dependable systems engineering: a survey of professionals from Europe and North America. *Empirical Software Engineering*, 25(6):4473–4546, Nov 2020.

[10] Iris Graessler, Dominik Wiechel, Deniz Oezcan, and Patrick Taplick. Tailored metrics for assessing the quality of MBSE models. *Proceedings of the Design Society*, 4:2545–2554, 2024.

[11] Bence Graics, Vince Molnár, András Vörös, István Majzik, and Dániel Varró. Mixed-semantics composition of statecharts for the component-based design of reactive systems. *Software and Systems Modeling*, 19(6):1483–1517, Nov 2020.

[12] Bence Graics, Milán Mondok, Vince Molnár, and István Majzik. Model-based testing of asynchronously communicating distributed controllers using validated mappings to formal representations. *Science of Computer Programming*, 242:103265, 2025.

[13] Henson Graves and Yvonne Bijan. Using formal methods with SysML in aerospace design and engineering. *Annals of Mathematics and Artificial Intelligence*, 63(1):53–102, Sep 2011.

[14] Kaitlin Henderson and Alejandro Salado. Value and benefits of model-based systems engineering (MBSE): Evidence from the literature. *Systems Engineering*, 24(1):51–66, 2021.

[15] Benedek Horváth, Vince Molnár, Bence Graics, Ákos Hajdu, István Ráth, Ákos Horváth, Robert Karban, Gelys Trancho, and Zoltán Micskei. Pragmatic verification and validation of industrial executable SysML models. *Systems Engineering*, 2023.

[16] Lennart C.L. Kats and Eelco Visser. The spoofax language workbench: rules for declarative specification of languages and IDEs. *SIGPLAN Not.*, 45(10):444–463, 2010.

[17] John C. Knight. Challenges in the utilization of formal methods. In Anders P. Ravn and Hans Rischel, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 1–17, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[18] Martin Kölbl, Stefan Leue, and Hargurbir Singh. From SysML to Model Checkers via Model Transformation. In *Proc. of the 25th International Symposium on Model Checking Software*, volume 10869 of *Lecture Notes in Computer Science*, pages 255–274. Springer, 2018.

[19] Qin Ma, Monika Kaczmarek-Heß, and Sybren de Kinderen. Validation and verification in domain-specific modeling method engineering: an integrated life-cycle view. *Software and Systems Modeling*, Oct 2022.

[20] Vince Molnár, Bence Graics, András Vörös, István Majzik, and Dániel Varró. The Gamma statechart composition framework: Design, verification and code generation for component-based reactive systems. In *Proceedings of ICSE'18: Companion Proceeedings*, pages 113–116. ACM, 2018.

[21] Vince Molnár, Bence Graics, András Vörös, Stefano Tonetta, Luca Cristoforetti, Greg Kimberly, Pamela Dyer, Kristin Giammarco, Manfred Koethe, John Hester, Jamie Smith, and Christoph Grimm. Towards the Formal Verification of SysML v2 Models. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS Companion '24, page 1086–1095, New York, NY, USA, 2024. Association for Computing Machinery.

[22] Object Management Group. Systems Modeling Language (SysML), 2012.

[23] Object Management Group. Unified Modeling Language (UML-v2.5.1), 2017.

[24] Object Management Group. Precise Semantics of UML State Machines (PSSM-v1.0), 2019.

[25] Object Management Group. Semantics of a Foundational Subset for Executable UML Models (fUML-v1.2), 2021.

[26] Object Management Group. *Kernel Modeling Language (KerML)*. Object Management Group, 2023. ptc/23-06-01.

[27] Object Management Group. *OMG Systems Modeling Language (SysML v2)*, 2023. ptc/23-06-02.

[28] OpenMBEE. OpenMBEE Simple Space Mission. https://github.com/Open-MBEE/OMGSpecifications, 2019. Accessed: 2024-03-29.

[29] Charlotte Seidner and Olivier H. Roux. Formal Methods for Systems Engineering Behavior Models. *IEEE Transactions on Industrial Informatics*, 4(4):280–291, 2008.

[30] Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. Theta: A Framework for Abstraction Refinement-Based Model Checking. In *Proceedings of FMCAD'17*, page 176–179, Vienna, Austria, 2017. FMCAD Inc.

[31] Emile van Gerwen, Leonardo Barbini, and Thomas Nägele. Integrating System Failure Diagnostics Into Model-based System Engineering. *INSIGHT*, 25(4):51–57, 2022.

[32] Willem Visser, Matthew B. Dwyer, and Michael W. Whalen. The hidden models of model checking. *Software and Systems Modeling*, 11(4):541–555, 2012.

[33] Tim Weilkiens. *SYSMOD-the systems modeling toolbox: Pragmatic MBSE with SysML*. MBSE4U, 2020.

[34] Ármin Zavada, Kristóf Marussy, and Vince Molnár. From transpilers to semantic libraries: Formal verification with pluggable semantics. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS Companion '24, page 311–317, New York, NY, USA, 2024. Association for Computing Machinery.

[35] Ármin Zavada, Kristóf Marussy, and Vince Molnár. High-level models represented in OXSTS. https://github.com/ftsrg/semantifyr/tree/6eb8dcbb3263935adc7cf1ef98f749209426adc3/subprojects/frontends/gamma/gamma-frontend, 2025.

[36] Ármin Zavada and Vince Molnár. From hard-coded to modeled: Towards making semantic-preserving model transformations more flexible. *31st Minisymposium of the Department of Measurement and Information Systems*, 10:41–45, 02 2024.