E. Kamburjan, M. Wimmer, M. Ohrndorf, P. Rani, T. Thüm, J. Spieler, T. Kehrer (Hrsg.): SE2026 – Companion Proceedings,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2026 93

Towards Verifiability-Aware Variability Modeling using SysML v2 for Security- and Safety-Critical Avionics

Hendrik Kausch[1], Mathias Pfeiffer[1], Deni Raco[1], Amelie Rath[1], Bernhard Rumpe[1], Andreas Schweiger[2], and Stephan Jahnke[3]

Abstract: The increasing need for configurable and reusable avionics system architectures introduces significant challenges in the systematic handling of variability across structure, behavior, and requirements. Current model-based systems engineering practices provide limited support for analyzing such variability in a consistent and verifiability-aware manner, particularly in early design phases. This paper presents a unified, SysML v2–based methodological framework for modeling, instantiating, and analyzing variability in avionics system architectures. The approach is evaluated using an avionics component case study featuring two alternative architectural design variants. A model-based trade-off analysis demonstrates that one variant enables automated verification, while the other alternative incurs significantly higher verification effort. The automatically verifiable architecture is selected based on objective, model-derived criteria. Formal verification evidence is provided to document the feasibility of the proposed solution.

Keywords: Variability, SysML v2, Formal Verification, Security

1 Introduction

1.1 Motivation

Avionics systems are facing a continuous increase in complexity, driven in particular by a steadily growing share of software and its tight integration with hardware and system-level functions. This rise in complexity directly impacts development effort, verification scope, and certification activities, leading to significantly increased costs and extended development cycles [Br18]. In avionics, where compliance with stringent certification standards is mandatory, this complexity often results in conservative design decisions and limited design-space exploration, as each architectural change may entail substantial re-verification and re-certification effort.

The costs and risks associated with late-stage defect detection [Ba95; Be84; Bo76] highlight the urgent need for systematic approaches to control complexity, reduce verification effort, and support rigorous certification compliance. Formal and model-based methods (ED216, ED218) have shown promise in addressing these challenges, but their adoption is often limited by high skill requirements and uncertain success rates [An19].

[1] RWTH Aachen University, Chair of Software Engineering, Aachen, Germany,
, https://orcid.org/0000-0002-0988-6149
[2] Airbus Defence and Space GmbH, Manching, Germany,
[3] OHB System AG, Bremen, Germany,

doi:10.18420/se2026-ws-09

[KPR+26] H. Kausch, M. Pfeiffer, D. Raco, A. Rath, B. Rumpe, A. Schweiger, S. Jahnke: Towards Verifiability-Aware Variability Modeling using SysML v2 for Security- and Safety-Critical Avionics. In: Avionics Systems and Software Engineering Workshop of the Software Engineering 2026 - Companion Proceedings (AvioSE), pp. 93-112, DOI 10.18420/se2026-ws_07, Gesellschaft für Informatik e.V., Feb. 2026.

Model-Based Systems Engineering (MBSE) provides a foundational approach to managing complexity by shifting development from document-centric processes to coherent, analyzable system models. The combination of model-based engineering with mathematically founded specification and verification techniques enables early detection of design flaws and systematic certification support. Prior work has shown the feasibility of such approaches in realistic avionics settings [Ka20a; Ka22; Kr19] using the theorem prover Isabelle [NPW02]. Theorem provers as a deductive method offer the highest level of assurance compared to other formal methods like model-checking [CE82] or abstract interpretation [ABG22], but come with a higher cost and are generally less automated. Formal methods are used to analyze safety properties [Ka25a] and security properties [Ch23; Ka25b; Kl14], even if those properties, e.g., liveness properties [AS85], are not testable by conventional means.

MBSE enables the explicit modeling of *variability*, which allows multiple alternative solutions to be represented within a single system model. In the MBSE literature, this principle is often described as *underspecification* [BR07; BS01], where design decisions are intentionally left open in early development phases to preserve flexibility. This notion has been formalized and applied to security-critical systems, enabling automated reasoning over refinement steps [Ka20b; Ka23]. Thus, variableness provides a structured basis for verification, trade-off analysis, and design space exploration.

By emphasizing variability, MBSE also supports deferred decision-making and more effective application of formal methods. Such variability-aware approaches are essential for controlling verification and certification costs, potentially while exploring broader design spaces without incurring prohibitive effort. This perspective naturally leads to the use of variable system models, refinement strategies, and advanced frameworks such as SysML v2, which integrate variability directly into the core modeling language and provide the necessary foundation for automated analysis, formal verification [Ka21a], and consistent management of complex avionics systems. Other options for modeling variants include feature diagrams [Dr19; Gr08a] or System Entity Structure models [Fo17]. Since SysML is widely used in industry, this paper focuses on the existing implementation of variability in SysML v2.

Variable system models define an explicit *design space* that spans multiple design options. Such models support the systematic comparison of alternatives and enable structured design space exploration, including the execution of trade-off analyses across competing solutions. Engineers can reason about sets of alternatives and evaluate their implications in a controlled and transparent manner, before committing prematurely to a single architecture.

A key benefit of variability-aware modeling is its support for trade-off analysis and optimization. By capturing alternative structures, behaviors, and requirements, variability enables the evaluation of competing objectives such as power consumption, mass, physical dimensions, or cost. Design decisions can thus be deferred and incrementally optimized as additional information becomes available, a capability that is particularly important in early system design phases and long-running development programs. For instance, the number of

realizable variants can already be reduced significantly in early system design phases by safety and certification requirements [Ku25].

Beyond single systems, variability modeling has proven to be an effective means for unifying and managing *product families*. In domains such as automotive systems engineering, variable models are used to represent entire product lines within a single model, allowing commonality and variability to be captured explicitly. Such models can be incrementally evolved and refactored [PR01] over time, supporting both reuse and continuous refinement as products and technologies change. Managing variability across large sets of variants further benefits from using a single-source-of-truth database, which allows the efficient configuration of various test environments for simulating models while maintaining traceability [CHT25].

Variability itself is not static but can be refined throughout the development lifecycle. This refinement may take the form of adding detail to variants, constraining previously open choices, or eliminating infeasible alternatives. In the literature, this stepwise restriction and elaboration of variability is commonly referred to as *refinement*, and it provides a systematic mechanism for progressing from abstract solution spaces to concrete, implementable system configurations [PR97].

Finally, SysML v2 introduces a standardized, human-readable yet machine-processable textual syntax. This foundation enables improved automation, formal analysis, and tool-supported reasoning over variability and design spaces, thereby providing the basis for advanced analyses such as automated trade-off evaluation and formal verification [Bü20; Ka25c]. Further modeling languages for specifying distributed systems have been developed, such as the Palladio Component Model [BKR09], AutoFocus [VZ14], UML [Ru99], or Ptolemy [Le16]. However, this paper uses SysML because it is prominently used in the aerospace and automotive industries for systems engineering. Concrete verification capabilities are obtained by mapping SysML-based architectures and behaviors to mathematically founded semantic models, which has been demonstrated for an avionic system of a data link uplink feed [Ka21b; Ka22]. As a result, SysML v2 forms a promising foundation for integrating variability modeling, design space exploration, and analysis within a unified MBSE framework.

With SysML v2, variability becomes a first-class concern of the language. SysML v2 formally introduces variation points and variants for requirements, structural elements, and behavior, thereby integrating variability directly into the core language concepts. This integration improves consistency across variants, reduces engineering errors caused by manual duplication or informal documentation, and supports more systematic and repeatable design processes. In SysMLv2, variability may apply to system structure, behavior, and requirements alike. This represents a significant evolution compared to SysML v1, which was not specifically designed for variability modeling. In SysML v1, variability was often captured in separate variability models or through tool-specific stereotypes, creating a risk that variability descriptions would diverge from the actual system models and limiting automated analysis.

1.2 Contribution

This paper contributes a systematic approach for modeling, managing, and analyzing variability in avionics systems using SysML v2, building on established variability concepts and capabilities from the automotive domain and transferring them to the aerospace and avionics context. The proposed contributions are structured along the following topics and culminate in a trade-off study with a particular focus on verifiability, on the example of an avionics security property.

We introduce a structured transfer of variability knowledge and modeling capabilities from the automotive domain to SysML v2–based models for avionics systems, enabling a systematic comparison of alternative design options in early avionics system design. We provide modeling concepts to precisely capture and differentiate structural characteristics of avionics systems in SysML v2. We enable mission- and configuration-specific instantiations without structural duplication, supporting reuse and consistency across variants. We extend variability modeling from purely structural aspects to include functional, behavioral, and requirements dimensions. As such, we introduce variability concepts for system behavior, allowing alternative behavioral realizations to be modeled and analyzed. We integrate variability in functional architectures and their associated behaviors. To support informed design decisions, we establish foundations for trade-off analyses across structurally and behaviorally variable models. We develop methods for the unambiguous identification of structurally valid avionics system configurations. We show approaches for managing and reasoning about requirements variability in avionics systems. Lastly, we provide tool-supported mechanisms for the evaluation and optimization of non-functional metrics, such as power consumption, mass, and physical dimensions, while considering certification constraints, for example, those defined in ED-201A for aeronautical information system security [Ka25a]. As such, we present tool-assisted generation and certification-aware validation of avionics system configurations, tool-supported trade-off analysis, and certification support for variable avionics system designs.

The proposed approach is evaluated in an avionics case study, where multiple design alternatives are compared in a trade-off study with a specific focus on verifiability. To address the identified challenges in managing verification complexity across variable architectures and behaviors, the study deliberately leverages the supplements of EUROCAE ED-12C / RTCA DO-178C, enabling the systematic application of model-based development and verification methods in accordance with RTCA DO-331 / EUROCAE ED-218, as well as formal methods as defined by RTCA DO-333 / EUROCAE ED-216. This standards-compliant use of model-based and formal techniques allows verification effort, traceability completeness, and certification impact to be assessed as explicit trade-off dimensions across alternative configurations.

2 SysML v2 as an Enabler for Variability

SysML v2 represents a significant evolution of system modeling languages by providing a more precise and tool-interpretable language for system engineering. In contrast to purely graphical notations, SysML v2 introduces a standardized *textual syntax* that is both human-readable and machine-interpretable. The SysML v2 tool used in this paper was created using MontiCore [HKR21; HR17], a state-of-the-art textual language workbench. This enables automated analysis, consistency checking, and transformation of system models, while preserving accessibility for engineers across disciplines. As a result, SysML v2 forms a suitable basis for advanced model-based engineering practices, including systematic variability modeling and analysis.

Variability modeling has long been recognized as a key enabler for managing complexity in large-scale, configurable systems. Variability modeling concepts were, for example, being developed and applied in the automotive domain, addressing challenges such as product line engineering, configuration management, and variant-aware analysis [Gr08b; Gu22; Ha11a; Ha11b; Ha11c; Ha12; Ha13a; Ha13b; Ha15a; Kr17]. Explicit variability representations support reuse, consistency, and systematic trade-off analyses across large design spaces. In addition, architectural considerations play a central role in structuring variability and ensuring analyzability across variants [Bi25].

However, SysML v1 was not originally designed with variability modeling as a first-class concern [Ep24; WFM25]. As a consequence, variability was often captured in separate, specialized variability models or annotations, rather than being an integral part of the actual system models. This separation introduced the risk of inconsistencies between variability descriptions and system architectures, limited automated analysis, and increased the potential for design and verification errors.

SysML v2 addresses these limitations by providing explicit and formally defined concepts for *variation points* and *variants*. By integrating variability directly into the system model, SysML v2 enables engineers to express alternative structures, behaviors, and requirements in a uniform and analyzable manner. This integration improves design processes, reduces modeling and configuration errors, and ensures consistency across system variants throughout the engineering lifecycle.

The remainder of this section is structured into four subsections, each focusing on a specific aspect of variability modeling in SysML v2. These subsections address structural variability, functional and behavioral variability, requirements variability, and the foundations for variant-aware analysis and trade-off evaluation, thereby establishing SysML v2 as a central enabler for systematic variability management in security-critical avionics systems.

2.1 Structural Variability Modeling

SysML v2 introduces the `variation` and `variant` keywords as first-class language concepts for explicitly modeling variability within system models. A `variation` defines the space of permitted alternatives by specifying what may differ and under what rules. In contrast, a `variant` represents a concrete, selectable alternative that conforms to a variation. These two constructs allow engineers to define variation points and their associated variants directly in the architectural model, thereby making variability an integral part of the system description rather than an external annotation.

In addition, as part of our participation in the *ESA MBSE Workshop 2025* in Vilnius [4], we contributed to an open-source SysML v2 model [5] that explicitly captures structural variability of a CubeSat platform. In this model, alternative structural configurations are represented using SysML v2 `variation` and `variant` constructs. An excerpt of this model is shown in Listing 1.

```
1  // Library, i.e., ESA or OHB profile
2  part def Platform;
3  part def Payload;
4
5  // Library, i.e., ESA or supplier data sheet
6  part def Cubesat;
7
8  // Variable system, i.e., system where trade-off is to be performed on
9  variation part def CometIOption {
10         variant part platform: Platform;
11         variant part payload: Payload;
12         variant part cubesat: Cubesat[1..3];
13 }
14
15 // Options, e.g, provided by suppliers
16 part platform1 : Platform;
17 part platform2 : Platform;
18
19 part payload1 : Payload;
20 part payload2 : Payload;
```

List. 1: Excerpt of a SysML v2 model illustrating structural variability of a CubeSat architecture using `variation` (space of permitted alternatives) and `variant` (concrete, selectable alternative that conforms to a variation) constructs.

To enable precise representation of *structural differences* between alternative system architectures, the keywords `variation` and `variant` are used. This is particularly relevant for avionics systems, where architectural variability arises from mission-specific requirements, redundancy concepts, or component reuse. By explicitly capturing such structural alternatives

[4] https://www.b2match.com/e/mbse-2025
[5] Model available at https://beta.sysand.org/

in SysML v2, configurable system architectures can be systematically planned, analyzed, and compared.

Furthermore, SysML v2 provides the mechanisms `subsets` (abbreviated `:>`) and `redefines` (`::>`) to precisely instantiate specific realizations of system elements by restricting or overriding existing structures. These constructs enable the creation of consistent yet variability-capable avionics system models, allowing engineers to define alternative configurations while preserving model integrity. An illustrative example of such a CubeSat subsystem using `subsets` and `redefines` is shown in Listing 2.

```
1  // Options, e.g, provided by suppliers
2  part platform1 : Platform;
3  part platform2 : Platform;
4
5  part payload1 : Payload;
6  part payload2 : Payload;
7
8  // Configuration
9  part def 'CometI Option 1' subsets CometIOption {
10
11         // Integrating supplier-parts directly
12         part platformChoice redefines platform = platform1;
13         part payloadChoice redefines payload = payload1;
14
15         // Assuming the existence of a single Cubesat
16         part cubesatChoice redefines cubesat : Cubesat[1];
17 }
```

List. 2: Example of a CubeSat subsystem model demonstrating the use of `subsets` (":>") and `redefines` ("::>") for variability-aware instantiation in SysML v2.

The feasibility and benefits of using SysML v2 for structural variability modeling have already been demonstrated, e.g., using an aerospace stowage example [Ep24], highlighting improved clarity and reuse of architectural variants.

2.2 Parametric and Configuration-Specific Variability

SysML v2 enables parametric and configuration-specific variability through `attributes` (see List. 3, line 2), which can be used to assign parameter values to system elements. These attributes allow avionics system structures to be flexibly adapted to mission-specific or configuration-specific requirements, enabling variability without structural duplication. The granularity of this flexibility is determined by *typing*. For instance, the `Naturals` type restricts attribute values to natural numbers:

$$0, 1, 2, 3, 4, 5, \ldots$$

The allowed range of attribute values can be further restricted using constraints (see List. 3, line 6-8, 12-14). E.g., if the sum of weights in a particular configuration amounted to more than the maximum weight stated by a constraint, the configuration would be deemed invalid. Generally speaking, checking of constraint adherence can be done through simulation, model checkers, or theorem provers. This combination of attributes, typing, and constraints provides a powerful mechanism to define configuration-specific variability in a systematic and analyzable way (see List. 3, line 18-20).

```
1  part def Payload {
2          attribute weight: Natural;
3  }
4
5  part def HeavyPayload specializes Payload {
6          assert constraint {
7                  150 [kg] < weight < 300 [kg]
8          }
9  }
10
11 part def MediumPlatform :> Platform {
12         assert constraint {
13                 8,5 [kg] < weight < 35 [kg]
14         }
15 }
16
17 part def 'Comet Interceptor - Option 2' :> CometInterceptorOption {
18         assert constraint {
19                 platform.weight + payload.weight < 100 [kg];
20         }
21 }
```

List. 3: Example of a CubeSat subsystem illustrating parametric and configuration-specific variability using `attributes` in SysML v2.

2.3 Extensions to Functional Structures and Behavior

It is increasingly recognized that variability should not be limited to structural aspects, but should also be applied to system behavior. In SysML v2, the core modeling technique for behavior is the use of *state machines* [Gr96] or *automata* [PR94]. Behavioral modeling answers how a component or system responds to its inputs. In the context of avionics systems, a given function may produce different outputs for identical sensor data, depending on the current system flight state.

The purpose of `States` in SysML v2 is to allow the system to react differently to the same input depending on the current state, while capturing all information available at the time. If no state-specific behavior is defined, the system is considered underspecified, meaning that any behavior is possible (*chaos*). This allows the model to remain flexible, supporting

incremental refinement. The model can be updated with additional details as they become available. Such an approach also enables the evaluation of design alternatives and trade-offs. For example, a more precisely defined function might increase power consumption, which can then be analyzed in the model.

Listing 4 illustrates a SysML v2 state machine of an equipment subsystem, showing underspecified power states. While initial information about switching the equipment on and the associated power consumption is included, all other possible behaviors remain underspecified, highlighting the flexibility and refinement potential inherent in variability-aware behavior modeling.

```
1  part def Equipment {
2          port cmdPowerOn  : ~Signal;
3          port powerCmdOut : Signal;
4
5          exhibit state 'Equipment power modes' {
6                  entry;
7                  then state OFF;
8                  then state <STBY> 'Standby';
9
10                 attribute powerConsumption := 0;
11
12                 transition
13                         first STBY
14                         accept cmdPowerOn
15                         if safeToPowerOn()
16                         do {
17                                 send onCmd to powerCmdOut;
18                                 assign powerConsumption := 20.0;
19                         }
20                         then ON;
21
22                 state ON;
23
24                 transition first OFF then ON;
25                 transition first STBY then OFF;
26                 transition first ON then OFF;
27                 transition first ON then STBY;
28         }
29 }
```

List. 4: Example of a SysML v2 state machine (keyword “state” in line 5) representing underspecified power states of equipment. The initial behavior is given after the “entry” keyword, see line 6-10, which sets the initial state and sets the initial value of the consumption. The power-on behavior is modeled by a “transition”, line 12-20. Other behaviors remain underspecified as transitions with no trigger, guard, or action defined, cf. line 24-27, allowing for future refinement.

In addition to state-based behavioral variability, SysML v2 enables modeling of large-scale state machines that encompass multiple operational modes across different aircraft types

(150% models). A 150% model is an intentionally overcomplete version of a design that includes all possible features, options, or variants, even though the final product will use only a subset. It's commonly used in systems engineering [Bu19; Gr08a; Gr08c] and product configuration to derive specific "100%" models by selecting the needed elements. Hence, these models describe a superset of potential solutions, where the final configuration or behavior for a specific aircraft must still be selected, either manually or automatically. Specific realizations are constrained through rules, such as UML feature diagrams or other formal constraints, allowing the model to remain both flexible and analyzable while ensuring consistency with system requirements.

Functional structures in SysML v2, typically represented as networks of *functional components* (`Parts`), complement variability-aware behavior modeling. Components are connected via *interconnections* that represent the transfer of energy, material, or information (analogous to IBDs in SysML v1). As with behavioral models, variable functional structures include optional components and connections that are used in specific configurations only. The selection and restriction of valid configurations can be governed by UML feature diagrams, as often done in the automotive domain, or by more expressive constraints enabled in SysML v2.

2.4 Trade-Off Concepts for Avionics Systems

Trade-off analysis is a key activity in the design of complex avionics systems, enabling systematic comparison of multiple alternatives. Within an MBSE context, trade-off analysis typically compares instances of variable models according to one or more metrics, while these instances can still retain internal variability. This allows engineers to evaluate design alternatives without prematurely committing to a single configuration, supporting flexibility and informed decision-making throughout the development process.

Example metrics for trade-off analysis [FHR08; Ha13b; Ka25a; Ka25b; KRR15] in avionics MBSE include:

- **Structural and complexity metrics:** number of components, depth of hierarchy, number of components per hierarchy level (branching factor)
- **Variability metrics:** number of remaining features, similarity of remaining options (homogeneity), number of remaining options, configuration effort (number of decisions required to finalize the model)
- **Quality metrics:** analyzability, consistency (contradiction-free), test coverage, verification coverage, automation of certifiability (time savings), degree of certification (e.g., 4-eyes principle vs. testing vs. formal verification)
- **Specific system properties:** mass optimization, energy consumption optimization, payload optimization, and other mission-specific performance indicators

By explicitly considering *verifiability* as a trade-off metric, engineers can identify architectural solutions that not only optimize performance and variability but also minimize the effort required for certification and formal verification. This approach is particularly valuable in avionics development, where strict safety and security requirements make automated verification and certifiability a critical design objective. For instance, finding a proof for desired constraints with an automatic prover tool like MontiBelle can show the verifiability without additional expert knowledge while also reducing certification effort in accordance with ED-218 [Ka25a].

3 Trade-Off Analysis of Verifiability for a Prioritized Avionics Downlink Component

Figure 1 illustrates the development of a prioritized, security-critical avionics downlink component (DLDF) within the context of a larger system-of-systems. The development was driven by 18 system requirements, including one that mandated assurance of a security-critical *liveness* property [Ka24a; Ka24b], e.g., prevention of a Denial-of-Service attack by overloading the downlink with messages. Liveness properties are notoriously challenging to verify: they cannot be exhaustively tested, since conventional testing can only demonstrate the *absence* of failures but cannot prove that the property always holds. As such, formal verification methods are required to demonstrate satisfaction of liveness properties, as regulated by standards such as EUROCAE ED-12C/DO-178C supplements and ED-216 [Th12].

To evaluate alternative design approaches for the DLDF, a trade-off analysis was conducted using the metric of *verifiability*: the ability to generate the highest level of assurance using sophisticated formal methods (e.g., theorem provers) within a reasonable timeframe, typically a few hours after model creation. The first option (*Variant 1*, bottom left in Figure 1) was modeled as a single component with one internal state machine representing the component's behavior. The second option (*Variant 2*, bottom right) followed a rigorous methodology based on FOCUS [BS01], SpesML [SpesML23], and the avionics-oriented MontiBelle framework [Ka25a]. In FOCUS, distributed and interactive systems consist of components exchanging messages through unidirectional channels. The semantics of a component is a (set of) stream processing functions where each function represents a potential behavior. Behavioral refinement is then represented by set inclusion. Concurrency is represented by an appropriate composition operator connecting channels. The most important reason for using Focus in this article is due to the fact that its refinement mechanism is fully compositional [BR07].

The MontiBelle approach identifies three key classes of system models: (1) Declarative specifications, (2) Architecture, and (3) Imperative specifications. To narrow the gap between typically informal System Requirements (SRs) and formal Low-Level Requirements (LLRs), e.g., an automaton describing the system's behavior, formal High-Level Requirements

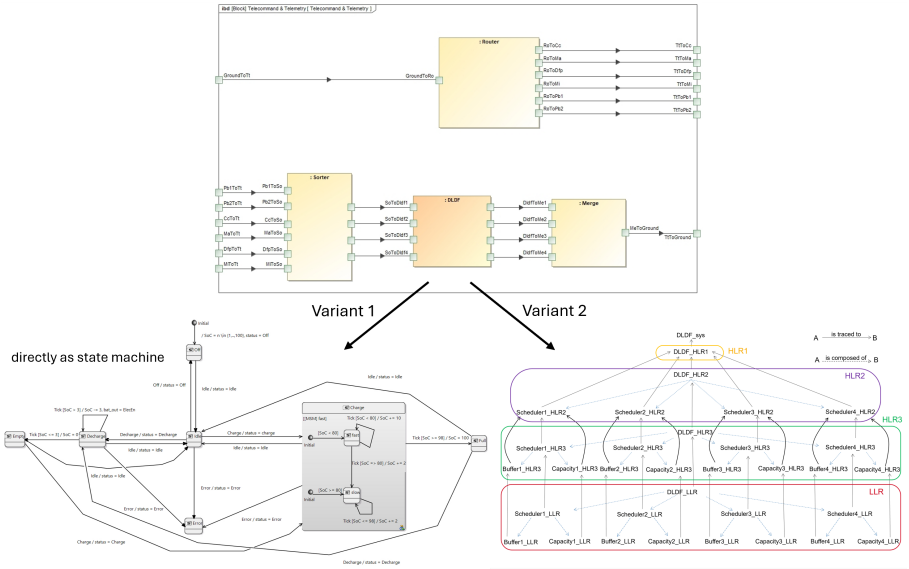

Fig. 1: Trade-off analysis of a prioritized avionics downlink component (DLDF) for verifiability. On the top, the context of a larger system-of-systems is shown. Bottom left: Variant 1 with a single component and state machine. Bottom right: Variant 2 developed via FOCUS/SpesML/MontiBelle methodology, with five hierarchy levels, 36 components, and 27 proof obligations. The system is considered in the context of the larger system-of-systems.

(HLRs) are introduced. HLRs are formalized as declarative specifications (1) over communication histories. To reach a feasible fine-grained architecture, HLRs are decomposed into communication architectures (2) of more detailed and specialized HLRs. Decomposition levels are linked using refinement relations, ensuring traceability and enabling verifiability of compliance and consistency. Each refinement relation results in a proof obligation, i.e., an unfinished (yet to be proven) theorem. To formalize LLRs, an imperative (3) and thus more implementation-oriented technique should be used. We propose the use of automata, specifically event-based automata, for software-intensive systems. A requirement described by an automaton is consistent, since it itself describes one possible implementation. The modeling of event-based automata is done using a profile for SysML v2 developed with the language workbench MontiCore [Ha15b; HR17; HRW18]. Traceability and verifiability are once again achieved using refinement relations. The compliance is assured by resolving the resulting proof obligation, i.e., typically an inductive proof over (the length of) communication histories, i.e., streams. Combining architecture and LLRs into a system design is achieved by refining the final HLR architecture to an equally structured architecture composed from an LLR.

Variant 2 in Fig. 1 comprises a hierarchical decomposition with five levels, across a total of 36 components: 13 abstract specifications, 10 state-based components, and 13 compositions

— summarized in Tab. 1. This resulted in 27 proof obligations, representing the set of properties to be formally verified. The 27 proof obligations are handled in the theorem prover Isabelle and checked automatically until the final proof, i.e., that the composition ($\bigotimes$) of LLRs fulfills (refines, $\subseteq$) the formalized non-starvation SR (DLDF_HLR1).

```
theorem ref: "Scheduler1_LLR ⨂ Scheduler2_LLR ⨂ Scheduler3_LLR
⨂ Scheduler4_LLR ⊆ DLDF_HLR1"
```

This final proof builds upon refinement relations between each sub-component and its requirements and then leverages the transitivity and compositionality of the refinement relation in FOCUS to automatically verify compliance between the DLDF system and its formal requirements.

Component Type	Count
Abstract specifications	13
State-based components	10
Compositions	13
Total proof obligations	27
Hierarchy levels	5

Tab. 1: Summary of Variant 2 components and verification proof obligations.

Thus, variant 2's verifiability of adherence to system requirements was ensured and the variant is considered the primary option for the DLDF system.

4 Conclusion

This paper presents a systematic approach for modeling, managing, and analyzing variability in avionics systems using SysML v2, building on established variability concepts from the automotive domain and transferring them to the aerospace and avionics context. The proposed approach enables early, model-based trade-off decisions, allowing engineers to systematically compare alternative design options and assess the impact of architectural choices on critical metrics such as verifiability.

We have introduced modeling concepts to precisely capture structural characteristics, enable mission- and configuration-specific instantiations without structural duplication, and extend variability modeling from purely structural aspects to functional, behavioral, and requirements dimensions. This includes variability-aware behavioral models, functional architectures, and requirements handling, supporting reuse, consistency, and systematic exploration of alternative configurations. The framework also provides methods for the unambiguous identification of structurally valid configurations and tool-supported evaluation and optimization of non-functional metrics such as power consumption, mass, and physical dimensions, while considering certification constraints.

The proposed approach has been validated in an avionics case study, where a prioritized, security-critical downlink component (DLDF) was analyzed across multiple design variants in a trade-off study with a focus on verifiability. By leveraging model-based development in compliance with EUROCAE ED-12C / RTCA DO-178C supplements and applying formal verification techniques defined in RTCA DO-333 / EUROCAE ED-216, we were able to fully formally verify the safe and correct implementation of the prioritized link for the selected variant. The study demonstrated that the verifiability metric—measuring the ability to apply sophisticated formal methods efficiently—provides a practical basis for early architectural decisions and trade-offs.

Scalability is achieved using the decomposition capabilities provided in SysMLv2, where three layers of 10 components already allow up to 1000 components to be represented, analyzed, and partially verified. The use of theorem-proving tools such as Isabelle offers explicit evidence for security-critical requirements while supporting systematic trade-off analysis.

The use of formal verification to reduce testing effort has direct implications for tool qualification in security-critical avionics development. In order to replace or complement testing activities, the involved tools must be qualified according to RTCA DO-330 / EUROCAE ED-215. Prior work has demonstrated that Isabelle can be qualified for proofs of functional correctness in avionics contexts, leveraging its small and trusted kernel and conservative extensions [An15; Kl14; NPW02]. The conservative approach used in Isabelle ensures that no inconsistencies are introduced in generated theories [Hu12]. In a SysML v2–based variability modeling workflow, this also implies qualification of the model-to-proof generator that translates variable system models into formal theories. Ensuring traceability between requirements, models, and verification artifacts is essential to achieve coverage objectives comparable to requirements-based and structural test coverage as required by EUROCAE ED-12C.

In summary, this work contributes a structured methodology for transferring variability knowledge from the automotive domain to aerospace and avionics applications, integrated into SysML v2. The approach enables comprehensive trade-off analyses, including certification-aware evaluation, and supports systematic reasoning about structural, functional, behavioral, and requirements variability. The avionics case study demonstrates the practical benefits of early, model-based design decisions, the impact of architectural choices on verifiability, and the feasibility of fully formally verifying critical system components in a complex system-of-systems context, thereby laying a foundation for shared variability handling approaches in avionics.

References

[ABG22] Ascari, F.; Bruni, R.; Gori, R.: Limits and difficulties in the design of under-approximation abstract domains. In (Bouyer, P.; Schröder, L., eds.): Foundations

of Software Science and Computation Structures. Springer International Publishing, Cham, pp. 21–39, 2022.

[An15] Andronick, J.: Please check my 500K LOC of Isabelle. In: Qualification of Formal Methods Tools – Report from Dagstuhl Seminar 15182. 2015.

[An19] Annighoefer, B. et al.: Challenges and Ways Forward for Avionics Platforms and their Development in 2019. In: 38th Digital Avionics System Conference (DASC). 2019, http://hdl.handle.net/11420/4819.

[AS85] Alpern, B.; Schneider, F. B.: Defining liveness. Information Processing Letters 21 (4), pp. 181–185, 1985, https://www.sciencedirect.com/science/article/pii/0020019085900560.

[Ba95] Baziuk, W.: BNR/NORTEL: path to improve product quality, reliability and customer satisfaction. In: Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE'95. Pp. 256–262, 1995.

[Be84] Beizer, B.: Software System Testing and Quality Assurance. Van Nostrand Reinhold Co., USA, 1984.

[Bi25] Bindick, S. et al.: Communication and Architectural Patterns for Developing Distributed Systems. In: 2025 8th International Conference on Software and System Engineering (ICoSSE). Pp. 31–38, 2025.

[BKR09] Becker, S.; Koziolek, H.; Reussner, R.: The Palladio Component Model for Model-Driven Performance Prediction. Journal of Systems and Software 82, pp. 3–22, 2009.

[Bo76] Boehm, B. W.: Software Engineering. IEEE Transactions on Computers C-25 (12), pp. 1226–1241, 1976.

[BR07] Broy, M.; Rumpe, B.: Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung. German, Informatik-Spektrum 30 (1), pp. 3–18, 2007, http://www.se-rwth.de/staff/rumpe/publications20042008/Modulare-hierarchische-Modellierung-als-Grundlage-der-Software-und-Systementwicklung.pdf.

[Br18] Brahmi, A. et al.: Formalise to automate: deployment of a safe and cost-efficient process for avionics software. In: 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018). Toulouse, France, 2018, https://hal.archives-ouvertes.fr/hal-01708332.

[BS01] Broy, M.; Stølen, K.: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer, New York, 2001.

[Bu19] Butting, A. et al.: Systematic Composition of Independent Language Features. Journal of Systems and Software (JSS) 152, ed. by Sevilla, R. C.; Fuentes, L.; Lochau, M., pp. 50–69, 2019, http://www.se-rwth.de/publications/Systematic-Composition-of-Independent-Language-Features.pdf.

[Bü20] Bürger, J. C. et al.: Towards an Isabelle Theory for distributed, interactive systems - the untimed case. Shaker Verlag, 2020.

[CE82] Clarke, E. M.; Emerson, E. A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In (Kozen, D., ed.): Logics of Programs. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 52–71, 1982.

[Ch23] Cheval, V. et al.: Hash Gone Bad: Automated discovery of protocol attacks that exploit hash function weaknesses. In: 32nd USENIX Security Symposium (USENIX Security 23). USENIX Association, Anaheim, CA, pp. 5899–5916, 2023.

[CHT25] Chrysalidis, P.; Halle, M.; Thielecke, F.: Establishing a Single Source of Truth for Avionics Platform Verification Through UCoF. In. Pp. 1–10, 2025.

[Dr19] Drave, I. et al.: Semantic Evolution Analysis of Feature Models. In (Berger, T. et al., eds.): International Systems and Software Product Line Conference (SPLC'19). ACM, Paris, pp. 245–255, 2019, http://www.se-rwth.de/publications/Semantic-Evolution-Analysis-of-Feature-Models.pdf.

[Ep24] Epp, J. et al.: Transitioning towards SysML v2 as a Variability Modeling Language. Innovations in Systems and Software Engineering 20 (4), pp. 585–596, 2024.

[FHR08] Fieber, F.; Huhn, M.; Rumpe, B.: Modellqualität als Indikator für Softwarequalität: eine Taxonomie. German, Informatik-Spektrum 31 (5), pp. 408–424, 2008, http://www.se-rwth.de/staff/rumpe/publications20042008/Modellqualitaet-als-Indikator-fuer-Softwarequalitaet-eine-Taxonomie.pdf.

[Fo17] Folkerts, H. et al.: Variability Modeling for Engineering Applications. SNE Simulation Notes Europe 27, pp. 167–176, 2017.

[Gr08a] Grönniger, H. et al.: Modelling Automotive Function Nets with Views for Features, Variants, and Modes. In: Proceedings of 4th European Congress ERTS - Embedded Real Time Software. 2008, http://www.se-rwth.de/staff/rumpe/publications20042008/Modelling-Automotive-Function-Nets-with-Views-for-Features-Variants-and-Modes.pdf.

[Gr08b] Grönniger, H. et al.: View-Centric Modeling of Automotive Logical Architectures. In: Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme IV. Informatik Bericht 2008-02, TU Braunschweig, 2008, http://www.se-rwth.de/staff/rumpe/publications20042008/View-Centric-Modeling-of-Automotive-Logical-Architectures.pdf.

[Gr08c] Grönniger, H. et al.: Modeling Variants of Automotive Systems using Views. In: Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen. Informatik Bericht 2008-01, TU Braunschweig, pp. 76–89, 2008, http://www.se-rwth.de/staff/rumpe/publications20042008/Modeling-Variants-of-Automotive-Systems-using-Views.pdf.

[Gr96] Grosu, R. et al.: State Transition Diagrams, tech. rep., TU Munich, 1996, https://www.se-rwth.de/staff/rumpe/publications/State-Transition-Diagrams.pdf.

[Gu22] Gupta, R. et al.: Implementation of the SpesML Workbench in MagicDraw. In: Modellierung 2022 Satellite Events. Gesellschaft für Informatik e.V., pp. 61–76, 2022, http://www.se-rwth.de/publications/Implementation-of-the-SpesML-Workbench-in-MagicDraw.pdf.

[Ha11a] Haber, A. et al.: Delta-oriented Architectural Variability Using MontiCore. In: Software Architecture Conference (ECSA'11). ACM, 6:1–6:10, 2011, http://www.se-rwth.de/publications/Delta-oriented-Architectural-Variability-Using-MontiCore.pdf.

[Ha11b] Haber, A. et al.: Hierarchical Variability Modeling for Software Architectures. In: Software Product Lines Conference (SPLC'11). IEEE, pp. 150–159, 2011, http://www.se-rwth.de/publications/Hierarchical-Variability-Modeling-for-Software-Architectures.pdf.

[Ha11c] Haber, A. et al.: Delta Modeling for Software Architectures. In: Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteterSysteme VII. fortiss GmbH, pp. 1–10, 2011, http://www.se-rwth.de/publications/Delta-Modeling-for-Software-Architectures.pdf.

[Ha12] Haber, A. et al.: Evolving Delta-oriented Software Product Line Architectures. In: Large-Scale Complex IT Systems. Development, Operation and Management, 17th Monterey Workshop 2012. LNCS 7539, Springer, pp. 183–208, 2012, http://www.se-rwth.de/publications/Evolving-Delta-oriented-Software-Product-Line-Architectures.pdf.

[Ha13a] Haber, A. et al.: Engineering Delta Modeling Languages. In: Software Product Line Conference (SPLC'13). ACM, pp. 22–31, 2013, http://www.se-rwth.de/publications/Engineering-Delta-Modeling-Languages.pdf.

[Ha13b] Haber, A. et al.: First-Class Variability Modeling in Matlab/Simulink. In: Variability Modelling of Software-intensive Systems Workshop (VaMoS'13). ACM, pp. 11–18, 2013, http://www.se-rwth.de/publications/First-Class-Variability-Modeling-in-Matlab-Simulink.pdf.

[Ha15a] Haber, A. et al.: Systematic Synthesis of Delta Modeling Languages. Journal on Software Tools for Technology Transfer (STTT) 17 (5), pp. 601–626, 2015, http://www.se-rwth.de/publications/Systematic-synthesis-of-delta-modeling-languages.pdf.

[Ha15b] Haber, A. et al.: Integration of Heterogeneous Modeling Languages via Extensible and Composable Language Components. In: Model-Driven Engineering and Software Development Conference (MODELSWARD'15). SciTePress, pp. 19–31, 2015, http://www.se-rwth.de/publications/Integration-of-Heterogeneous-Modeling-Languages-via-Extensible-and-Composable-Language-Components.pdf.

[HKR21] Hölldobler, K.; Kautz, O.; Rumpe, B.: MontiCore Language Workbench and Library Handbook: Edition 2021. Shaker Verlag, 2021.

[HR17] Hölldobler, K.; Rumpe, B.: MontiCore 5 Language Workbench Edition 2017. Shaker Verlag, 2017.

[HRW18] Hölldobler, K.; Rumpe, B.; Wortmann, A.: Software Language Engineering in the Large: Towards Composing and Deriving Languages. Journal Computer Languages, Systems & Structures 54, pp. 386–405, 2018, https://www.se-rwth.de/publications/Software-Language-Engineering-in-the-Large-Towards-Composing-and-Deriving-Languages.pdf.

[Hu12] Huffman, B. C.: HOLCF '11: A definitional domain theory for verifying functional programs. Portland State University, Portland, Or., 2012.

[Ka20a] Kausch, H. et al.: MontiBelle - Toolbox for a Model-Based Development and Verification of Distributed Critical Systems for Compliance with Functional Safety. In: AIAA Scitech 2020 Forum. American Institute of Aeronautics and Astronautics, Orlando, 2020, http://www.se-rwth.de/publications/MontiBelle-Toolbox-for-a-Model-Based-Development-and-Verification-of-Distributed-Critical-Systems-for-Compliance-with-Functional-Safety.pdf.

[Ka20b] Kausch, H. et al.: An Approach for Logic-based Knowledge Representation and Automated Reasoning over Underspecification and Refinement in Safety-Critical Cyber-Physical Systems. In (Hebig, R.; Heinrich, R., eds.): Combined Proceedings of the Workshops at Software Engineering 2020. Vol. 2581, CEUR Workshop Proceedings, Innsbruck, 2020, http://www.se-rwth.de/publications/An-Approach-for-Logic-based-Knowledge-Representation-and-Automated-Reasoning-over-Underspecification-and-Refinement-in-Safety-Critical-Cyber-Physical-Systems.pdf.

[Ka21a] Kausch, H. et al.: Model-Based Development and Logical AI for Secure and Safe Avionics Systems: A Verification Framework for SysML Behavior Specifications. In: Aerospace Europe Conference 2021 (AEC 2021). Council of European Aerospace Societies (CEAS), 2021, http://www.se-rwth.de/publications/Model-Based-Development-and-Logical-AI-for-Secure-and-Safe-Avionics-Systems-A-Verification-Framework-for-SysML-Behavior-Specifications.pdf.

[Ka21b] Kausch, H. et al.: Model-Based Design of Correct Safety-Critical Systems using Dataflow Languages on the Example of SysML Architecture and Behavior Diagrams. In (Götz, S. et al., eds.): Proceedings of the Software Engineering 2021 Satellite Events. Vol. 2814, CEUR, 2021, http://www.se-rwth.de/publications/Model-Based-Design-of-Correct-Safety-Critical-Systems-using-Dataflow-Languages-on-the-Example-of-SysML-Architecture-and-Behavior-Diagrams.pdf.

[Ka22] Kausch, H. et al.: Correct and Sustainable Development Using Model-based Engineering and Formal Methods. In: 2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC). IEEE, 2022, http://www.se-rwth.de/publications/Correct-and-Sustainable-Development-Using-Model-based-Engineering-and-Formal-Methods.pdf.

[Ka23] Kausch, H. et al.: A Theory for Event-Driven Specifications Using Focus and MontiArc on the Example of a Data Link Uplink Feed System. In (Groher, I.; Vogel, T., eds.): Software Engineering 2023 Workshops. Gesellschaft für Informatik e.V., pp. 169–188, 2023, http://www.se-rwth.de/publications/A-Theory-for-Event-Driven-Specifications-Using-Focus-and-MontiArc-on-the-Example-of-a-Data-Link-Uplink-Feed-System.pdf.

[Ka24a] Kausch, H. et al.: Applied Model-Based Co-Development for Zero-Emission Flight Systems Based on SysML. In: Deutscher Luft- und Raumfahrtkongress (DLRK 2024). Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V., 2024, http://www.se-rwth.de/publications/Applied-Model-Based-Co-Development-for-Zero-Emission-Flight-Systems-Based-on-SysML.pdf.

[Ka24b] Kausch, H. et al.: Enhancing System-model Quality: Evaluation of the MontiBelle Approach with the Avionics Case Study on a Data Link Uplink Feed System. In: Avionics Systems and Software Engineering Workshop of the Software Engineering 2024 - Companion Proceedings (AvioSE). Gesellschaft für Informatik e.V., pp. 119–138, 2024, https://www.se-rwth.de/publications/Enhancing-System-model-Quality-Evaluation-of-the-MontiBelle-Approach-with-the-Avionics-Case-Study-on-a-Data-Link-Uplink-Feed-System.pdf.

[Ka25a] Kausch, H. et al.: Model-driven Development for Functional Correctness of Avionics Systems: A Verification Framework for SysML Specifications. Council of European Aerospace Societies Aeronautical Journal (CEAS 16 (1), pp. 33–48, 2025, http://www.se-rwth.de/publications/Model-driven-Development-for-Functional-Correctness-of-Avionics-Systems-A-Verification-Framework-for-SysML-Specifications.pdf.

[Ka25b] Kausch, H. et al.: Enhancing System Model Quality: Evaluation of the Systems Modeling Language (SysML)-Driven Approach in Avionics. Journal of Aerospace Information Systems (JAIS) 22 (5), pp. 367–378, 2025, http://www.se-rwth.de/publications/Enhancing-System-Model-Quality-Evaluation-of-the-Systems-Modeling-Language-SysML-Driven-Approach-in-Avionics.pdf.

[Ka25c] Kausch, H. et al.: Towards an Isabelle Theory for Distributed, Interactive, Real-Time Systems Volume 2. Shaker Verlag, 2025.

[Kl14] Klein, G. et al.: Comprehensive Formal Verification of an OS Microkernel. ACM Trans. Comput. Syst. 32 (1), 2014, https://doi.org/10.1145/2560537.

[Kr17] Kriebel, S. et al.: The Next Generation of BMW's Electrified Powertrains: Providing Software Features Quickly by Model-Based System Design. In: 26th Aachen Colloquium Automobile and Engine Technology. 2017, http://www.se-rwth.de/publications/The-Next-Generation-of-BMWs-Electrified-Powertrains-Providing-Software-Features-Quickly-by-Model-Based-System-Design.pdf.

[Kr19] Kriebel, S. et al.: Model-Based Engineering for Avionics: Will Specification and Formal Verification e.g. Based on Broy's Streams Become Feasible? In (Krusche, S. et al., eds.): Proceedings of the Workshops of the Software Engineering Conference. Workshop on Avionics Systems and Software Engineering (AvioSE'19). Vol. 2308. CEUR Workshop Proceedings, CEUR Workshop Proceedings, Stuttgart, Germany, pp. 87–94, 2019, http://www.se-rwth.de/publications/Model-Based-Engineering-for-Avionics-Will-Specification-and-Formal-Verification-Based-on-Broys-Streams-Become-Feasible.pdf.

[KRR15] Kolassa, C.; Rendel, H.; Rumpe, B.: Evaluation of Variability Concepts for Simulink in the Automotive Domain. In: System Sciences Conference (HICSS'15). IEEE, pp. 5373–5382, 2015, http://www.se-rwth.de/publications/Evaluation-of-Variability-Concepts-for-Simulink-in-the-Automotive-Domain.pdf.

[Ku25] Kuelper, N. et al.: Integration of a Model-Based Systems Engineering Framework with Safety Assessment for Early Design Phases: A Case Study for Hydrogen-Based Aircraft Fuel System Architecting. Results in Engineering 25, p. 104249, 2025.

[Le16] Lee, E.: Fundamental Limits of Cyber-Physical Systems Modeling. ACM Transactions on Cyber-Physical Systems 1, pp. 1–26, 2016.

[NPW02] Nipkow, T.; Paulson, L. C.; Wenzel, M.: Isabelle/HOL: A proof assistant for Higher-Order Logic. Springer, Berlin et al., 2002.

[PR01] Philipps, J.; Rumpe, B.: Roots of Refactoring. In (Kilov, H. and Baclavski, K., ed.): Tenth OOPSLA Workshop on Behavioral Semantics. Tampa Bay, Florida, USA, October 15. Northeastern University, 2001, http://www.se-rwth.de/staff/rumpe/publications/Roots-of-Refactoring.pdf.

[PR94] Paech, B.; Rumpe, B.: A new Concept of Refinement used for Behaviour Modelling with Automata. In: Proceedings of the Industrial Benefit of Formal Methods (FME'94). LNCS 873, Springer, pp. 154–174, 1994, http://www.se-rwth.de/staff/rumpe/publications/A-new-Concept-of-Refinement-used-for-Behaviour-Modelling-with-Automata-FME.pdf.

[PR97] Philipps, J.; Rumpe, B.: Refinement of Information Flow Architectures. In (Hinchey, M., ed.): ICFEM'97 Proceedings. IEEE CS Press, Hiroshima, Japan, 1997, https://www.se-rwth.de/staff/rumpe/publications/Refinement-of-Information-Flow-Architectures.pdf.

[Ru99] Rumpe, B. et al.: UML + ROOM as a Standard ADL? In (Titsworth, F. M., ed.): Engineering of Complex Computer Systems, ICECCS'99 Proceedings. IEEE Computer Society, 1999, https://www.se-rwth.de/staff/rumpe/publications/UML-ROOM-as-a-Standard-ADL.pdf.

[SpesML23] Gesellschaft für Systems Engineering e. V.: SpesML Project Site, Accessed: 2023-10-24, 2023, https://spesml.github.io, accessed: 10/24/2023.

[Th12] The European Organisation for Civil Aviation Equipment: FORMAL METHOD SUPPLEMENT TO ED-12C AND ED-109A, Standard ED-216, EUROCAE, 2012.

[VZ14] Voss, S.; Zverlov, S.: Design Space Exploration in AutoFOCUS 3 – An Overview. In: IFIP First International Workshop on Design Space Exploration of Cyber-Physical Systems. 2014.

[WFM25] Weilkiens, T.; Forlingieri, M.; Molnár, V.: Next Generation MBPLE with SysML v2: Feature Modeling, Variability Modeling, and API Potentials. INCOSE International Symposium 35 (1), pp. 1589–1602, 2025.