REGULAR PAPER

Pragmatic Verification and Validation of Industrial Executable SysML Models

Benedek Horváth[1,2] | Vince Molnár[3] | Bence Graics[3] | Ákos Hajdu[3] | István Ráth[1] | Ákos Horváth[1] | Robert Karban[4] | Gelys Trancho[5] | Zoltán Micskei[3]

[1]IncQuery Labs cPlc., Budapest, Hungary

[2]Johannes Kepler University Linz, Linz, Austria

[3]Deptartment of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary

[4]Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA

[5]TMT International Observatory LLC, Pasadena, CA, USA

Correspondence
Benedek Horváth, IncQuery Labs cPlc., Budapest, Hungary
Email: benedek.horvath@incquerylabs.com

Funding information
European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813884; NRDI Fund of Hungary, financed under the 2019-2.1.1-EUREKA-2019-00001 funding scheme.

Abstract

In recent years, Model-Based Systems Engineering (MBSE) practices have been applied in various industries to design, simulate and verify complex systems. The verification and validation (V&V) of such systems engineering models is crucial to develop high-quality systems. However, this is a challenging problem due to the complexity of the models, and semantic differences in how different tools interpret the models, which can undermine the validity of the obtained results if they go undiscovered. To address these issues, we propose (i) a subset of the SysML language for which the practical semantic integrity of tools can be achieved, and (ii) a cloud-based verification and validation framework for this subset, lifting verification to an industrial scale. We demonstrate the feasibility of our approach on an industrial-scale model from the aerospace domain, and summarize the lessons learned during transitioning formal verification tools to an industrial context.

KEYWORDS

MBSE, SysML, formal verification, model checking, hidden formal methods

1 | INTRODUCTION

Model-Based Systems Engineering (MBSE) [49] is a widely used discipline to design and implement complex systems. MBSE promotes models from static documents to executable first-class citizens [67], connecting the steps of the whole engineering lifecycle with explicit traceability information: starting from requirements through design, simulation and verification, until implementation and reporting [65]. A recent trend in MBSE is moving from standalone tools to integrated environments connecting various tools working on linked models. OpenMBEE [43] is a community providing core components to realize the vision of connected models and environments, which are used by organizations such as the European Southern Observatory (ESO) or NASA Jet Propulsion Laboratory (JPL).

Context

The *NASA Jet Propulsion Laboratory* (JPL) develops complex robotic space missions. JPL has started to adopt MBSE by transforming its document-centric engineering processes to integrated, collaborative model-based practices, which have been already used in several flight missions, e.g., the Mars 2020 Perseverance rover. JPL missions incorporate a wide variety of languages and tools. To support its engineers, JPL offers common engineering environments that connect these tools and models through cloud-based services. This paper is concerned with the models and ecosystem around the *Systems Modeling Language* (SysML) [57].

Motivation

SysML supports the architectural design and analysis by providing structural and executable behavioral models [60, 40, 41]. Requirements are usually validated by simulating operational scenarios on SysML state machines and activity diagrams. However, as these SysML models are complex, simulating a handful of traces is usually not enough to find subtle design errors in the models. Many organizations have used *model checking* [10] – an automated formal verification technique that systematically traverses execution paths in the model – to verify design models [23, 22]. The widespread adoption of formal methods is still limited due to two main reasons. First, model checkers usually require expertise in formal methods (e.g., in aligning the semantics of formal and engineering models). Second, even modern tools require a significant amount of computing resources, which may not be easily available on end-user devices, e.g., workstations or mobile appliances.

Objectives

Based on the motivations above, we aim to extend verification and validation (V&V) capabilities for SysML models in the OpenMBEE ecosystem. Our goal is to propose an "off-the-shelf", scalable, and multi-user framework that checks properties of the model and offloads the resource-intensive verification tasks to cloud-based services. Recognizing the difficulties in realizing an ideal solution, we define the *long-term goals* as (1) operating at an industrial scale (e.g., hundreds of actions in a state machine), (2) supporting frequently used modeling constructs (e.g.,, communicating state machines), and (3) being deployable in an enterprise environment as a common service used by teams of engineers.

Method

Driven by these goals, we conducted several interviews and focus group meetings with subject matter experts from several organizations during 2020. As a result, we identified a prioritized list of objectives and modeling language elements that need to be supported. We reviewed the relevant fUML and PSSM specifications [58, 56] to reveal potentially challenging elements with respect to V&V. We studied the modeling practices and patterns employed by engineers. We worked with one of the largest open-source SysML model, the model of the Thirty-Meter Tele-

scope (TMT) [11, 37], which applies the practices of the OpenSE Cookbook [39]. We experimented with industry-leading SysML tools to simulate these models. As a consequence, we adopted an attitude of favoring practicality and performance over completeness and precision: certain elements are not supported because they unnecessarily complicate verification, the semantics of certain elements are simplified to the most common interpretations, and thus the explorable state space is reduced. We counter the reduction in completeness with detailed feedback through verification-specific validation rules. This process was supported by iteratively building prototypes that eventually grew into a cloud-based framework that can validate and verify SysML models using the selected subset of elements.

Contributions

We make the following contributions towards the V&V of industrial executable SysML models:

- We recommend a pragmatic subset of SysML behavioral modeling elements (state machines and activity diagrams) that can be validated and verified even at an industrial scale (Section 3).
- We develop a V&V workflow and cloud-based framework based on IncQuery Suite [29] for model validation and transformation, using the Gamma Framework [52] as an intermediate format for the Theta [62] and UPPAAL [5] model checkers. The framework enables verification to be used as a common service by many users on a scalable platform (Section 4).
- We demonstrate the workflow on the TMT model in an illustrative multi-user scenario (Section 5).
- We summarize the key lessons learned while developing the V&V workflow (Section 6).

In this paper, we extend our initial prototype [33] with both theoretical (a pragmatic subset of the SysML language, supporting activity diagrams describing detailed actions) and engineering (moving to a new architecture backed by Kubernetes) contributions that made it possible to verify a SysML model an order of magnitude larger than in the previous workshop paper. This is an important milestone, as real-world projects are generally not accessible until one can show convincing results – posing a significant challenge in developing such solutions. By reaching this level of maturity through the collaboration of a diverse team, the possibility of experimenting with confidential models of the industry finally opens up.

Lessons learned

This paper reinforces the lessons learned from successful academic-industrial collaborations [25, 21] in the context of MBSE and V&V.

1. Supported modeling elements are constrained by practice, not convenience: constructs frequently used by engineers cannot be ignored.
2. Single-user desktop applications are not applicable in industrial settings: requirements like access control, multi-user jobs, and deployability are not just optional (even though they are not “novel”).
3. A tool must be useful for the end-users, not theoretically complete: it is better to successfully discover one actionable issue and ignore others than to fail while trying to find all of them (even if this is a “limitation”).

2 | INDUSTRIAL CONTEXT

Model-Based Systems Engineering (MBSE) is a widely used discipline to design and implement complex systems in the aerospace and other domains [38]. NASA JPL has approximately 4000 engineers developing and deploying complex

cyber-physical and aeronautical systems. JPL employs *multi-paradigm modeling* using SysML, Matlab/Simulink and Python. A single vendor rarely possesses the functionality needed for the whole modeling lifecycle, thus products of multiple vendors are integrated into a toolchain [42].

Recently, many organizations have started to utilize *cloud-based, collaborative environments* to integrate modeling and analysis tools. Such platforms provide various viewpoints to the different stakeholders and keep system models consistent and traceable. In order to achieve the interoperability of such tools in a cloud-based environment, they should provide a web-based API and leverage technologies that are used to ease the deployment of the applications to the cloud, e.g., Kubernetes[1]. A prime example is the open-source *OpenMBEE* project[2], an open environment for connected models. NASA JPL is a member of the project among many other companies and organizations in the aerospace domain, e.g., Boeing, ESO. OpenMBEE includes a model repository (MMS), model authoring tool adapters for syncing models to the repository (MDK), and a web-based environment to interact with views and documents generated from the model (View Editor), as depicted in Figure 1. Although it supports the design phase with several tools, there is no explicit tool support for verification and validation aspects in the ecosystem.

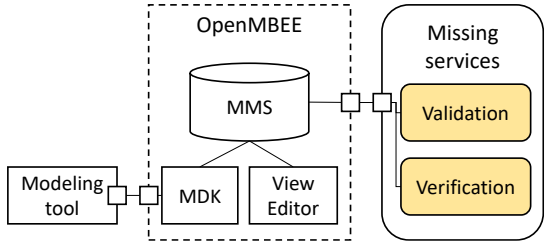


FIGURE 1 Open Model Based Engineering Environment (OpenMBEE) architecture.

Systems engineering in SysML is supported by several tools. Models are created for example in MagicDraw[3] or Cameo Systems Modeler[4], and are versioned in central repositories (MMS or Teamwork Cloud[5]). Cameo Simulation Toolkit (CST)[6] is used to simulate models and validate timing or functional properties. Documents and reports are generated by the View Editor, later to be consumed by other engineering teams. Implementation code is either created manually or by code generation (e.g., with COMODO [3]).

The *OpenSE Cookbook* [39] collects modeling practices and processes proven to be useful in industrial practice. Many organizations follow the *Executable Systems Engineering Method* (ESEM), emphasizing executable models to deliver high-quality systems. Textual requirements are formalized using constraints and parameters. System behavior is modeled using state machines communicating through ports. The detailed behavior of states and transitions are defined using activity diagrams. Operational scenarios can be defined as sequence diagrams, driving simulations to obtain e.g., timing diagrams or other analysis results.

The largest, open-source, industrial-scale application of the OpenSE Cookbook design patterns is the *Thirty-Meter Telescope* (TMT) [11]. Its objectives are to adopt the design principles used in aerospace flight missions at NASA JPL, and to provide mass and power roll-up analysis by simulating the system design to validate the requirements [37].

[1] Cloud Native Computing Foundation. Kubernetes `https://www.kubernetes.io`

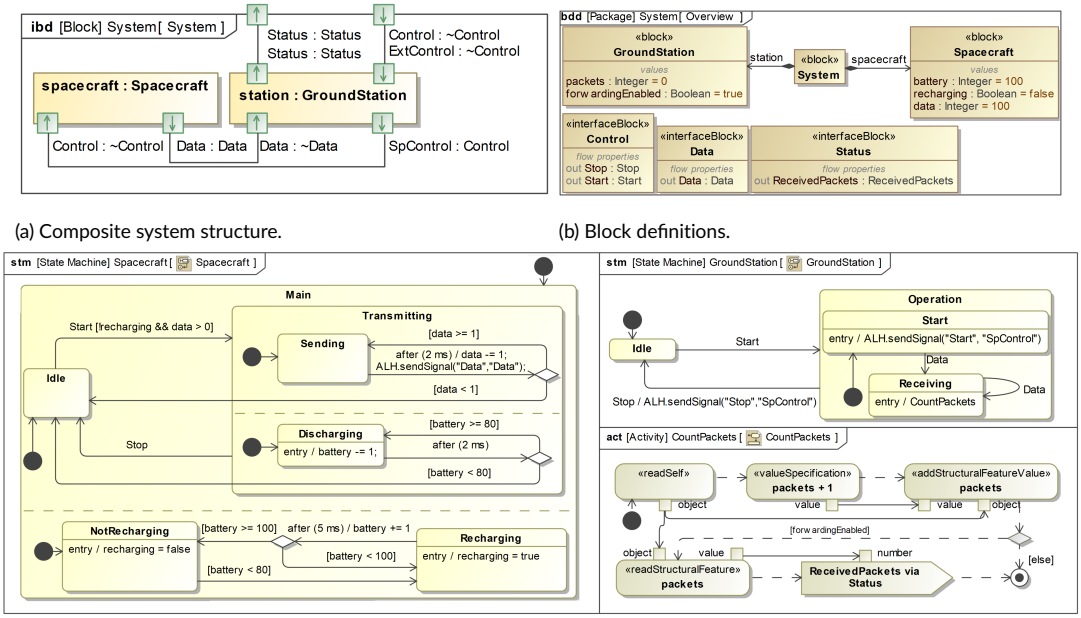
[2] Open Model Based Engineering Environment (OpenMBEE) `https://www.openmbee.org`

[3] Dassault Systèmes. MagicDraw `https://www.nomagic.com/products/magicdraw`

[4] Dassault Systèmes. Cameo Systems Modeler `https://www.3ds.com/products-services/catia/products/no-magic/cameo-systems-modeler/`

[5] Dassault Systèmes. Teamwork Cloud `https://www.nomagic.com/products/teamwork-cloud`

[6] Dassault Systèmes. Cameo Simulation Toolkit. `https://www.3ds.com/products-services/catia/products/no-magic/cameo-simulation-toolkit/`



(a) Composite system structure.

(b) Block definitions.

(c) Behavior definitions.

FIGURE 2 A SysML model describing a simplified Spacecraft and Ground Station to illustrate the scope.

2.1 | Running example demonstrating industrial modeling practices

As a running example model in the paper, let's consider the composite system of a simplified Spacecraft and Ground Station model in Figure 2 based on OpenSE Cookbook. As soon as the `Ground Station` is in `Operation`, it notifies the `Spacecraft` to `Start` sending `Data`. The station counts and forwards the incoming data packets via its `Status` port. The `Spacecraft` is waiting in the `Idle` state until it receives a `Start` signal from the station to start sending data in packets. Data transmission consumes 1% of `battery` power per packet, and if the battery level falls below 80%, ongoing data transmission is paused until a full recharge. Several properties can be formulated about the behavior of the system. For illustrative purposes, let's consider that the Spacecraft[7]

(a) only starts transmitting when receiving a `Start` signal,

(b) never transmits when the `battery` is below `80%` and the Ground Station has already received at least `20 packets`.

While property *(a)* could be checked in principle by reviews or validation rules, property *(b)* is much harder as we have to consider all paths in the model. A handful of executions can be explored in a simulator (e.g., CST), however engineers could easily get lost if finding a trace requires solving complex conditions or ordering concurrent behaviors.

These properties can be captured as *reachability properties* on state machines of the composite system. A reachability property describes a *state configuration* with predicate logic (that can refer to state nodes and state variables). A reachability property can be *checked*: a model checker can compute whether a state configuration matching the property can be reached from the initial state or not. A requirement may be derived from a reachability property by specifying whether the state configuration is *desirable* (a liveness requirement) or *undesirable* (a safety requirement). A liveness requirement is satisfied by a system if there exists an execution path (a *witness*) that leads the system into

[7] These properties were derived to illustrate the challenges during verification.

a matching configuration. In contrast, a safety property is violated if there is such an execution (which in this case is called a *counterexample*). Note that a safety requirement can be converted into a liveness requirement by negating both the description of the state configuration and the result of model checking.

A SysML sequence diagram can be used as a surface notation to define a reachability property. The diagram consists of *Lifelines* representing the Part Properties of the composite Block it describes. Each Lifeline can contain several *State Invariants*. The property, illustrated by Figure 3, corresponds to property *(b)*, describing a configuration where the `Transmitting` state of the Spacecraft is active, the `battery` level is below `80`%, the Ground Station is in the `Receiving` state and has already received at least `20` `packets`.

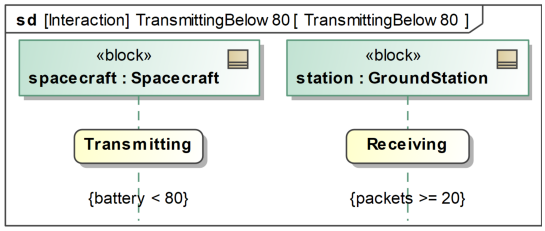


FIGURE 3 A Sequence Diagram describing the reachability property *(b)*.

2.2 | V&V of industrial SysML models: State of the art

The correctness of systems engineering models is a crucial aspect in the aerospace and astronomy domains, due to the large design, development and operational costs, the longevity and the safety critical properties of the systems. There are several techniques for validating the designs [23, 22]. During manual reviews, engineers inspect the models and documentation. Static validation rules are used to check the structural consistency of the models. Simulations are used to evaluate executions of the behavior or run test cases. The primary result of simulation is one or more execution traces, which can be used to analyze different qualitative and quantitative properties.

Testing and simulation are proven methods, but they depend on the engineer's expertise, and in case of complex models, have a chance of missing problems (partly mitigated by test design methodologies). This has inspired research in formal methods to support verification with complementary approaches that reduce this chance by systematic, (semi-)automated reasoning. One such approach is model checking, which can be viewed as an automated and highly optimized exploratory simulation driven by a declaratively specified goal. Traditionally, this goal is to demonstrate the satisfaction or violation of a requirement with a suitable execution (witness). In case of a reachability property, the goal is to look for traces that reach a state configuration specified by the property. Therefore, model checking can be used to enhance a set of hand-made operational cases with machine-assisted exploration of hard-to-find corner cases. Note that from the engineer's point of view, the result of model checking should be an execution trace similar to the result of simulation or testing, which can be used to analyze the behavior. An example for this practice is NASA JPL's usage of Java Pathfinder [28] and Spin [32] to verify properties on code generated from state machines [23].

Tool integration is an important aspect in the applicability of model checking in industrial settings. Typically, engineers work with feature-rich tools that support the authoring of high-level, generic and flexible models, while model checkers are often single-user applications with a lot of configuration options and a mathematically precise, low-level input language. This gap is probably the most dominant obstacle in the industrial adoption of results accumulated during decades of formal methods research.

Given this task of integrating (primarily academic) formal verification tools with industrial authoring environments, we can identify several levels:

1. The design models and formal models are built separately by a group of specialists (e.g., [20]). Traceability can be added manually.
2. The formal models and traceability information are derived from the design models via automated model transformation (e.g., [44]). Results are still evaluated by experts familiar with the formal model and tool.
3. In addition to automatic generation, results are also automatically back-annotated into the design model [30], which allows the complete workflow to be executed from the design environment.

An orthogonal aspect is how much of the language elements is supported by the verification workflow. An ideal solution would reach level *3.* for the whole language – in our case, SysML. Since this is extremely challenging due to the richness and flexibility of the language, most solutions deal with only a fragment of the modeling elements.

2.3 | Objective and challenges

We defined the vision to extend an industrial SysML-based engineering environment with capabilities to check properties defined over state machine models using formal verification techniques, with the following objectives:

1. The approach shall check reachability properties over executable SysML models to provide early feedback to engineers in a language they are familiar with.
2. Verification shall be hidden and offloaded to cloud services supporting multiple model checkers.
3. The solution shall be integrated with *OpenMBEE* and support a reasonable subset of widespread modeling practices.
4. The solution shall satisfy the entry conditions of typical enterprise deployments, i.e., multi-user and parallel jobs, containerized environment, and access control.

Our *main challenges* were that (1) there are subtle differences in how different tools and engineers interpret SysML models or even the standard itself, (2) there is a significant gap between the models of computation of SysML models and most formal languages, and (3) formal verification approaches generally do not scale to larger models.

We have started our collaboration to tackle these challenges in 2019, with the identification of the modeling elements to be supported by an acceptable solution. In 2020, we implemented the first prototype presented in a workshop paper [33]. In the autumn of 2020, we extended the prototype with support for activity diagrams and communicating state machines, reaching a set of supported elements that is sufficient to tackle real-world models. In 2021, with major engineering efforts put into the optimization of the transformation and a technological upgrade to a cloud-native architecture, the solution has been scaled to the TMT model and industrial deployment became feasible.

In this paper, we present the details of this journey. The prototype at its current state is sufficiently evolved to serve as a demonstration that verification of industrial SysML models is feasible. The distinguishing feature of our approach is that the heterogeneous team resulted in a healthy balance between academic and industrial interests and constraints, driving the development towards the needs of end users while respecting the capabilities of existing verification tools. We call the resulting compromise *a pragmatic V&V approach for SysML*.

3 | A PRAGMATIC V&V APPROACH FOR SYSML

3.1 | Challenges of verifying SysML models

SysML and UML semantics

Because SysML inherits many elements and semantics from UML, in this chapter we explore SysML and UML together. Both standards [57, 54] have been defined to ensure that the different tools in this workflow process models the same way. However, standards are usually ambiguous because standardization needs to find a compromise between human intelligibility and formal preciseness. This flexibility may lead to subtle differences in how the models are interpreted by both stakeholders and tools. For example, a verification tool may return an execution trace that is not reproducible by the simulator. To address this issue, several standards have been proposed as an extension to the base standards (e.g., PSSM [56], PSCS [55], fUML [58]). These standards define a detailed operational semantics, capture some of the known variations explicitly as *semantic variation points*, and contain test suites to demonstrate the expected behavior of model elements. However, as the main parts of the standards are still semi-formal natural text, they still could not completely eliminate ambiguity and specify every possible corner case.

To complement these standards, researchers have proposed to map UML/SysML to a formal analysis domain. These formalizations differ in the types of supported elements: ranging from handful of elements in activity diagrams [47] and state machines [12] to rich feature support for UML state machines [48]. The approaches also use several alternatives for the mathematical basis: e.g., Petri nets [34], PROMELA [45], CML [47] or Kripke structures [46]. However, these formalizations are usually not complete [12].

According to our experience, in an industrial setting further practical perspectives needs to be considered apart from the above theoretical one. (1) In a large organization, *model users* might interpret the same diagram differently or use certain modeling constructs in conflicting ways. This might result in inconsistent, non-reusable models if modeling guidelines and validations are not used [4, 27]. (2) Some industrial design, simulation and verification *tool developers* strive for for maximum standard compliance, but this is hard to achieve due to the imprecise semantics that makes it difficult to correctly implement the corner-cases. [18, 13, 16]. (3) Furthermore, the integration of these tools are difficult in an industrial environment, because the execution traces provided by them might not be consistent with each other. For example, a formal verification tool might find a trace trace that cannot be produced by the simulator or the model user might come up with an execution trace that is not possible due to some limitations of the language [17].

Challenges

To summarize, we observed these challenges when using formal verification for executable SysML state machines:

1. The SysML and UML standards are complex with many elements and intricate details.

2. Several constructs that are used in practice result in demanding verification problems (e.g., conflicting read-write access to shared variables in orthogonal regions or calling arbitrary source code in effects).

3. MBSE tools might handle certain constructs differently. For example, it is a serious risk if the execution order of orthogonal regions is handled differently in the model simulator, code generator, and model checker.

4. Humans might use and interpret certain constructs in a non-standard way, e.g., priorities of transitions or aborting a do activity, that may conflict with the result of the verification.

Semantic integrity

In an ideal setting, there would be full semantic integrity across the whole organization and engineering lifecycle: simulation, verification, code generation tools, execution platforms and engineers would interpret the models consistently, and the behavior of an artifact produced in one tool would conform to the behavior in other tools. However, ensuring such semantic consistency across a supply chain or an organization is an extremely challenging task despite the efforts of standardization bodies, academic researchers, tool vendors, and even domain experts. Therefore, we aimed for consistency with a *pragmatic subset of SysML* and a *particular integrated toolchain*, where a validation and verification tool shall report problems that can be discovered with a simulator or explained to systems engineers.

3.2 | Pragmatic semantic integrity

As discussed in the previous section, achieving the semantic integrity of a full MBSE lifecycle is an overly ambitious goal. Therefore as a first step we aimed for a pragmatic approach by making the following compromises: we focused (1) on the V&V of the SysML-based system-level design phases, (2) with a fixed set of tools, (3) for a subset of the SysML language, (4) by using syntactic and semantic restrictions that can limit the ambiguity, and the possible state space of the behavioral models to make formal verification scalable. However, we made sure that the resulting approach is not too limited from a practical point of view, and engineers could use it to design real systems. Thus sometimes we included language elements for which semantic integrity could not be guaranteed but are required by existing modeling practices. In these cases we give warnings that explain the possible differences in the interpretations.

Pragmatic semantic integrity

The pragmatic semantic integrity of a given toolchain means that the tools in the workflow interpret the models the same way, or provide means to explain the differences. In this way, systems engineers are informed that the results are consistent with each other and with the modeling practices, considering the restrictions of the respective tools.

Example

A good example for making such pragmatic choices is handling orthogonal regions in state machines. According to the official PSSM standard, the execution of effects and related actions can interleave in any order for transitions in different regions even in one run-to-completion step, resulting in many possible traces. Such concurrency could cause state space explosion in complex models. Nevertheless, orthogonal regions shall be supported as they are essential in industrial models. Thus we restricted the execution of orthogonal regions to a strict sequential order in formal verification. Simulation tools usually also choose to have a deterministic execution order and produce only one of the possible traces. However, this order is not documented, therefore the simulation might produce a trace different from the one returned by formal verification. Thus our approach gives a warning if orthogonal regions are used in a model.

Method

We used the following method to achieve the pragmatic semantic integrity among the selected tools and to compile a pragmatic subset of the SysML language. We

1. manually inspected of relevant SysML, fUML and PSSM standards, to collect the most important concepts;

2. conducted interviews and focus group meetings with engineers from NASA JPL, ESO and OpenMBEE to learn about their engineering practices;

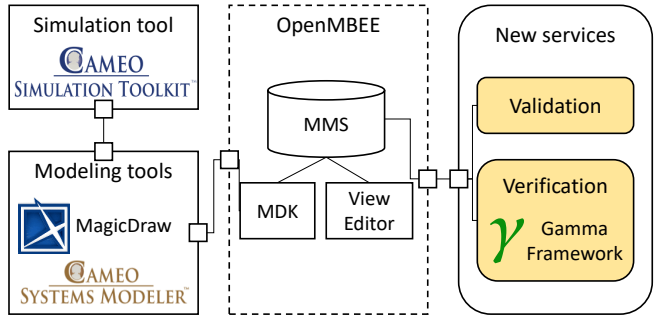


FIGURE 4 Selected toolchain of design (MagicDraw, Cameo Systems Modeler), simulation (Cameo Simulation Toolkit), and repository (OpenMBEE) tools extended with the new V&V services proposed in this paper.

3. inspected available domain-specific models from TMT and the OpenSE Cookbook to identify the design patterns and most common elements;

4. experimented with selected SysML modeling, simulation and verification tools to discover the differences in their operation (see Figure 4);

5. inspected the used modeling elements from the formal verification's complexity point of view; and

6. finally compiled a pragmatic subset of the SysML language.

As formal verification tools vastly differ in their expressive power and capabilities, we relied on the intermediate language offered by the *Gamma Framework* [52]. The Gamma Statechart Composition and Verification Framework was specifically developed for formal verification of reactive systems, and it offers and a tailored modeling language with precise formal semantics and a common interface supporting several formal verification backends.

Pragmatic subset

The central artifact of our approach is a subset of the SysML language for which the pragmatic semantic integrity can be achieved in case of design and V&V activities. To create this subset we selected first modeling elements that are uniformly and unambiguously supported by each tool. Next, the elements that have different interpretations between the tools or have large performance implications in simulation or verification, but were required due to modeling practice reasons, were included in the subset. The differences among the tools were bridged either by restricting their execution for a common subset, or defining warnings to explain the possible inconsistencies. The elements that had unmanageable differences between the tools were excluded from the subset. This process could be repeated by iteratively updating the pragmatic subset, as the standards, engineering practice and tools evolve.

3.3 | A pragmatic subset of the SysML language

Our goal was to give systems engineers a design-time model validation and verification workflow that is able to check the dynamic behavioral properties of the models in a semantically integrated way. We focused on SysML state machines and activity diagrams, since they are one of the most commonly used formalisms for behavioral modeling.

Element	Restriction
Block, Composite Block, Interface Block	-
Port	-
Parameterized signal	-
Data type	Integer, Boolean, Enumeration Literal
Initial, Choice, Deep and Shallow History pseudostates	-
Simple and composite states	-
Transition with Time Event or Signal Event trigger	-
JavaScript guards, actions	variable assignments, predicates, ALH.sendSignal to send signals, ALH.inState to check if state is active, no loop, no array, no dynamic memory, no other function call is allowed
Transition effect	Activity, or restricted JavaScript
State entry or exit actions	Activity, or restricted JavaScript

(a) State machine and structure elements.

Element	Restriction
Control Flow	loop and recursion free
Data Flow	-
Initial, Activity Final Nodes	there must exist exactly one of each
Decision and Merge Nodes	Decision Node must have a default true branch
Fork and Join Nodes	sequential execution of concurrent Control Flows, that must join in the same Join Node
Call Behavior Action	behavior is Activity, no recursion in the call hierarchy
Send Signal Action	-
Add Structural Feature Value Action	-
Read Structural Feature Action	-
Value Specification Action	Literal Integer or Boolean value, restricted JavaScript expression
Opaque Expressions, Actions	see JavaScript guards and actions

(b) Activity diagram elements.

TABLE 1 A pragmatic subset of the SysML language that is suitable for formal verification.

Although the SysML language gives huge degree of freedom in using and combining the different modeling elements, we had to apply restrictions in order to achieve formal verification in practice. (1) We limited the execution order of orthogonal regions to the same order as they are defined in the model. (2) We forbade do-activities completely as they are running concurrently to the state machines and aborting them is not well-defined in the standard. (3) We forbade arbitrary source code, dynamic memory allocation and loops in guards, actions and effects, because they are well-known difficult problems in software model checking [22]. (4) Although the activity diagrams help engineer's describe complex control and communication logic easily, but their asynchronous and parallel execution impose huge verification complexity. Therefore we simplified them so they can be verified by the Gamma Framework.

Due to some of the restrictions, not all theoretically possible traces are explored during verification (e.g., not all interleavings of events in orthogonal regions). But we can still check a much greater number of traces with these restrictions than what is possible with simulation, while giving warnings when such trade-offs are applied.

The final, refined pragmatic subset of modeling elements are listed in Table 1. State machines communicate with each other using parameterized Signals via Ports; and have timed Transitions. Expressions can include variables and literals of type Boolean, Integer or Enumeration. Activities or simplified JavaScript can be used to express an uninterruptible, loop- and recursion-free Control Flow. We have confirmed with subject matter experts from NASA-JPL, ESO and OpenMBEE that this subset contains the most commonly used elements in the aerospace domain and still enables practitioners to build sufficiently complex, yet verifiable systems engineering models (like the TMT model [11] which adopts design patterns applied at NASA JPL). Language elements left out either can be replaced by a combination of elements from the practical subset, or they pose such complexity for verification that they cannot be reasonably verified anyway.

Although the concrete, pragmatic subset of the SysML language was compiled from models and modeling practices in the aerospace and astronomy domain, but the method we outlined can be applied in other domains as well, because the process is domain-independent.

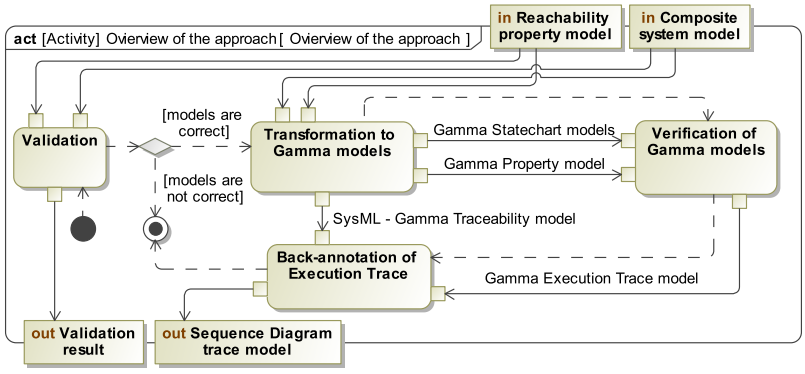


FIGURE 5 Overview of the V&V workflow.

4 | THE V&V WORKFLOW

Figure 5 overviews the workflow supporting the design-time verification and validation of SysML models using elements of the pragmatic subset. The workflow is built around the Gamma Framework: *validating* if the SysML model conforms with the pragmatic subset, *transforming* the model to the input language of Gamma, running *verification* and *back-annotating* the results to the modeling domain.

4.1 | Steps of the V&V workflow

Validation

In the *validation* phase, we check if the selected models conform with the restrictions of our pragmatic subset of SysML. If the model contains unsupported or structurally incorrect elements, then we return *validation errors*. *Warnings* are

returned for elements that are interpreted with restricted semantics. We used the Viatra Query Language (VQL) [63] to implement a comprehensive validation suite consisting of more than 130 imperative and declarative rules. Imperative rules check the JavaScript guard expressions, action and effect statements. Declarative rules are composed as graph patterns and validate the structure of the SysML models.

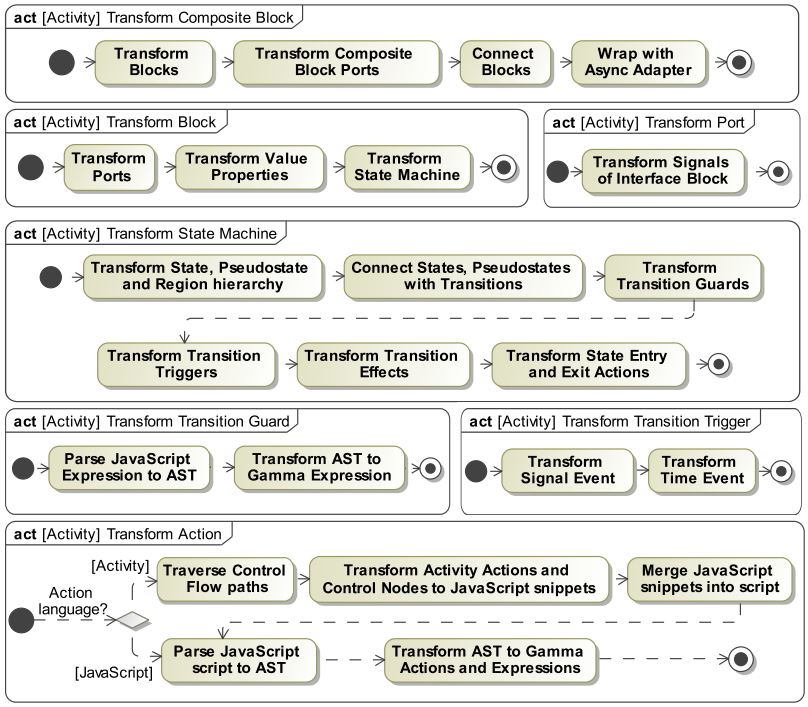


FIGURE 6 Composite Block transformation process.

Transformation to Gamma models

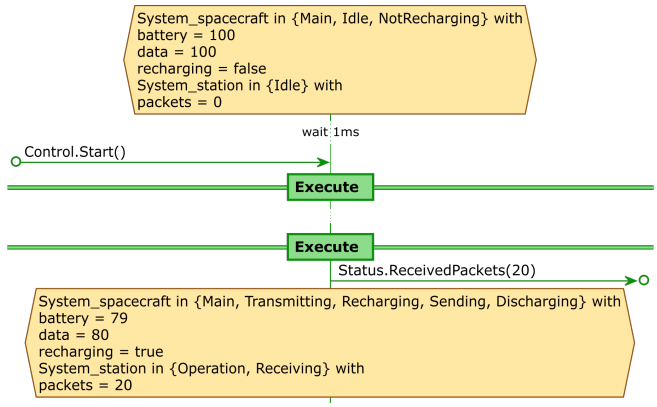
Next, the SysML composite system and reachability property models are transformed to the Gamma Statechart and Property languages, respectively. During the transformation, a traceability model between the SysML State Machine and the Gamma statechart is built to track the mapping between the source and target models. This traceability model is used for transforming the reachability property and back-annotating the verification results to the source domain.

Figure 6 illustrates the transformation process of the SysML composite Block. Each Block of the composite system is transformed to a Gamma state machine. After the transformation of the individual Blocks, the ports of the composite system are transformed and the generated state machines are connected to each other according to the internal structure of the system. Finally, the components are wrapped according to the compositional semantics of Gamma.

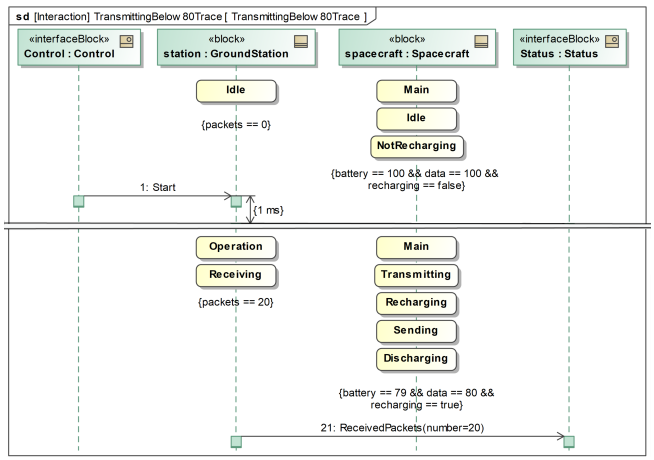
Verification of Gamma models

In the *verification* phase, Gamma transforms the Gamma Statechart Models to formal models and the Gamma Property Model to formal queries specific to the respective model checker. During the translation, several semantics-preserving optimizations reduce the size of the models that reduces the verification time. Such optimizations among others are:

removing the statically unreachable states and unfireable transitions, removing unused variables and events, inlining consecutive variable assignments [26].



(a) Execution Trace in Gamma.



(b) SysML sequence diagram trace.

FIGURE 7 Execution Trace to sequence diagram mapping.

Back-annotation of the Execution Trace

If the reachability property is satisfied, then Gamma returns an Execution Trace (Figure 7a) leading to the target state configuration. Each step of the trace contains: (1) the active state configuration of components including the values of variables (represented by a hexagon), (2) the set of input signals that were received by the composite system (represented by incoming arrows), (3) the set of output signals that were sent by the system (represented by outgoing arrows), and (4) the time that is spent before proceeding to the next step (in case of a timed system, represented by `wait X ms`). The steps in the trace are visually separated by a horizontal line an `Execute` rectangle in the middle.

Using the traceability model created during the forward-transformation, we back-annotated the Execution Trace to a SysML sequence diagram, as depicted in Figure 7b for the running example. On the diagram, we can see that

the composite system is in the initial state configuration, when it receives a *Start* Signal via the *Control* Port and *1 ms* elapses until we get to the next state configuration. After that, due to space limitations, we only represent the last state configuration of the trace. The *spacecraft* is in the *Transmitting* State, it has sent 20 packets to the *station* and its *battery* is 79. The *station* is in the *Receiving* State, it has received 20 *packets* and as a last action it sends the *ReceivedPackets(20)* Message via the *Status* Port of the composite system. Reaching the last state configuration proves that the reachability property of the running example is satisfied.

The back-annotated SysML sequence diagram (Figure 7b), can be used for manual inspection of the execution, and to simulate the state machines in the Cameo Simulation Toolkit (CST).

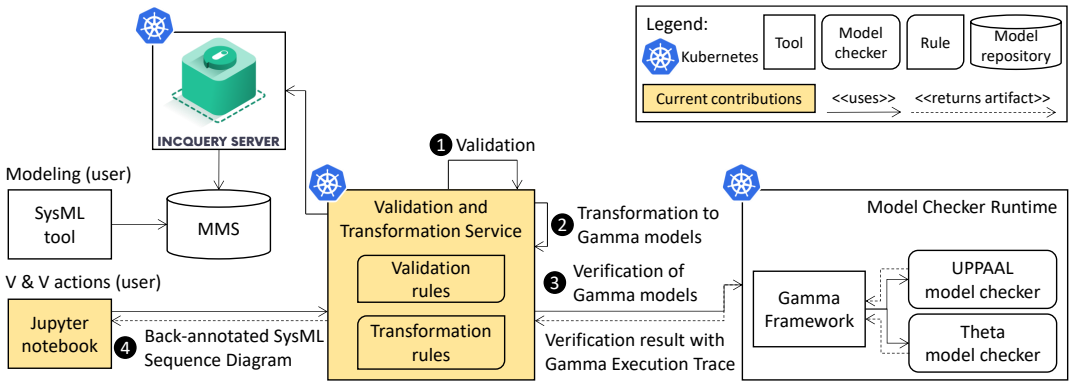


FIGURE 8 Overview of the architecture.

4.2 | Technical realization

We implemented the aforementioned verification and validation (V&V) workflow, using proprietary and open-source tools. The architecture is depicted in Figure 8.

Systems engineers design the state machines, activity diagrams and define the reachability properties in SysML modeling tools, e.g., MagicDraw or Cameo Systems Modeler, and push them to the OpenMBEE MMS model repository. Then, users open a web browser to perform V&V actions in a *Jupyter notebook*[8] serving as a frontend. The frontend is connected to the *Validation and Transformation Service (VTS)* in the backend which validates the SysML models ❶, before transforming them to Gamma models ❷ and verifying them with the Gamma Framework ❸. The verification result is presented in the browser ❹, possibly including a Gamma Execution Trace that is back-annotated to a SysML sequence diagram. (If the reachability property cannot be satisfied, then no SysML sequence diagram is returned.)

On the backend side, we use IncQuery Suite (IQS), a scalable model query middleware on top of collaborative model repositories [29], to improve the performance of the validation and transformation phases. IQS builds an in-memory index from the model and can efficiently evaluate queries implemented in the Viatra Query Language. These queries are used by model validation (Section 4.1) and transformation rules. We implemented several model transformation rules that build the target and traceability models. The transformed models are persisted, and are used in the verification and back-annotation phases of the V&V workflow.

In the verification phase, the Gamma Framework translates the intermediate models to formal models and queries

[8] Jupyter Community. Jupyter `https://jupyter.org`

to be checked by different model checkers, i.e., UPPAAL [5] or Theta [62]. The philosophy of Gamma is to have a portfolio of supported model checkers. Each tool implements different algorithms tailored to specific classes of problems, therefore having a larger variety of model checkers increases the chance of successful verification. For example, UPPAAL uses explicit model checking and is efficient for timed systems, while Theta uses a wide array of abstraction-based symbolic techniques that have a larger overhead on simpler problems, but have a chance of verifying harder ones where an explicit algorithm would not scale.

In order to have a scalable architecture, we moved the processing from workstations to the cloud. Most components in the architecture can be deployed on Kubernetes, an open-source cloud orchestration, deployment and scalability tool for containerized applications. Applications can be deployed into different execution units, called pods, that can have certain amount of computation resources allocated. In this way, they can be scaled up with high amount of CPU and memory (vertical scalability) or new pods can be started on-demand (horizontal scalability), as long as the cluster has enough computational resources. The elastic scalability of the cloud enables the adaptive allocation of a high amount of computational resources [59], that can address the high resource demand of the *Model Checker Runtime (MCR)* component. Moreover, the portfolio of model checkers philosophy of Gamma can also benefit from the cloud-based setting, because we can execute multiple model checker configurations in parallel and stop the process when one of them gives a result. Ultimately, if we accumulate enough data about the target models, we may be able to reduce the executed configurations to a few promising ones and achieve a relatively good price/performance ratio.

As model checking is the most resource intensive task in the workflow, we serve each verification task in a separate pod, running the *MCR*. This way, the long-running tasks can be served in parallel. As the concurrent pods (managed by Kubernetes) can be deployed to a computing cluster, the overall performance is no longer limited by the resources available on a single computer. Therefore, the workflow can serve many users running their verification tasks concurrently and enable its use as a common verification service for a team of systems engineers.

Besides scalability, another advantage of our approach is the separation of concerns for the engineering and the formal verification domains: systems engineers design both the models and the properties in a high-level engineering language they are familiar with. The formal models are automatically derived from these high-level design models by a series of transformations hidden from the users. The verification result is back-annotated to the original engineering language, thereby making it easier to understand potential issues without expertise in formal methods. Besides, it helps engineers to inspect the execution in detail or to derive further test cases.

4.3 | Demonstration of semantic integrity

The semantic integrity of the toolchain can be illustrated by checking Property (b) on the motivating example (Figure 3). The verification result trace (Figure 7) proves that the property is violated, because in the last state `battery` is 79%, the Spacecraft is still transmitting and the Ground Station has already received 20 `packets`. Simulating the sequence diagram trace in CST, one can find that the entry action of `Discharging` state causes the issue (Figure 2c). The faulty model can be fixed by moving the action to the effect of the outgoing timed transition. Rerunning verification proves that the undesired state is not reachable anymore.

From the systems engineering perspective, the checked properties are representatives of requirements, that would have been difficult to check with inspection or traditional simulation of operational scenarios. The warnings during the transformation emphasize that only part of the possible behavior is explored. But if an execution trace is returned, then it can help engineers inspect the actual behavior of their model. Note that we will revisit the limitations of such trace back-annotation in Section 6.

5 | EVALUATION

We evaluated our proposed validation and verification workflow in different scenarios imitating real-world use. These scenarios demonstrate that

1. our approach can scale to industrially relevant models,
2. the cloud-based workflow can efficiently server multiple users running different verification jobs.

Environment

We set up an evaluation environment on Amazon Elastic Kubernetes Service (EKS). The infrastructure consisted of two EC2 nodes: one m5.xlarge instance (4 vCPUs, 16 GB RAM) and one m5.2xlarge instance (8 vCPUs, 32 GB RAM). The elements of the architecture in Figure 8 were deployed as follows: *IncQuery Suite (IQS)* and the *Validation and Transformation Service (VTS)* were running on the m5.xlarge instance; the *Model Checker Runtime (MCR)* was deployed in varying number of pods on the m5.2xlarge instance, depending on the scenario.

Models

For benchmark purposes, we used the running example *Spacecraft model* from this paper with two versions and a modified version of OpenMBEE's Thirty-Meter Telescope (TMT) [11] model. In the *faulty* version of the Spacecraft model, the reachability property (Figure 3) is satisfied; in the *fixed* version, the reachability property is unsatisfied due to the correction described in Section 4.3. Both the *faulty* and the *fixed* versions of the model contain 11 states, 23 transitions and 5 activity actions.

In the *TMT model*, we chose the Procedure Executive and Analysis Software (PEAS) Block and adapted its state machine and activities to the pragmatic subset of SysML supported by our workflow. We removed do-behaviors from the state machine, removed unsupported Actions from the Activities, replaced Float variables with Integers, and resolved data type inconsistencies in Activity actions. This was necessary to make the model conform with our pragmatic subset. Nevertheless, the resulting model is still quite complex and represents the general modeling patterns used in the OpenSE Cookbook. We specified four reachability properties for the TMT model, denoted as *TMT1–4* in Table 2. The table contains the name of the target state and the number of states and activity actions for the shortest paths satisfying the respective property. The PEAS block contains 61 states, 93 transitions and 2310 activity actions althogether.

5.1 | Evaluation scenarios

We describe the two scenarios in which the workflow was executed and the results obtained in each of them.

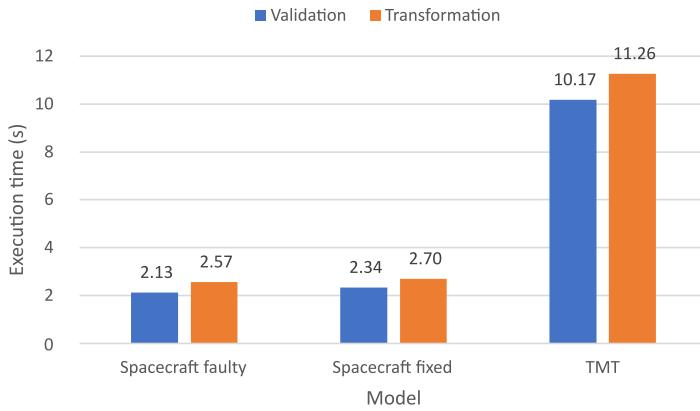
Name	Target state	Number of states	Number of activity actions
TMT1	Minimize Sensor Readings	5	213
TMT2	M3 Alignment 3	8	533
TMT3	Broad Band Phasing 3um	9	636
TMT4	Broad Band Phasing 1um 3	10	739

TABLE 2 Reachability properties in TMT and metrics on the shortest path to reach the target state.

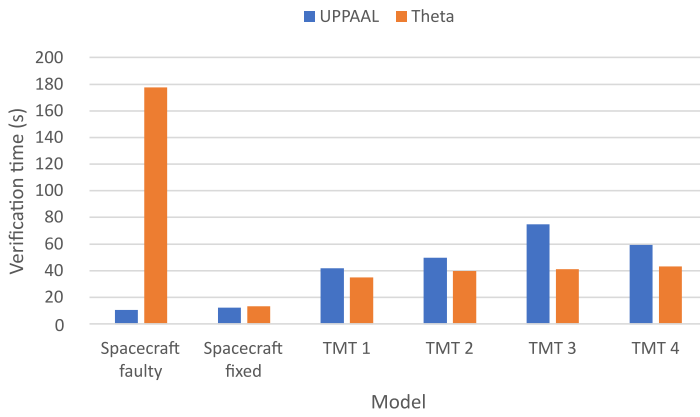
5.1.1 | Scenario 1: Artificial and real-world models

Description

We deployed the *MCR* in one pod and we only sent one validation, transformation and verification request at a time for each model and reachability property. This scenario serves as a high-level validation and as a baseline.



(a) Validation and transformation times.



(b) Verification times.

FIGURE 9 Validation, transformation and verification times of the Spacecraft and TMT models.

Results

Figure 9 depicts the measurement results. Figure 9a shows that the model validation and transformation were completed in an acceptable time, despite running more than 130 validation rules. The model sizes are reflected in the execution times: while the simpler Spacecraft model is validated in 2 seconds and transformed in 2.5 seconds, the more complex TMT model is validated in 10 seconds and transformed in 11 seconds.

Regarding the verification times, depicted in Figure 9b, it can be seen on the one hand, that UPPAAL outperforms Theta for the *Spacecraft faulty* model which contains timed transitions. On the other hand, Theta performs better for the TMT model which contains many data variables, due to the large number of activity actions whose instructions are transformed to variable assignments. The large difference in verification times of the *Spacecraft faulty* model is due to the abstraction-refinement algorithm implemented by Theta that tracks the values of the clock variables in the model, which results in a large number of combinations that needs to be checked by the model checker. However, in case of the *Spacecraft fixed* model, where the property is unsatisfied, Theta finishes in about the same time as UPPAAL, due to the abstraction domain used in verification.

Conclusions: As Figure 9 shows, the approach is capable of transforming and verifying properties on artificial and complex industrial models within an acceptable time. Besides, the differences between explicit and abstraction-based model checkers could be also observed: UPPAAL may handle timed systems better than Theta, however further models and verifiable properties are needed to draw a definitive conclusion in this respect. As Figure 9 shows, the verification times are the most influential in the whole validation and verification workflow, therefore in the subsequent scenario we only measure the verification time.

5.1.2 | Scenario 2: Horizontal scalability

Description

This scenario investigates that if we have multiple verification requests (possibly coming from different users) then how does the approach scale if more than one *MCR*s are performing verification. Therefore we deployed the *MCR* on 1, 2, 4 pods. Each pod was handling only one verification request at a time. We sent 4 verification requests (*TMT1–4*) in a fixed order directly after each other asynchronously to the workflow. The verification requests were waiting in a queue until being processed by a free pod. We repeated the measurements with each model checker (UPPAAL and Theta) separately, resulting in six sub-scenarios (3 pod configurations $\times$ 2 model checkers).

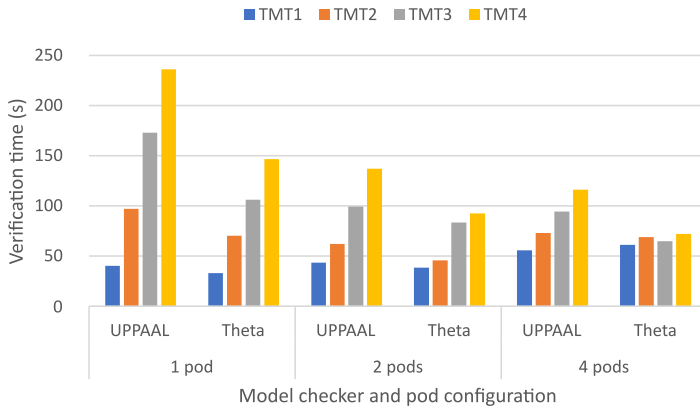


FIGURE 10 TMT model verification times when using the elastic scalability of the workflow.

Results

By looking at Figure 10, we can see the execution order and the effect of the queuing times of the requests. As the number of processing pods grow from 1 to 2, so is the overall execution time of the requests reduced. However, in case of 4 pods we can see that the concurrent verification requests give higher load to the single processing m5.2xlarge node. This results in longer verification times than in Figure 9b when only one request was processed by the node at the same time. Besides, we can also observe that Theta was able to verify the models faster than UPPAAL, confirming our observations from *Scenario 1*.

Conclusions: The benefit of using a cloud-based approach in the deployment is that, as the number of pods grow, we can serve more users concurrently with the same resources. This approach can result in lower round-trip times for long-running verification tasks and better resource utilization in multi-user environments. However, we shall also consider the computational capacity of the processing nodes when allocating pods to the nodes, to avoid starvation.

5.2 | Discussion

Summary of results

The results of the evaluation show that using the recommended SysML subset and the developed V&V workflow checking reachability properties on large, executable models with hundreds of detailed actions is feasible. Due to the reasonable V&V times, the workflow can provide feedback to engineers on potential design issues. The cloud-based, scalable architecture makes it possible to use the workflow in a multi-user, enterprise environment.

Threats to validity

To address threats to the *internal validity* of the measurements, we executed 20 rounds of warm-up requests in each phase, followed by the 7 rounds of measurements whose median represents a data point on each figure. In order to address the *external validity* concerns, we used an industrial model (TMT) apart from a syntactic model (running example). However, our results might not be generalizable for models built from elements that are not part of the pragmatic subset defined in Section 3.3 (*construct validity*). Nevertheless, it is important to note that industrial models are generally confidential, so having an evaluation as presented in this paper is vital to move forward to industrial use.

6 | LESSONS LEARNED

We present the lessons learned during designing a method capable of verifying system models in an industrial setting. We discuss the requirements and challenges related to (1) supporting SysML models in practice, (2) industrial deployment and (3) scaling model checking.

6.1 | Handling and interpreting industrial models

Minimum supportable elements

During the interviews with systems engineers we have learned that pure SysML state machines alone are not enough to tackle practical engineering problems. Script-based action and guard languages help engineers express statements and conditions with a compact textual syntax. Based on the TMT model and discussions with systems engineers from NASA JPL, activity diagram is a commonly used formalism to express behavior in an engineering-friendly way. The data and control flow-based semantics of activity diagrams is usually easier for systems engineers to understand than

the state-based semantics, due to its resemblance to programming.

Lesson 1. Supporting some executable action and guard language, and activity diagrams for specifying detailed actions is a must to verify industrial models.

Interpretation of semantics

Although the precise definition of the semantics of UML/SysML models has vastly improved with the introduction of the fUML and PSSM specifications, there are still ambiguous interpretations in the execution semantics of activity and state machine diagrams [50, 16]. The questions related to semantics are further complicated by two aspects in practice. First, tool vendors developing a modeling tool for a longer time may implement some parts based on previous specifications or they need to fill the gaps in the specification. This can lead to situations like that the simulation tool of MagicDraw (CST) executes the models according to a semantics (SCXML[9]) that is independent from the OMG-specified standards [56, 58]. Second, even if some parts of the specification is precise, systems engineers might interpret a diagram differently. We encountered several occasions when based on previous experiences with other tools or statechart notations, engineers did not realize all the possible interleavings due to non-determinism in a complex statechart with orthogonal regions. Due to these reasons the model checker may return several traces that cannot be reproduced by the corresponding tools or contradict the intuitions of systems engineers.

Lesson 2. Finding a pragmatic subset of the modeling language can help with reducing the ambiguity in interpretation among the tools and humans.

A pragmatic subset of SysML

On the one hand, the pragmatic subset of SysML chosen in Section 3.3 can be regarded as an opinionated modeling approach that puts some restrictions on the modeling practice of the engineers. On the other hand, these restrictions are necessary to find an appropriate balance between verifiable models and the large degree of freedom engineers like to have when designing systems. From the engineering point of view, the pragmatic subset enables the design of composite asynchronous reactive systems that communicate with signals and have actions defined with textual JavaScript syntax or graphical activities. From the formal verification perspective, the pragmatic subset represents the most common modeling elements that state-of-the-art model checkers are able to verify.

In this paper, we have experimented with the tools mentioned in Section 3. The semantic integrity of the workflow is specific for the tools involved in it. Therefore, further analysis is needed to test, if some tools could be replaced by others, e.g., using a tool other than MagicDraw and Cameo Systems Modeler for system design.

Back-propagation of results

Running a successful formal verification on a complex state machine is only the first part of the challenge. In case of a long counterexample, presenting the low-level trace of the model checkers usually does not add much business value as systems engineers need a representation that can help them to locate and correct the root cause of the problem. Therefore the verification results must be mapped back to the domain of the system model. This is a non-trivial task known in the literature. For example, Hegedüs et al. [30] have listed several challenges that could occur when back-propagating a formal execution trace to a domain execution sequence: mismatch between step granularity, independent subsequences, and spurious formal sequences. We encountered similar challenges that were later tackled by the Gamma Framework during the back-propagation of the low-level formal traces to the Execution

[9] State Chart XML (SCXML): `https://www.w3.org/TR/scxml/`

Trace in Gamma. After the Gamma Execution Trace to SysML Sequence Diagram transformation, we faced the issue that the resulting Interaction was not completely compatible with Cameo Simulation Toolkit, as the simulator did not wait the exact amount of time as set in the Duration Constraints. Besides, the simulation of the Interaction sporadically became unsynchronized with the state machines, i.e., the Interaction did not proceed from a State Invariant, even though the respective state was active. These issues hindered the correct simulatability of the traces, which can be rather used for manual inspection purposes. Full simulation was not a priority feature at the moment, but a future challenge is to be able to simulate all special corner cases returned by the model checker.

Lesson 3. Back-propagating and simulating complex counterexamples involving multiple state machines with exact timing information is an open challenge not supported by all current tools.

6.2 | Infrastructure for an industrial environment

Requirements

Having a verification solution alone is unfortunately not enough to be deployed to an enterprise environment. The tool should (1) be deployable both on-premise and to cloud infrastructure, (2) should be able to integrate to other systems. Besides, for confidentiality reasons it should have (3) authentication and authorization with auditing, (4) should be able to serve jobs from multiple users in parallel with an acceptable response time. For auditing purposes, the system (5) should log the executed operations without degrading its performance, and (6) should make the transformation artefacts and logs easily retrievable. Fulfilling these requirements needs extra effort, but it enables applying state-of-the-art research in practice.

Lesson 4. The verification workflow shall be adapted to the policies and requirements of the corresponding enterprise, and contain many features that are usually not supported by research tools.

Realization

To be able to deploy the V&V workflow to the industrial context presented in the paper, one of the objectives set out in Section 2.3 was to have a toolchain that can be integrated with OpenMBEE. This goal was achieved in the following way. The workflow uses MMS as model repository, where the SysML models can be uploaded by using the MDK plug-in of MagicDraw and can be reviewed by the View Editor component (Figure 1). Although we used MMS in our case study, but our solution is model repository agnostic, because it delegates the model acquisition task to IncQuery Suite (IQS). Therefore, further repositories (e.g., Teamwork Cloud) can be added to the workflow, as long as they are supported by IQS. Moreover, all the features of the workflow are accessible through REST APIs, which makes it easy to integrate them into various engineering workflows with different tools. A major effort was put into developing a container-based wrapper around the Gamma Framework and model checker tools that can be easily deployed (i.e., Model Checker Runtime), and a job management feature that orchestrates the requests and the steps in the V&V workflow.

6.3 | Scaling model checking

State space explosion is one of the most difficult problems in model checking, which makes it difficult to scale for industry-grade models. In order to proceed with tackling this challenge, we took the benefit of several optimizations in the Gamma Framework. On the level of the state machines, such optimizations include removing statically unreach-

able states and unfireable transitions. On the level of the formal models, these include merging consecutive, atomic transition sequences to one transition, using scoped local variables in actions, inlining statically deducible variable assignments, removing unused variables. Besides, to reduce concurrency in the models, we restricted the execution order of orthogonal regions in state machines, and control flows between fork-join nodes in activities. These restrictions reduce the state space that has to be explored and therefore provide verification results faster.

Proving properties with abstraction-based model checking techniques may cause unexpectedly long execution times, if the wrong abstraction refinement is chosen by the model checker. To avoid this outcome, when the Theta is chosen as a model checker for a verification task, we run multiple instances of it with different abstraction parametrizations (forming a model checking portfolio). As soon as the quickest instance returns a result, the others are terminated. Nevertheless, it may occur that all instances run indefinitely, in which case formal engineering expert knowledge is needed to fine-tune the parametrizations according to the models and the properties being verified. However, this adaptive fine-tuning is left as future work [1].

Nevertheless, by applying the approach proposed in the paper, we have learned that applying model checking on industry-size models remains challenging despite the optimizations mentioned before. On the one hand, in model checking, improving abstraction-based verification algorithms [61] can help with gaining some runtime performance. On the other hand, the cloud-ready architecture enables the use of elastic cloud resources which give more computation power even if this improvement is less significant compared to an improvement on the algorithm level. Finally, further evaluations are needed to find the scalability limits of the approach with respect to the model size.

Lesson 5. To proceed towards scaling verification for industrial behavioral models, restricting the explored state space, according to the environmental constraints in which the models will be used, is a necessary optimization choice.

7 | RELATED WORK

To put our research in a broader context, we collected related work about the practical advantages and perceived challenges of using *model-driven engineering and formal methods in industry* (Section 7.1), and applications of *hidden formal methods* in verifying system models (Section 7.2).

7.1 | MDE and formal methods in industry

Bucchiarone et al. [6] described tool and implementation challenges hindering the wide-spread use of Model-Driven Engineering (MDE). Among others, lack of good tooling, e.g., component-based integrated environments, that adopt textual languages and treat them like models or graphical interfaces tailored for the engineers' needs. From the implementation perspective, traceability between the different artifacts [2] and understanding the semantics of such links are also important. Finally, scalability in terms of size and diversity of artifacts, i.e., models, metamodels, model transformations and dependencies in any non-trivial project, have been denoted as one of the open challenges that tools tried to address in the last decade. We encountered similar challenges and therefore focused on an integrated, scalable tooling environment.

Hutchinson et al. [36] conducted a survey with 449 participants and complemented it with in-depth interviews with practitioners from four companies to learn about the success and failure factors of adopting MDE practices in industry. They concluded that the successful adoption of MDE in industry requires a progressive and iterative approach with a transparent organizational commitment, clear business focus and willingness to align internal processes accordingly.

The social implications of this change management, e.g., increased training costs, willingness to change the engineering methodology, should be also considered besides the technical factors. *Huldt and Stenius* [35] identified similar factors in a survey with 66 professionals. Our experiences confirm that an iterative approach is key to success, and the cost and challenge of changing modeling methods and practices should be taken into account from start.

Gleirscher and Marmsoler [25] surveyed 216 participants from industry (78%) and academia (22%) about the use of formal methods (FM) in mission-critical software domains. Their results indicate an increased intent to apply FMs in industry across all application domains, suggesting a positively perceived usefulness. Besides, the intrinsic motivation to use FM is stronger than the regulatory one. Scalability, skills, and education were perceived as the toughest challenges of applying FMs in practice. More experienced respondents more often rated these challenges as highly difficult, compared to less experienced ones. Finally, past experience with formal methods was positively correlated with future usage intent. We observed a similar situation in NASA JPL, where successful previous projects [23, 24] opened the way for working an environment to use formal verification as a service.

Garavel et al. [21] surveyed 130 high-profile experts in different aspects of formal methods for the 25th international conference on Formal Methods for Industrial Critical Systems. One aspect was the industrial adoption of formal method practices. 67.7% of the responders believe that formal methods are now ready to be used in industry in a limited extent. The reasons for the limited applicability are often related to the domains, tool maturity, and people's skill and willingness to transfer and adopt academic research results to industrial case studies. According to the survey, the most mentioned limiting factor of wider adoption of formal method by industry are the improper integration of formal methods in the industrial design life-cycle, the lack of proper training of FMs and its steep learning curve. According to the research participants, more collaborative projects between research and industry and increased support for academic researchers developing tools can contribute to addressing these challenges. We contributed to overcoming these limitations by developing a workflow that is tightly integrated into the engineering environments.

Woodcock et al. [66] surveyed 62 industrial projects to collect experiences about industrial adoption of formal methods. On correlating the techniques used against the project date, they found an increase of model checking from 13% in the 1990s to 51% in 2009. According to the survey, improved quality is one of the main benefits of applying formal methods, 92% of all cases reporting an increase in quality compared to other techniques. Detection of faults, improvements in design, increased confidence and improved understanding were the most mentioned specificities of quality improvement. On the other hand, education of engineers, and the integration of model-checking solutions into the toolset of engineers, and quick response time ("formal methods need to provide answers in seconds or minutes rather than days") are the major impediments to formal methods adoption. Our objectives of developing a pragmatic verification approach were motivated by the same limitations.

7.2 | Verifying systems with hidden formal methods

To motivate the use of hidden formal methods, *Visser et al.* [64] surveyed papers which applied such methods to verify complex domain-specific hardware and software models. They observed that keeping the formal and intermediate models hidden from the end-user is paramount to the success of the domain-specific model checker. Besides, they argued that model checking becomes effective when the natural notations of the design domain are supported in the formal models as well. However, if there is a semantic gap between the two domains, then similar to our approach, they advocate to develop appropriate intermediate representations (models) to bridge the gap both in the forward transformation and the back-annotation phases.

Software and systems model checking is a widely researched area, with a plethora of tools and approaches available. *Ciccozzi et al.* [9] performed a systematic review of solutions for the execution of UML models. The closest one

to ours is *Kölbl et al.* [44] who proposed an approach to translate SysML models to the language of NuSMV, Prism, and Spin model checkers. Similar to us, they used an intermediate metamodel between the engineering and formal domains, and the counterexamples are returned as SysML sequence diagrams. In contrast to us, they verified Linear Temporal Logic (LTL) expressions specified as OCL state invariants in SysML, supported only send signal actions in activities, and used only a small model of an airbag system.

Calvino and Apvrille [7] proposed the direct model-checking of SysML state machine models. In their paper, they used AVATAR to design the SysML models that are directly verified by TTool. They support the verification of a broader set of formal expressions specified in text. The verification results of reachability and liveness properties are back-annotated directly to the state machines, but a trace of the model checker representation is also returned.

Gibson et al. [23] verified properties on SysML statecharts by combining code generation with software model checking techniques. They translated the state machines to Java code by the COMODO model-to-text transformation tool, and evaluated certain properties by Java Pathfinder [28]. The verifiable property was directly inserted in the generated code, the guards of the transitions were transformed manually, and the result was not annotated back to the original model. They used depth-limits as a trade-off between performance and the ability to verify properties [24].

de la Croix et al. [14] proposed a role-based framework for the design, execution and verification of robotic applications. The framework extends the Business Process Modeling Language and Notation to design missions. To check the models' correctness, they translate a subset of the elements to Spin [31] and verify simple LTL properties. The back-annotation of results is a proposed future work.

Miller et al. [51] proposed a translator framework that allows engineers to automatically translate synchronous dataflow models and state machines from MATLAB Simulink and Esterel Technologies SCADE Suite to a variety of model checkers and theorem provers (NuSMV, SAL, Prover, PVL, ACL2). As an intermediate language they use the Lustre formal specification language. Similar to Gamma, the translator framework applies optimizations to the models to reduce their formal verification time. The counterexamples produced by the model checkers are translated to a simple spreadsheet showing the inputs and outputs of the model for each step.Finally, they demonstrated the application of the framework in three industrial case studies.

Cicchetti et al. [8] proposed the CHESS framework to design, validate and verify and generate code from complex component-based embedded software system models. For modeling, they use CHESSML that adopts elements from the UML, SysML and MARTE languages. In the validation and verification phase, engineers can perform dependability and schedulability analysis (e.g., Failure Mode and Effect Analysis, Fault Tree Analysis, etc. [8]), contract-based analysis and formal verification [15]. In formal verification, the system and component-level properties are formalized into LTL expressions that are verified by the nuXmv model checker. The results of the analysis and the verification are back-propagated to the design model, following the principles of hidden formal methods.

To the best of our knowledge, all of these tools are purpose-built, single user tools targeted towards experts without APIs. The main differentiating feature of our approach is a workflow integrated into common engineering design tools hiding formal methods that can be deployed into a multi-user, cloud-based enterprise environment.

8 | CONCLUSION

In this paper, we presented an automated verification and validation workflow of industrial executable SysML models. The workflow is integrated to the OpenMBEE project and maintains a pragmatic semantic integrity among the tools used in the design, validation and verification phases. We identified a pragmatic subset of the SysML language supported by all tools that allows for complex practical modeling, while also supports scalable, but limited state-space

verification. The workflow enables the verification of reachability properties on state machines and activity diagrams, using the Gamma Framework and different model checkers. The process is hidden, fully automated and offloaded to cloud services. The results are back-propagated to the original domain, to help engineers understand them in a language they are familiar with.

The most important lessons learned in our research were: (1) finding a pragmatic subset of the modeling language can help with reducing the ambiguity in interpretation among the tools and humans, (2) supporting some executable action and guard language is a must to verify industrial models, (3) restricting the explored state space is a necessary optimization choice to proceed towards scaling verification for industrial behavioral models.

As future work, we are planning to adapt our approach to the next generation of SysML (SysML v2), which completely changes the way semantics are captured in the standard [19, 53], and has the potential of widening the semantic integrity for a larger set of next-generation tools. Secondly, we are planning to deploy the workflow to an industrial partner to improve the work of systems engineers and collect feedback from them. Finally, we will fine-tune the portfolio-based execution to predict which model checker parametrization provides the fastest results.

Acknowledgment

This research was carried out at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under a contract with the National Aeronautics and Space Administration (NASA). The authors would like to thank Luigi Andolfato (ESO), as well as Myra Lattimore and Ivan Gomes (NASA JPL) for their suggestions to improve the paper. We would also like to thank Milán Mondok (BME) and Ábel Hegedüs, Ármin Zavada and Balázs Várady (IncQuery Labs) for their help with implementing the workflow. This work partially received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813884, and the NRDI Fund of Hungary, financed under the [2019-2.1.1-EUREKA-2019-00001] funding scheme.

Disclaimer. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

References

[1] Zsófia Ádám, Levente Bajczi, Mihály Dobos-Kovács, Ákos Hajdu, and Vince Molnár. Theta: Portfolio of CEGAR-based analyses with dynamic algorithm selection (competition contribution). In *Proc. TACAS (2)*. Springer, 2022.

[2] Deniz Akdur, Vahid Garousi, and Onur Demirörs. A survey on modeling and model-driven engineering practices in the embedded software industry. *J. Syst. Archit.*, 91:62–82, 2018.

[3] L Andolfato, G Chiozzi, N Migliorini, and C Morales. A platform independent framework for statecharts code generation. In *Proc. of the 13th Int. Conf. on Accelerator and Large Experimental Physics Control Systems*, 2011.

[4] Ronan Baduel, Mohammad Chami, Jean-Michel Bruel, and Iulian Ober. SysML Models Verification and Validation in an Industrial Context: Challenges and Experimentation. In *Modelling Foundations and Applications*, pages 132–146. Springer International Publishing, 2018.

[5] Gerd Behrmann, Alexandre David, Kim G. Larsen, John Hakansson, Paul Petterson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *Proc. of the 3rd Int. Conf. on the Quantitative Evaluation of Systems*, page 125–126. IEEE, 2006.

[6] Antonio Bucchiarone, Jordi Cabot, Richard F. Paige, and Alfonso Pierantonio. Grand challenges in model-driven engineering: an analysis of the state of the research. *Softw. Syst. Model.*, 19(1):5–13, 2020.

[7] Alessandro Tempia Calvino and Ludovic Apvrille. Direct model-checking of SysML models. In *Proc. of the 9th Int. Conf. on Model-Driven Engineering and Software Development*, pages 216–223. SCITEPRESS, 2021.

[8] 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.

[9] Federico Ciccozzi, Ivano Malavolta, and Bran Selic. Execution of UML models: a systematic review of research and practice. *Software & Systems Modeling*, 18(3):2313–2360, 2018.

[10] Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick P Bloem. *Handbook of model checking*. Springer, 2018.

[11] TMT Observatory Corporation. Thirty Meter Telescope SysML model, 2022. Last accessed on 2022-09-09.

[12] Michelle L. Crane and Juergen Dingel. On the semantics of UML state machines: Categorization and comparision. In *In Technical Report 2005-501, School of Computing, Queen's*, 2005.

[13] Michelle L. Crane and Jürgen Dingel. UML vs. classical vs. Rhapsody statecharts: not all models are created equal. *Softw. Syst. Model.*, 6(4):415–435, 2007.

[14] Jean-Pierre de la Croix, Grace Lim, Joshua Vander Hook, Amir Rahmani, Greg Droge, Alexander Xydes, and Chris Scrapper Jr. Mission modeling, planning, and execution module for teams of unmanned vehicles. In *Unmanned Systems Technology XIX*, volume 10195, page 101950J. International Society for Optics and Photonics, 2017.

[15] Alberto Debiasi, Felicien Ihirwe, Pierluigi Pierini, Silvia Mazzini, and Stefano Tonetta. Model-based analysis support for dependable complex systems in CHESS. In *Proc. of the 9th Int. Conf. on Model-Driven Engineering and Software Development*, pages 262–269. SCITEPRESS, 2021.

[16] Márton Elekes and Zoltán Micskei. Towards testing the UML PSSM test suite. In *Proc. of the 10th Latin-American Symposium on Dependable Computing*, pages 1–4. IEEE, 2021.

[17] Márton Elekes, Vince Molnár, and Zoltán Micskei. Assessing the specification of modelling language semantics: A study on UML PSSM, 2022.

[18] Rik Eshuis. Reconciling statechart semantics. *Sci. Comput. Program.*, 74(3):65–99, 2009.

[19] Sanford Friedenthal. Requirements for the Next Generation Systems Modeling Language (SysML v2). *INSIGHT*, 21(1):21–25, 2018.

[20] Jonas Fritzsch, Tobias Schmid, and Stefan Wagner. Experiences from large-scale model checking: Verification of a vehicle control system. *CoRR*, abs/2011.10351, 2020.

[21] Hubert Garavel, Maurice H. ter Beek, and Jaco van de Pol. The 2020 expert survey on formal methods. In *Formal Methods for Industrial Critical Systems*, volume 12327 of *LNCS*, pages 3–69. Springer, 2020.

[22] Corrina Gibson, Michael Bonnici, and Jean-Francois Castet. Model-based spacecraft fault management design & formal validation. In *2015 IEEE Aerospace Conference*, pages 1–12, 2015.

[23] Corrina Gibson, Robert Karban, Luigi Andolfato, and John Day. Formal validation of fault management design solutions. *Softw. Eng. Notes*, 39(1):1–5, 2014.

[24] Corrina Gibson, Robert Karban, Luigi Andolfato, and John C. Day. Abstractions for executable and checkable fault management models. In *Systems Engineering Research*, pages 146–154. Elsevier, 2014.

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

[26] Bence Graics, Vince Molnár, and István Majzik. Integration test generation for state-based components in the Gamma framework. *Preprint*, 2022.

[27] Joshua Logan Grumbach and Lawrence Dale Thomas. Systems integration implications of component reuse. *Systems Engineering*, 2022.

[28] Klaus Havelund and Thomas Pressburger. Model checking JAVA programs using JAVA finder. *Int. J. Softw. Tools Technol. Transf.*, 2(4):366–381, 2000.

[29] Ábel Hegedüs, Gábor Bergmann, Csaba Debreceni, Ákos Horváth, Péter Lunk, Ákos Menyhért, István Papp, Dániel Varró, Tomas Vileiniskis, and István Ráth. IncQuery Server for Teamwork Cloud: Scalable query evaluation over collaborative model repositories. In *Proc. of the 21st Int. Conf. on Model Driven Engineering Languages and Systems*, page 27–31. ACM, 2018.

[30] Ábel Hegedüs, Gábor Bergmann, István Ráth, and Dániel Varró. Back-annotation of simulation traces with change-driven model transformations. In *Proc of the 8th Int. Conf. on Software Engineering and Formal Methods*, pages 145–155. IEEE Computer Society, 2010.

[31] Gerard Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 1st edition, 2011.

[32] Gerard J Holzmann. Mars code. *Comm. of the ACM*, 57(2):64–73, 2014.

[33] Benedek Horváth, Bence Graics, Ákos Hajdu, Zoltán Micskei, Vince Molnár, István Ráth, Luigi Andolfato, Ivan Gomes, and Robert Karban. Model checking as a service: towards pragmatic hidden formal methods. In *Companion Proc. of the 23rd Int. Conf. on Model Driven Engineering Languages and Systems*, pages 37:1–37:5. ACM, 2020.

[34] Zhaoxia Hu and Sol M. Shatz. Explicit modeling of semantics associated with composite states in UML statecharts. *Autom. Softw. Eng.*, 13(4):423–467, 2006.

[35] T. Huldt and I. Stenius. State-of-practice survey of model-based systems engineering. *Systems Engineering*, 22(2):134–145, 2019.

[36] John Edward Hutchinson, Jon Whittle, and Mark Rouncefield. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Sci. Comput. Program.*, 89:144–161, 2014.

[37] Nerijus Jankevicius. Resource analysis and automated verification for the thirty meter telescope using executable SysML models. In *Executable Modeling*, volume 1760, pages 2–4. CEUR-WS.org, 2016.

[38] P. A. Jansma and R. M. Jones. Advancing the practice of systems engineering at JPL. In *IEEE Aerospace Conference*, page 19 pp., March 2006.

[39] Robert Karban, Amanda G Crawford, Gelys Trancho, Michele Zamparelli, Sebastian Herzig, Ivan Gomes, Marie Piette, and Eric Brower. The OpenSE cookbook. In *Modeling, Systems Engineering, and Project Management for Astronomy VIII*, volume 10705, page 107050W, 2018.

[40] Robert Karban, Frank G. Dekens, Sebastian Herzig, Maged Elaasar, and Nerijus Jankevičius. Creating system engineering products with executable models in a model-based engineering environment. In *Modeling, Systems Engineering, and Project Management for Astronomy VII*, volume 9911, pages 96–111. SPIE, 2016.

[41] Robert Karban, Nerijus Jankevičius, and Maged Elaasar. ESEM: Automated systems analysis using executable SysML modeling patterns. In *INCOSE International Symposium*, volume 26, pages 1–24. Wiley, 2016.

[42] Robert Karban, Marie Piette, Eric Brower, Ivan Gomes, Emilee Bovre, Myra Lattimore, Blake Regalia, John Carr, Chad Harris, Christopher Delp, and Cin-Young Lee, 2020. Last accessed on 2022-09-09.

[43] Robert Karban, Robbie Robertson, Ahsan Qamar, and Erich Lee. Preface to the OpenMBEE int. workshop. In *Companion Proc. of the 23rd Int. Conf. on Model Driven Engineering Languages and Systems*, pages 464–464. IEEE, 2021.

[44] 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 *LNCS*, pages 255–274. Springer, 2018.

[45] Diego Latella, István Majzik, and Mieke Massink. Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker. *Formal Aspects Comput.*, 11(6):637–664, 1999.

[46] Diego Latella, Istvan Majzik, and Mieke Massink. Towards a formal operational semantics of UML statechart diagrams. In *Formal Methods for Open Object-Based Distributed Systems*, pages 331–347. Springer, 1999.

[47] Lucas Lima, André Didier, and Márcio Cornélio. A formal semantics for SysML activity diagrams. In *Formal Methods: Foundations and Applications*, volume 8195 of *LNCS*, pages 179–194. Springer, 2013.

[48] Shuang Liu, Yang Liu, Étienne André, Christine Choppy, Jun Sun, Bimlesh Wadhwa, and Jin Song Dong. A formal semantics for complete UML state machines with communications. In *Proc. of the 10th Int. Conf. on Integrated Formal Methods*, volume 7940 of *LNCS*, pages 331–346. Springer, 2013.

[49] Azad M. Madni and Michael Sievers. Model-based systems engineering: Motivation, current status, and research opportunities. *Systems Engineering*, 21(3):172–190, 2018.

[50] Zoltán Micskei, Raimund-Andreas Konnerth, Benedek Horváth, Oszkár Semerath, András Vörös, and Dániel Varró. On open source tools for behavioral modeling and analysis with fUML and Alf. In *1st Workshop on Open Source Software for Model Driven Engineering*, volume 1290, pages 31–41. CEUR, 2014.

[51] Steven P. Miller, Michael W. Whalen, and Darren D. Cofer. Software model checking takes off. *Commun. ACM*, 53(2):58–64, 2010.

[52] 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 *Proc. of the 40th Int. Conf. on Software Engineering*, pages 113–116. ACM, 2018.

[53] OMG. *Systems Modeling Language (SysML) v2 RFP*, 2017. ad/17-12-02.

[54] OMG. *Unified Modeling Language (UML)*, 2017. formal/17-12-05.

[55] OMG. *Precise Semantics of UML Composite Structures (PSCS)*, 2019. formal/19-02-01.

[56] OMG. *Precise Semantics of UML State Machines (PSSM)*, 2019. formal/19-05-01.

[57] OMG. *System Modeling Language (SysML)*, 2019. formal/19-11-01.

[58] OMG. *Semantics of a Foundational Subset for Executable UML Models (fUML)*, 2021. formal/21-03-01.

[59] Amir Molzam Sharifloo and Andreas Metzger. Mcaas: Model checking in the cloud for assurances of adaptive systems. In *Software Engineering for Self-Adaptive Systems III*, pages 137–153. Springer, 2013.

[60] Walter McGee Taraila and Sharanabasaweshwara Asundi. Model-based systems engineering for a small-lift launch facility. *Systems Engineering*, 2022.

[61] Tamás Tóth. *Abstraction refinement-based verification of timed automata*. PhD thesis, Budapest University of Technology and Economics, 2021.

[62] Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. Theta: a framework for abstraction refinement-based model checking. In *Formal Methods in Computer-Aided Design*, pages 176–179, 2017.

[63] Dániel Varró, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, István Ráth, and Zoltán Ujhelyi. Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. *Softw. Syst. Model.*, 15(3):609–629, 2016.

[64] Willem Visser, Matthew B. Dwyer, and Michael W. Whalen. The hidden models of model checking. *Softw. Syst. Model.*, 11(4):541–555, 2012.

[65] Sabine Wolny, Alexandra Mazak, Christine Carpella, Verena Geist, and Manuel Wimmer. Thirteen years of SysML: a systematic mapping study. *Softw. Syst. Model.*, 19(1):111–169, 2019.

[66] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John S. Fitzgerald. Formal methods: Practice and experience. *ACM Comput. Surv.*, 41(4):19:1–19:36, 2009.

[67] Paulo Younse, Jessica Cameron, and Thomas H. Bradley. Comparative analysis of model-based and traditional systems engineering approaches for simulating a robotic space system architecture through automatic knowledge processing. *Systems Engineering*, 25(4):360–386, 2022.