Mechanisms for Model Consistency: A Comparative Analysis of Guideline Implementation in SysML v1 and SysML v2

1[st] Nassim Awabdy
Fachbereich 5
Aachen University of Applied Sciences
Aachen, Germany
nassim.awabdy@alumni.fh-aachen.de

***Abstract*—TODO**

***Index Terms*—cyber-physical systems, model-based systems engineering, sysml, modeling guidelines, model validation, model verification, systems engineering**

I. Introduction

Modern Cyber-Physical Systems (CPS) are technical systems that combine mechanical, electronic, and software subsystems with physical elements embedded in the real world. The development of CPSs is becoming increasingly complex and challenging, due to their interdisciplinary nature and the need to ensure seamless integration between their physical and computational components [2], [3].

Model-Based Systems Engineering (MBSE) is a methodology for the development and management such complex systems, that addresses issues arising from the complexity and interdisciplinary nature of CPS, and provides the agility required to adapt to changing requirements and technologies. MBSE incorporates a centralized system model as the primary source of information, throughout the system lifecycle [2]–[5], [7], [8].

SysML v1 has been widely adopted as the standard for modelling CPS and served as a key enabler MBSE. SysML v1 is a graphical, general purpose modelling language that is defined as an extension of the Unified Modeling Language (UML). Because it was built on top of UML, SysML v1 inherited several limitations from UML, that limited it's expressiveness and usability for CPS modelling. However, it still provided a solid foundation for specifying and analyzing a systems's behavior, structure and requirements [3], [4], [8].

The release of SysML v2 represents the next generation of the Systems Modeling Language, designed as an overhaul of SysML v1 that addresses its limitations and enhance the efficacy of MBSE practices. Unlike it's predecessor, SysML v2 is built upon the Kernel Modeling Language (KerML), this approach ensures that SysML v2 inherits a formal semantic foundation, that is crucial for enhanced precision and automation in MBSE workflows. [3], [4], [8], [13].

While these advancements introduced by SysML v2 are promising, the abstract nature of the language still presents challenges for ensuring consistent modeling practices across diverse engineering teams. Model inconsistencies creates a high risk of redundant effort, potential modeling errors, and lack of reuse of system elements, preventing them from being aggregated into a coherent overall system model. Therefore, mechanisms for implementing and enforcing modeling guidelines must evolve to leverage the native formal capabilities of SysML v2, that enable effective model verification processes throughout the development lifecycle [2]–[4], [13], [15].

Within the context of model analysis, a distinct differentiation between model validation and model verification is necessary. Following the ISO 15288 standard, this works adopts the following definitions:

- *Verification* is the process of proving that a design solution conforms to defined architectural and technical standards. This includes *requirements traceability* and *syntactic verification* [6].
- *Validation*, conversely, aims to prove that the system fulfills its business objectives and stakeholder requirements in its intended operational environment. This is achieved through *behavioral simulation* and the execution of *operational scenarios* [6].

This work addresses the following research question: **How do the mechanisms for implementing modeling guidelines differ between SysML v1 and SysML v2, and how do these differences impact the capabilities automated verification?**

To answer this, we conduct a comparative analysis of the underlying implementation mechanisms, specifically contrasting the profile-based constraints of SysML v1 with the metamodel-driven and formal semantic capabilities of SysML v2. We analyze the implications of this shift by examining specific scenarios where these mechanisms facilitate structural verification, rather than providing a holistic overview of all available commercial tools.

II. Theoretical Background

A. SysML v1 Foundations

SysML v1 is a graphical, general purpose modeling language that is widely recognized as the standard language for MBSE, serving as a foundational tool for specifying, analyzing, designing and verifying complex multidisciplinary systems [1], [4], [10].

SysML v1, adopted by the Object Management Group (OMG) in 2007, was essential in advancing MBSE practice by providing capabilities for formally capturing system requirements, structure, behavior, and parametric [8]. The language is defined as an extension of the Unified Modeling Language (UML), allowing it to adopt established modeling concepts [4].

SysML v1 organizes its modeling constructs into four main categories:

- **Structure**: This aspect is modeled using Block Definition Diagrams (BDDs) and Internal Block Diagrams (IBDs). BBDs are used to define components, interfaces, and relationships at black-box level, while IBDs offer a white-box perspective by outlining the internal structure of a single block [10].
- **Behavior**: SysML v1 provides several behavioral diagrams, including State Machine Diagrams, Sequence Diagrams, and Activity Diagrams [10], [17]. These diagrams capture distinct uses for modeling system behavior.
- **Requirements**: System requirements are specified in Requirement Diagrams, that largely rely on natural language representation, although they can be linked logically to other model elements [10], [13].
- **Parametric**: Parametric Diagrams define mathematical constraints and equations between system elements. They support preliminary calculations and analysis within the model, though they often require external tools for complex evaluations [10].

SysML v1 allows the restriction of modeling practices through UML profiling mechanism, enabling the construction of specialized extensions. The central element for customization is the *stereotype*, which functions as a distinct metaclass within UML [1]. Stereotypes enable the customization of existing metaclasses by associating them with specific properties and constraints, thereby tailoring the modeling language to meet domain-specific requirements [1], [16].

Object Constraint Language (OCL) is a expression language that enables formally defining rules, invariants and constraints on model elements, that must be satisfied for a model to be considered valid [15]. It identifies what a valid state is but does not provide the mechanism to calculate variables to reach that state.

B. SysML v2 Foundations

SysML v2 represents a major evolution over it's predecessor, having been engineered independently from UML to overcome the limitations inherited from it [4], [8]. It aims to enhance MBSE adoption and effectiveness by focusing on improving precision, expressiveness, consistency, usability, interoperability, and extensibility. [8] The foundation of SysML v2 is built upon a new general-purpose modeling language called the *Kernel Modeling Language (KerML)* [13].

The mechanism of KerML is built upon a hierarchical, three-layered architecture, successively progressing from general to specific constructs [14].

- The **Root Layer** establishes the essential syntactic scaffolding for constructing models. The main focus of this layer is to define organizational constructs, such as *Elements*, *Namespaces*, and *Relationships*, leaving out model-level semantic interpretation relative to the modeled system [13], [14].
- The **Core Layer** introduces the language's semantic foundation based on classification, defined in first-order logic axioms. The central primitive is *Type*, which is divided into *Classifiers*, *Features*. It also defines key relationships necessary for organizing classification hierarchies [13], [14].
- The **Kernel Layer** finalizes the language specification by adding specialized constructs used in common modeling applications, such as *Data Types*, *Classes*, *Structures*, and *Behaviors* [13], [14].

KerML achieves its consistent semantics through formal mathematical logic and library-based ontological modeling that maintain a precise interpretation of complex models. Since the semantics in the *Core Layer* are defined using first-order logic, a consistent basis for mathematical reasoning about models is established [13], [14].

For comprehensive concepts introduced in the *Kernel Layer*, KerML extends its semantics through the reuse of elements found in the *Kernel Semantic Library*. This library is itself expressed in KerML, meaning that all concepts in the language are ultimately grounded in the same formal semantic framework [13], [14].

SysML v2 introduces several key features for further enhancing modeling capabilities, accessibility, and tool integration.

- **Textual Notations**: In addition to graphical notation, SysML v2 features a standardized textual syntax that provides advantages for interoperability with external tools and exchange of models [13], [14].
- **Standardized API**: The language includes the new Systems Modeling API (SysML API), which enables full access to the model and general Model as Code workflows [13], [14].
- **Metadata Definitions**: In SysML v2, *Metadata Definitions* (`metadata def`) serve as the primary mechanism for annotating model elements with domain-specific semantics, analogous to Stereotypes in SysML v1. They define a specific schema (properties and constraints) that extends the language's metaclasses, enabling the creation of domain-specific modeling languages (DSMLs) [3], [13].
- **Constraints**: KerML provides the foundation for defining constraints through constructs such as *assert*, *invariant*, *requirement*. These KerML constructs enable the model to be treated as a mathematical system that can be analyzed and solved directly [15].

- **Cases**: SysML v2 introduced a generic *Case* construct which is essentially a calculation that can declare a subject and an objective to provide a formal and executable way to check model correctness and evaluate system properties. Two specialized cases are provided *Analysis Case* (Quantitative Analysis) and *Verification Case* (Qualitative Analysis), further enhancing model analysis capabilities [13].

C. Computational Solvers

Computational Solvers in MBSE are engines that automate the search for solutions within a "design space". They process a set of declarative constraints (mathematical and logical) to identify valid configurations, verify that requirements are met, and optimize system performance. The utility and implementation of solvers differ fundamentally between SysML v1 and SysML v2, due to the fundamental shift in language architecture.

III. Mechanisms for Guideline Implementation

This section analyzes the structural mechanisms used to define and enforce domain-specific modeling guidelines. The comparison reveals a shift from extrinsic constraints in SysML v1 to intrinsic semantic definitions in SysML v2.

A. SysML v1: Profile-Based Implementation

In SysML v1, the implementation of domain-specific guidelines is achieved through multiple layer approach. First, system engineers utilize the *Stereotypes* to build a custom profile that overlays the native SysML v1 metamodel, thereby introducing semantic labeling and domain specific terminology [1]. The application of SysML v1 elements is then restricted to a specific palette of stereotyped elements, ensuring that the model structure reflects domain-specific semantics rather than generic block definitions [1].

A concrete application of this mechanism is demonstrated by Beers et al. [1] in the development of a Domain-specific Modeling Language (DSML) for formal process description. To enforce the guideline that system functions must be standardized, the author extends the metaclass *CallBehaviorAction* to create a specific *Process Operator* stereotype. This ensures that an element cannot be designated as a *Process Operator* unless it inherits the specific meta-attributes defined by the stereotype.

While stereotypes provide semantic labeling, UML profiles are not expressive enough to represent constraints on the models [12]. Therefore, *Object Constraint Language (OCL)* is layered onto the profile to specify invariants, constraints and complex relationships between elements [1], [12]. Beers et al. [1] demonstrates this when enforcing the VDI/VDE 3682 standard that provides the rules to implement the "Product-Process-Resource" (PPR) concept, which mandates that production systems are modeled through the strict interconnection of the *Process*, the *Product*, and the *Resource* [1]. In their implementation, a "State" (e.g., a Product or Energy) cannot legally connect directly to another "State" without an intermediary process. Since standard SysML v1 syntax permits the connection of any two compatible nodes, the guideline is enforced through an OCL invariant attached to the *Flow* stereotype, that prevents two state-describing elements from being connected together [1].

B. SysML v2: Intrinsic Enforcement via Metamodel Definitions

SysML v2 implements guidelines intrinsically by allowing modelers to extend the language's ontology directly using KerML (Kernel Modeling Language). This removes the separation between the model and the rules found in v1.

Metadata Definitions is used to construct a domain-specific metamodel hierarchies directly within the language architecture by specializing standard KerML constructs [3].

1) Metamodel-Driven Guidelines: Boelsen et al. [3] demonstrates this by implement modeling guidelines for reusable mechanical system elements based on the *motego* methodology. The structural foundation is built by defining an abstract metadata definition for a general *solution*, which is set up as a specialization of the standard *SysML::PartDefinition* (see Fig. 1). From this abstract base, concrete domain-specific types *SolutionElement* and *SystemSolution* are derived [3].

The guidelines further mandate specific structural and behavioral components. The *ActiveSurface* and *ActiveSurfaceSet* are defined as specializations of *SysML::PartDefinition* to represent geometrical structure, while the *Material* is integrated as a reusable structural part within these surfaces.

Crucially, the behavioral logic is enforced by defining the *PhysicalEffect* not as a block, but as a specialization of the *SysML::ConstraintDefinition* meta type. Corresponding metadata definitions are also derived for *PartUsage* to enable the instantiation of these elements within the system model [3].

```
metadata def SolutionElementDef :> PartDefinition;
metadata def ActiveSurfaceSetDef :> PartDefinition;
metadata def ActiveSurfaceDef :> PartDefinition;

metadata def PhyEffectDef :> ConstraintDefinition;

metadata def SolutionElement :> PartUsage;
metadata def ActiveSurfaceSet :> PartUsage;
metadata def ActiveSurface :> PartUsage;
metadata def PhyEffect :> ConstraintUsage;

metadata def Material :> PartUsage;
```

Listing 1. Definition of the domain-specific metamodel for mechanical elements in SysML v2 (adapted from Boelsen et al. [3])

The enforcement of these guidelines is demonstrated in the modeling of a specific component, such as a "Lubricated Mechanical Line Rolling Contact". The component is instantiated using the *#SolutionElementDef* metadata definition (see Fig. 2). By using this specific definition, the modeler is guided to explicitly include the required sub-elements defined the in the meta model, specifically the *ActiveSurface* and *PhysicalEffect* [3]. Within this structure, the *ActiveSurface* integrates essential parameters via attribute definition and references the material using the *#Material* command [3]. Simultaneously, the *PhysicalEffect* is completed by adding input and output attributes and defining the calculation specification as expression, thereby ensuring that the relationship between the physical effect and the functional flows are strictly quantified accordingly [3].

```
#SolutionElementDef def LubMechLineRollingContact {
    #PhyEffect pe1 : SurfacePressure;
    #PhyEffect pe2 : CurvedCurvedKinematics {
        in omega : Real;
        out v_out : Real = r * omega;
    }

    #ActiveSurfaceSet ass : CylindricalLatSurfaces {
        #ActiveSurface as1 : CylindricLatSurface {
                attribute radius : Real;
        }
        #ActiveSurface as2 : CylindricLatSurface {
            #Material mat : Steel;
        }
    }

    bind pe2.radius = ass.as1.radius;
}
```

Listing 2. Implementation of a "Lubricated Mechanical Line Rolling Contact" using the enforced metadata definitions (adapted from Boelsen et al. [3])

2) Usage Definition Separation: In SysML v2, the language distinguishes between the *definition* of an element (Its "Type") and the *usage* of an element (Its "Instance") [3], [8].

- **Definition** defines the reusable template, including features, attributes, and constraints. This acts as the "Library" element [8].
- **Usage** represents the occurrence of that definition within the systems. It inherits features for the definition but can redefine them to adapt to the specific context [3], [8].

Boelsen et al. leverages this to enforce a standardized structure for mechanical system elements, to ensure that engineers cannot deviate from the required structure. Using this approach Boelsen et al. was able to enforce guidelines in two ways.

- **Structural Inheritance**: When an engineer uses a library element, they create a *Usage* defined by the *Definition*. Because the usage inherits from the *Definition*, it automatically includes all required internal structures (e.g., *Physical Effect*) mandated by the guideline [3].
- **Restricted Customization**: the guideline can dictate the the internal structure is defined once in the library (*Definition*). In the system model (*Usage*), the engineer interacts with the exposed parameters (e.g., input/output flows) but relies on the validated internal logic of the *Definition*.

Overall, this separation allows the guidelines to treat the *Definition* as the "Single Point of Truth" (SPOT) stored in libraries. The *Usage* are merely pointers to these definition. Thereby, preventing redundant modeling efforts and diverse formalization.

IV. Mechanisms for Model Verification

A. SysML v1

The native verification capabilities of SysML v1 are inherently limited due to its UML-based architecture. While stereotypes provide a fundamental layer of structural consistency, through restriction of available elements, they offer no intrinsic verification capabilities [1].

We discussed how OCL is layered onto profiles to define formal constraints that can be evaluated against the model, thereby enabling basic verification [1], [12]. However, OCL is declarative and static, which is excellent for checking "flags", but to be able to verify that a model conforms to defined technical standards (e.g., Power ¡ 5W), OCL alone is insufficient if the values are derived variables rather than static constants [12].

To bridge this gap, external solvers are required to perform mathematical verification by evaluating the constraints defined in OCL against the model's parameters and derived values [11], [12]. However, solvers cannot directly interpret SysML v1 models therefore transformers are required to transform the models into a format compatible with the solver (e.g., OWL, SMT-LIB) [11], [12].

A prominent example of this verification workflow is the approach by Lu et al., based on the Cloud Agility Baseline (CAB) model. In their work, the authors explicitly highlight the semantic limitations of native SysML v1 tools, noting that tools such as Cameo Enterprise Architecture often validate OCL constraints based solely on syntax. Consequently, the native tool reported that the verification was "successful" even when the model contained known inconsistencies.

To perform actual verification, their workflow required mapping the SysML Block Definition Diagrams, State Machine diagrams, and associated OCL invariants into the Web Ontology Language (OWL). This transformation enabled the use of the Pellet OWL reasoner to evaluate the logical consistency of the CAB model. Crucially, the external solver identified that the *Shipment* block was logically equivalent to "Nothing" (unsatisfiable) due to conflicting constraints in the state machine, a fatal design flaw that was undetectable within the native SysML v1 environment.

This case study exemplifies the extrinsic nature of SysML v1 verification, where the validity of a design cannot be determined by the modeling environment itself but is contingent upon the successful transformation to an external mathematical domain.

B. SysML v2

1) Formal Semantic Constraints: In this section we refer to the work of Ratzke et al. [15] that builds upon the capabilities of SysML v2 native constraints to provide a more generalized constraint evaluation mechanism, especially for early-phase system modeling [15]. Ratzke's work introduces range-based semantics that enable modelers to enforce guidelines regarding system variability and precision throughout three constraints:

- **oneOf**: This semantic assigns exactly one value from a specified range that satisfies the constraint, which can be used to represent variation (e.g., choosing a specific diameter for a part) [15].

- **anyOf**: This semantic allows any value within a range to satisfy the constraint, useful for approximations (e.g., tolerances or acceptable performance ranges) [15].
- **allOf**: This mandates that all contained values in a range must fulfill the constraint. This is applicable for operational envelopes (e.g., ensuring a system operates across an entire temperature range) [15].

To demonstrate the practical application of these constraints, a *Tank* part definition was modeled with attributes for width, height, and length (see Listing 3). These attributes leverage the *oneOf* operator, to define permissible dimensions with precise variability. The requirement *tankBigEnough* then enforces that the derived volume attribute falls within a specific range.

```
part def Tank {
    attribute width: ISO::Length = oneOf(10.0 .. 100.0) [cm];
    attribute height: ISO::Length = oneOf(1.0 .. 3.0) [m];
    attribute length: ISO::Length = oneOf(1.0 .. 1.2) [m];

    attribute volume: ISO::Volume = width * height * length;
}

requirement tankBigEnough {
    subject t: Tank;

    require t.volume == oneOf(1000.0 .. 2000.0) [L];
}
```

Listing 3. Implementation of constructive model analysis using Range-Based Semantics (adapted from Ratzke et al. [15])

When a restriction is imposed on the final volume, the solver utilizes constraint propagation to automatically reduce the valid domains of the dimensional attributes ($width \times height \times length$), ensuring the system model remains consistent with the specified design guidelines.

V. Discussion and Implications

A. Stereotypes vs. Metadata

- Stereotypes - Compliance is extrinsic. the model structure and the rule are separate entities (Profiles + OCL) [1].
- Metadata - Compliance is intrinsic. the rule is the definition of the element itself [3].

B. The Impact on Verification Capabilities

- SysML v1 - They ensure the diagram is readable and topologically correct, but they do not ensure the math is sound [4].
- Metadata - Because the guidelines enforce structure using KerML (First-Order Logic), the resulting models are natively compatible with formal solvers [11], [13]

C. Scalability via Library First Verification

- SysML v1 - Complex OCL constraints often have to be re-evaluated for every instance in the system, leading to performance bottlenecks in large models. [2], [15]
- SysML v2 - Separation of Definition (Type) and Usage (Instance) allows for a "Verify Once, Use Many" approach [3]

D. Solver Agnosticism and the Shift from Compliance Checking to Constructive Analysis

VI. Conclusion and Outlook

Acknowledgment

The author would like to express sincere gratitude to Professor Sebastian Voss for his supervision and guidance throughout the preparation of this work. His support in defining the research scope regarding SysML modeling guidelines was fundamental to the direction of this comparative analysis.

References

[1] L. Beers, H. Nabizada, M. Weigand, F. Gehlhoff, and A. Fay, "A sysml profile for the standardized description of processes during system development," in 2024 IEEE International Systems Conference (SysCon). IEEE, 2024, pp. 1–8.

[2] S. Bergemann, "Challenges in multi-view model consistency management for systems engineering," in Modellierung 2022 Satellite Events. Bonn: Gesellschaft f¨ur Informatik e.V., 2022, pp. 77–89.

[3] K. Boelsen, M. May, G. Jacobs, and et al., "Sysml v2 based modelling guidelines for mechanical system elements," Forsch Ingenieurwes, vol. 89, no. 60, 2025.

[4] E. Cibrián, J. Olivert-Iserte, C. Díez-Fenoy, R. Mendieta, J. Llorens, and J. M. Álvarez Rodríguez, "Ensuring semantic consistency in sysml v2 models through metamodel-driven validation," IEEE Access, vol. 13, pp. 121 444–121 457, 2025.

[5] S. Dehn, G. Jacobs, P. Höck, and G. Höpfner, "Enhancing model-based development with formalized requirements: integrating temporal logic and sysml v2 for comprehensive state and transition modeling," Forschung im Ingenieurwesen, vol. 89, no. 1, p. 53, 2025.

[6] Systems and software engineering — System life cycle processes, ISO/IEC/IEEE Standard 15288:2023, May 2023.

[7] P. Dukić, "Generating sysmlv2 models from structured natural language," 2025.

[8] S. Friedenthal, "Future directions for mbse with sysml v2." in MODELSWARD, 2023, pp. 5–9.

[9] D. D. Walden, G. J. Roedler, K. J. Forsberg, R. D. Hamelin, and T. M. Shortell, Eds., *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, 4th ed. Hoboken, NJ, USA: John Wiley & Sons, 2015.

[10] N. Jansen, J. Pfeiffer, B. Rumpe, D. Schmalzing, and A. Wortmann, "The language of sysml v2 under the magnifying glass." J. Object Technol., vol. 21, no. 3, pp. 3–1, 2022.

[11] H. Kausch, M. Pfeiffer, D. Raco, B. Rumpe, and A. Schweiger, "Model-driven development for functional correctness of avionics systems: a verification framework for sysml specifications," CEAS Aeronautical Journal, vol. 16, no. 1, pp. 33–48, 2025.

[12] S. Lu, A. Tazin, Y. Chen, M. M. Kokar, and J. Smith, "Detection of inconsistencies in SysML/OCL models using OWL reasoning," SN Computer Science, vol. 4, no. 175, 2023.

[13] V. Molnár, B. Graics, A. Vörös, S. Tonetta, L. Cristoforetti, G. Kimberly, P. Dyer, K. Giammarco, M. Koethe, J. Hester et al., "Towards the formal verification of sysml v2 models," in Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, 2024, pp. 1086–1095.

[14] Object Management Group, "Kernel Modeling Language (KerML), Version 1.0," Object Management Group, Specification formal/25-09-01, Sep. 2025. [Online]. Available: https://www.omg.org/spec/KerML/1.0/PDF

[15] A. Ratzke, J. Koch, and C. Grimm, "Modeling and analysis of system models with constraints in sysml v2," in 2025 20th Annual System of Systems Engineering Conference (SoSE), 2025, pp. 1–6.

[16] S. Raedler, J. Mangler, and S. Rinderle-Ma, "Model-driven engineering method to support the formalization of machine learning using sysml," arXiv preprint arXiv:2307.04495, 2023.

[17] U. Yildirim, F. Campean, A. Korsunovs, and A. Doikin, "Flow heuristics for functional modelling in model-based systems engineering," Proceedings of the Design Society, vol. 3, pp. 1895–1904, 2023.

[18] Zavada, G. Kulcsár, V. Molnár, and Horváth, "Towards a configurable verification and validation framework for critical cyber-physical systems," in 2025 IEEE 8th International Conference on Industrial Cyber-Physical Systems (ICPS), 2025, pp. 1–6.

[19] Z. Li, F. Faheem, and S. Husung, "Collaborative model-based systems engineering using dataspaces and sysml v2," Systems, vol. 12, no. 1, 2024. [Online]. Available: https://www.mdpi.com/2079-8954/12/1/18