Modeling and Analysis of System Models with Constraints in SysMLv2

Axel Ratzke, Johannes Koch, Christoph Grimm
Working Group Design of cyber-physical Systems
University of Kaiserslautern-Landau
Kaiserslauern, Germany
axel.ratzke|johannes.koch|cgrimm@rptu.de

2025 20th Annual System of Systems Engineering Conference (SoSE) | 979-8-3315-1535-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/SOSE66311.2025.11083847

***Abstract*—Requirements of technical systems are often given by inequations. In this paper we show how system model constraints can be described in SysMLv2 and how an underlying system of inequations can be used for analysis. We in particular formalize the problem, introduce different semantics for ranges, methods to support the specification of constraints, and give an overview of a proof-of-concept implementation with some examples.**

***Index Terms*—SysMLv2, requirements, effect chains, range-based semantics, solver, constraint propagation**

I. Introduction

In model-based systems engineering (MBSE), one uses models in particular for specifying the high-level architecture, functionality, use cases, requirements and constraints of technical systems. The models are typically used for documentation, exchange of models, and change management. Furthermore, initial calculations are done using e.g. parametric diagrams in SysML. For more complex calculations, more or less complex and specific tools are used. For example, Excel is popular for doing an initial rough analysis. AADL (Architecture Analysis & Design Language), as an example for a more complex framework, is a tool and language for the performance analysis of processor networks. In this paper,

- We give SysMLv2 models a more general semantics for model-level evaluation of constraints and ranges.
- We introduce different kinds of semantics of ranges and constraints.
- We demonstrate how constraints can be integrated into definitions and expressions through the example of SysMLv2 textual, and describe a proof-of-concept implementation along with several examples.

While we specifically refer to SysMLv2 and KerML, the underlying concepts and methods might also be applicable for other modeling languages.

II. Related work and preliminaries

A. Constraint Modeling in MBSE

Model-Based Systems Engineering (MBSE) has increasingly embraced formal methods to handle complex system requirements. In MBSE, constraints play a crucial role by defining boundaries within which system components must operate. Recent efforts emphasize using constraint-based programming to manage system variability and dependencies, as seen in configuration tools and constraint programming frameworks. For instance, [1] provides a foundational framework for constraint programming-based configuration, highlighting how constraints can enhance system flexibility and modularity without compromising on analysis rigor. Integrating constraints into MBSE tools offers a structured way to conduct early-phase system analyses, but challenges remain in extending these frameworks to high-level models as found in SysMLv2.

B. Existing Tools and constraint Programming

Dedicated tools like AADL and OCL (Object Constraint Language) have traditionally addressed specific aspects of constraint handling. AADL, for example, enables detailed performance analysis of hardware and software systems and has been valuable for tasks like processor network optimization. However, its complexity can limit applicability during initial MBSE phases. Model transformations between AADL and SysML(v2) like the one presented by [2] are typically used to exploit the advantages of both languages. Similarly, OCL provides a language for defining constraints within UML (Unified Modeling Language) models. Rajic and Sruk's 2024 review [3] examines the computational properties of OCL, underscoring its utility in defining precise model constraints. However, despite its robustness, OCL is not fully integrated into SysMLv2's native constraint-handling capabilities, necessitating either hybrid approaches or model transformation tools like the one presented by [4] to bridge gaps between SysML and formal verification tools.

C. Constraints and Consistency in SysML

Several academic and commercial modeling tools exist, that offer validation & verification capabilities. Capozucco researched what modeling situations in SysML need to be constrained and if those situations are caused by the language specification itself or the underlying tools like Eclipse Papyrus in [5]. It was found, that inconsistencies at tool level were indeed caused by the SysML specification and subsequently, EVL constraints were proposed to detect inconsistencies. At

This work was partially funded by the BMBF project KI4BoardNet No. 16ME0782

a deeper level, validation & verification of static model structures and behavior is realized by obtaining a symbolic model representation that is then interfaced with satisfiability solvers as demonstrated in [6] with OCL annotated UML models. With the increased research of the role of ontologies in MBSE, also verification & validation approaches started to exploit the new techniques. An exemplary approach to consistency checking with ontologies can be observed in [7], where Lu et al. map OCL annotated SysML models to OWL in order to use a specialized OWL reasoner to find inconsistencies.

D. Advances in SysMLv2 and Constraint Integration

During the design and development of SysMLv2, integrated support for expressions and constraints was identified as a key concept and innovation [8]. And with the introduction of a textual syntax and an abstract meta model, the language advanced from constraint properties and constraint blocks to more general constraint definitions and constraint usages. The most important change, regarding constraint integration, introduced with the new version is KerML, which contains the first-order logic based semantics to further build upon [9]. But Almeida et al. found that the model-theoretic semantics for first-order logic are only defined for the KerML core layer, supported by Kernel semantic library in the kernel layer. Therefore, they urge for a complete formalization of the core layer semantics [10]. However, KerML and the contained formal semantics already enable new tools to formally analyze models and offer support for tasks such as formal verification and automated reasoning [9].

E. Summary of Gaps and Contributions

Despite these advancements, there remains a lack of a unified approach for integrating constraint programming directly within SysML models. Existing studies suggest various methods for managing constraints and detecting inconsistencies, yet the need for domain-independent, flexible constraint handling remains largely unmet. This paper builds upon SysMLv2's native capabilities by proposing a proof-of-concept implementation that integrates a broader set of semantics, such as anyOf and oneOf, into SysMLv2's constraints framework. This approach seeks to fill a gap in the current literature by providing more generalized constraint evaluation within MBSE, particularly for early-phase system modeling. In SysMLv2, model-level evaluable expressions can be computed if values are bound to concrete values, either literal values directly from a model, or Metadata from e.g. a tool. We combine the concepts of constraint programming and configuration tools and integrate them deeply in SysMLv2. This provides a modeling platform with powerful and domain-independent analysis capabilities.

III. Constraint Modeling

In this section, we present how SysMLv2 supports constraint modeling, especially through KerML-based constructs, and detail the semantics and syntactic structures used to define constraints. We then introduce additional semantics for range evaluations, allowing for broader applicability of constraint analysis.

A. Constraints in SysML and KerML

SysMLv2 is built on top of KerML, which provides a foundation for defining constraints through constructs such as invariant, assert, and requirement. These constructs enable modelers to specify Boolean conditions that must hold for a model to be valid. For instance, invariants in KerML define mandatory conditions for a system's state, while assertions allow modelers to validate specific attributes or relationships at the model level. For example, we can model the environment temperature as:

```
feature temp: Temperature;
inv minTemp { temp ≥ 0.0 [K] }
```

This example ensures that the temperature value does not fall below absolute zero, a necessary physical constraint for accurate modeling in certain domains. SysMLv2's constraint language, therefore, supports the explicit validation of physical parameters through constructs like assert and requirement. Another example involves setting constraints for a device's drift attribute, which could be defined within a broader temperature range:

```
attribute temp: Temperature;
assert minTemp { temp ≥ 0.0 [K] }
part def Device {
  attribute drift: Real = f(temp, ... );
  ...
}

requirement tempRange {
  subject device: Device;
  require {
    device.drift < 0.01
  }
  require assume {
    temp ≥ -30 [°C] && temp ≤ 100 [°C]
  }
}
```

In this example, we define a requirement for the drift of a device under an assumed temperature range, showing how SysMLv2 enables conditional constraints that depend on environmental factors

B. Semantics of ranges and its use in expressions

One of the primary contributions of this paper is the introduction of extended semantics for handling range constraints within SysMLv2. These semantics include:

1) oneOf: This semantic assigns exactly one value from a specified list or range that meets the constraint, offering a choice to either an engineer or the solving process to choose the assignment according to further dependencies or a form of optimization. This semantic offers itself to depict variation. An exemplary use case for this scenario could be the different variations of a Part: $Diameter = oneOf(29[cm]..31[cm])$.

2) anyOf: This semantic is used to assign any value within a given range or list that satisfies a constraint. It is particularly useful for scenarios where multiple acceptable values or conditions might exist or where the exact value is of less importance than it laying between specified bounds. An exemplary use case for this scenario could be the result of an hardware dependent computation such as $\sqrt{2} = anyOf(1.414..1.415)$ or $\frac{Diameter}{\Pi} = anyOf(9.54[cm]..9.55[cm])$.

3) allOf: This semantic assigns a list or range, of which all contained values, have to fulfill the expressed constraint. A typical use case for this semantic could be to ensure the correct operation under a every anticipated possible value: $OperatingTemperature = allOf(-10.0[°C]..100.0[°C])$.

These semantics provide modelers with a flexible way to represent constraints that reflect real-world conditions more accurately than traditional binary constraints, such as in cases where system performance varies based on selected operational modes or component statuses.

C. Boundedness of Ranges

These presented semantics for ranges hinge upon being bounded for several reasons: Our first (technical) rationale was the representation of Reals. Reals cannot be expressed in a direct manner on computer systems, but need some technical representation like e.g. Doubles or Floating-point Numbers. These representations are dependent on the hardware and offer no true equality, but approximated values. Instead of e.g. checking the approximation $\sqrt{2} == 1.4142135623$, we propose checking the sound predicate $\sqrt{2} \in [1.4142135623..1.4142135624]$. Our second rationale is motivated through modeling praxis, where Engineers often use ranges with larger diameter to avoid over-specification or give tolerances and generally consider uncertainties and variations. A third, important, rationale was to enable the concept of Constraint Propagation (please refer to the Implementation chapter for an informal definition): Even strong bounding methods lose efficiency or outright fail on possibly infinite domains [11]. Our last rationale behind the decision is our assumption that using these semantics on unbounded ranges constitutes a modeling antipattern: Consider e.g. constraints of the form: $x = anyOf(-Inf..Inf)$, or $y = allOf(-Inf..Inf)$. These constraints lose meaning through their generality and we argue that respective constraints can be modeled in a more general way without the use of ranges (e.g. through simple type checking).

D. Semantic differences between Ranges and between representation of values

KerML defines a series of primitive Data Types including Reals, Integers, Booleans and Strings. Dependent on the Domain (which is determined by the underlying Data Type) as well as the boundedness, ranges differ in their semantics: A bounded Integer range contains only finite many Numbers which are discrete and precise, while even a bounded Real range represents infinite many continuous numbers, which may be subject to precision limitations in computer systems. For obvious reasons, we did not define any of the above semantics (anyOf, oneOf, allOf) for Booleans.

In accordance with the newly introduced range semantics, we change the representation of those Data Type definitions to suffice range-based semantics to:

- Booleans shall be represented by constraints or possible values;
  Boolean := A value from {true, false, unknown}
- Strings and Enumerations shall be represented by enumeration of all possible values;
- Integers shall be represented by constraints or explicit enumeration of possible values
  Integer := An unknown value from the Integers from $[lowerBound..upperBound]$ or from a List; where lowerBound and upperBound are Integers
- Reals shall be represented by constraints or explicit enumeration of possible values
  Real := An unknown value from the Integers from $[lowerBound..upperBound]$ or from a List; where lowerBound and upperBounds are e.g. Doubles

E. Semantics in definitions and inheritance

A strong feature of inheritance is given by the Liskov principle [12]. Behavior and expressions can be redefined without considering the Liskov principle. This makes sense because an evaluation thereof can hardly be analyzed in a general way. Hence, we introduce specific constraints that are evaluable in a static way and that are inherited and checked by the solver. For this purpose, we introduce the following functions that return a Boolean value:
isAnyOf(List), isAnyOf(Range): Checks if any of the values in a list or range is compatible with the base model constraint.
isOneOf(List), isOneOf(Range): Checks if exactly one value from the list or range is valid for the constraint.

```
attribute temp: Temperature;
assert minTemp { isAnyOf([0.0 [K] .. Inf [K↩
    ]]) }
part def Device {
  attribute drift: Real = f(temp, ... );
  ...
}

requirement tempRange {
  subject device: Device;
  require {
    device.drift = isAnyOf(0.0..0.009)
  }
  require assume {
    temp = isAllOf(-30.0[°C]..100[°C])
  }
}
```

Considering the Liskov principle, the following inheritance semantics arise for the newly defined range-based semantics:

1) anyOf: Specializations of the constraint can be between tighter bounds, as long as the actual value is still contained within the bounds. Considering the small example from the previous section, $\sqrt{2} = anyOf(1.4142..1.4143)$, would be

a sound specialization. A specialization that expands this range to $\sqrt{2} = anyOf(1.4..1.5)$ would violate the Liskov Substitution Principle, as the broader range includes values that are not valid approximations of $\sqrt{2}$ within the original bounds. This invalid specialization undermines the precision required by the original constraint, leading to inconsistencies when integrating with systems relying on the narrower bounds.

2) oneOf: Specializations choose a value to satisfy the constraint. Additionally, further constraints, that are connected through dependencies, can be considered to facilitate optimization tasks. Clearly, the range cannot be widened: $Diameter = oneOf(10[cm]..40[cm])$, would be an inconsistent specialization of the example from before.

3) allOf: Specializations of the constraint can be of wider bounds, but obviously not tighten the range. Continuing the example from the previous section, $OperatingTemperature = allOf(-15.0[°C]..110.0[°C])$ is a valid specialization, while $OperatingTemperature = allOf(0.0[°C]..100.0[°C])$ is not. Following these rules for the narrowing and widening of the constraints domain, we can use static evaluation of the Liskov Principle to check validity of specializations.

```
part def Vehicle {
  attribute maxSpeed: Real;
  assert safeSpeed {
    maxSpeed < 120
  }
}
part def Car specialices Vehicle {
  assert limitSpeed {
    isOneOf([90, 100, 110, maxSpeed])
  }
}
```

In this example, Car inherits the Vehicle constraints but further restricts the maxSpeed attribute to one of a limited set of values. This enables refined constraints to be applied at a more specific level while maintaining the general safety constraint defined in Vehicle. This proof-of-concept implementation demonstrates how the introduced semantics (anyOf and oneOf) can be applied within SysMLv2 models to improve constraint flexibility and evaluation. By embedding these semantics, we provide a mechanism for modelers to specify constraints in a more domain-independent way, expanding SysMLv2's applicability across various system domains.

IV. Implementation

This section consists of two parts: Subsection IV-A will go over the KerML implementation of our Ranges standard library package; and subsection IV-B will explain how these new semantics are integrated into a Constraint Satisfaction Problem.

A. Ranges standard library package

Our presented semantics are implemented in KerML as a standard library package. The current implementation (please note that this is a work in progress and can change at any time) is shown in Listing IV-A. It uses multiple inheritance to specialize an abstract range concept for a chosen primitive data type. Ranges for Quantities are also extended to feature SI Unit capabilities (implemented as a string). It relies upon our implementation of the data type Real, which we introduced in [13].

```
standard library package Ranges {

    //Abstract concept of a Range from which a Real is chosen
    datatyp InRange {
        feature min: ScalarValues::Real;
        feature max: ScalarValues::Real;
        inv selfIsInRange { (min ≤ self) and (self ≤ max) }
    }

    datatype AllInRange :> InRange;
    datatype IntegerInRange :> Scalarvalues::Integer, InRange;
    datatype RealInRange :> ScalarValues::Real, InRange;
    datatype AllIntegerRange :> ScalarValues::Integer, AllInRange;
    datatype AllRealInRange :> ScalarValues::Real, AllInRange;

    datatype QuantityInRange :> Inrange {
        feature unit: Scalarvalues::String;
    }
    datatype AllQuantityOf :> QuantityInRange;
}
```

B. Constraint Satisfaction

This section builds upon the approach of constructive model analysis introduced in [13]: for a more detailed explanation of the introduced concepts as well as definitions, we refer to that work. The basic idea is to facilitate model analysis capabilities of SysMLv2 models through tools and solvers by (re-)formulating the system model into a Constraint Satisfaction Problem (CSP). Informally a CSP can be defined as consisting of a finite set of variables, a finite set of respective variable domains and a finite set of constraints on those variables. Accordingly, a solution to given CSP is then an assignment of all variables from their respective domains, so that all constraints hold [14], [15].

Analysis of the model is then further supported by retaining all valid solutions to the CSP, while reducing the variables' domains as much as possible. This process of restricting variable domains to a minimum, such that all consistent solutions are contained is called Constraint Propagation [14], [15]. Following that approach, we first compile SysMLv2 models into aKerML abstract representation. Then we create instances of inherited variables and expressions. For this purpose, we clone features (and their owned elements) and constraints from general to specialized types. In the next step we create a list of all variables, their constraints and assertions using the KerML's Binding relationships for set-intersections. In a last step, Constraint propagation algorithms are employed to reduce the domains to a minimum. Constraint Propagation Algorithms are dependent on the respective variable's domain. We distinguish between Boolean, Integer and Real Domains and use specialized solvers accordingly: A (custom-built) CSP solver extracts necessary KerML relations from the abstract model and sets up the initial problem formulations for the different domain types, before processing and reducing Boolean and finite Integer domains. Infinite Integer domains will be reduced by interval arithmetic techniques. Arc-Consistency will be enforced until a fixed point is reached, before a LP-

solver is invoked with the computed conditions for the domain reduction on the real domain. The results are then fed back into the model.

On a last note regarding complexity: Solving CSPs is NP-hard in best cases and NP-complete in the rest [14]. To alleviate tools finding sound solutions fast and ensure scalability to industry-grade big problems, we allow over-approximations provided that no sound solution is discarded. As a tradeoff domains might not be reduced to a minimum and still contain values that are not part of any solution. We use affine arithmetic with linear constraints together with Chebyshev approximations for continuous domains. Additionally, we restrict constraint propagation to features and the static structure of the system model, excluding dynamic behaviors, to reduce the problem size further.

We now show how the CSP at the core of the approach is extended to represent the newly introduced semantics: As seen in the Ranges standard library package, we distinguish between Real-, Integer and String ranges. These primitive data types determine the variables domain. Reals reside within a continuous domain, while the realm of Integers is a discrete domain.

1) anyOf: In the context of a CSP anyOf is represented as a variable with a bounded number domain. We use the leaves affine arithmetic decision diagrams [16] which respect possible dependencies through contained noise symbols to obtain a LP-Problem formulation that can then be interfaced with a LP or SMT solver. Integers are handled by Interval Arithmetic methods to obtain an ILP formulation as input for a SMT solver.

2) oneOf: This semantic extends the one above so that the variables' chosen value not only obeys one ore more constraints but is also subject to some form of optimization. Similarly they extend our CSP to a conditional CSP [14]. In the discrete case we handle this by the introduction of if-then-else activation constraints. In the continuous case, we reformulate our LP as to optimize the looked for value.

3) allOf: For this semantic a given constraint must hold for every possible value. In the discrete case, we can construct a conditional CSP and activate all activation constraints.

V. Examples and Discussion

In this section we first give some more detailed examples from the implementation of our tool, before discussing our approach.

A. Examples

As a first example, we model a tank, as can be seen in Figure 1. The tank is defined by its attributes for width, height and length, which result in the final volume attribute. As one can see, the attributes are annotated with their SI Unit meter, respective centimeters. The attributes width, height and length are defined over ranges with the oneOf-Semantic, offering variability. The tank is constrained by a requirement, ensuring the tank has a certain minimal and maximal volume

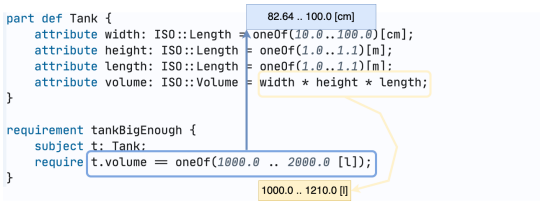

Fig. 1. Continuous constraint propagation with a Tank.

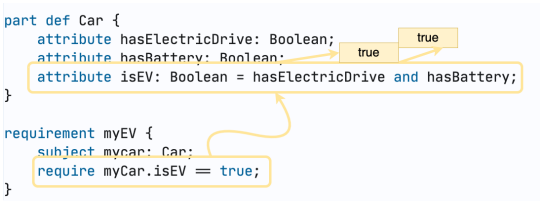

Fig. 2. Discrete constraint propagation with a car.

determined in liters, from which also exactly one value has to be chosen.

After constraint propagation, the domain of the requirement is reduced to a smaller Range, due to the evaluation of maximal values for the attributes. In a subsequent propagation step, then the domain of the tank's width is reduced accordingly, to assure that the requirement keeps satisfiable.

Another example, for a mere boolean case, we show in Figure 2: We model as simplified car, that can either be an electrical vehicle (EV) or an internal combustion engine vehicle, depicted by boolean attributes. Again, a requirement restricts the attributes domain: The car has to be an EV. In this example, constraint propagation, will reduce the boolean domain of the attribute *isEV* to true. Subsequently, the domain reduction is communicated to the respective dependencies and as a consequence, also the domains of the attributes *hasElectricDrive* and *hasBattery* are reduced also to true.

B. Discussion

After introducing the approach of enabling system model analysis through a constraint propagation on range-based semantics and detailing its implementation, this subsection provides a comprehensive analysis of the proposed constraint propagation framework, the necessary additional semantics and the perceived disadvantages regarding efficient solving.

1) Precise range-based Semantics: The introduction of the semantics of anyOf, oneOf and allOf together with a ranges library package enables the precise mapping of constraints to different problem formulations that can be used as input for different verification tools and solvers.

As laid out in Section IV-B, the semantics can be mapped to CSP, SMT, LP or ILP problem formulations, respectively which then can be analyzed by a specialized solver or tool, as we demonstrated in the examples of the previous subsection V-A.

2) Analysis of System Models through constraint propagation: As described in Section IV-B, range-based semantics

enable efficient reformulation of models into a CSP. Then domain reduction and network consistency algorithms facilitate constraint propagation accordingly. Through hierarchical (de-)composition of models, the constraint propagation can be extended to the whole system model.

This mechanism is expanded through the newly introduced semantics. The tank example in the previous subsection V-A demonstrates the hierarchical usage of the oneOf semantic as well as the constraint propagation in two directions across part definition and requirement.

3) NP-hardness: It is already mentioned in subsection IV-B, that solving CSPs is NP-hard in the best case. Also other problem classes such as SAT, SMT or ILP are known to be (NP-)hard to solve. As a consequence, constraint propagation is maybe not able to process some models in an efficient way.

This problem can be alleviated through the use of bounded ranges, to narrow the problem space, combined with over-approximations (as mentioned in subsection IV-B).

To further mitigate the problem, we propose to use constraint propagation as an analysis tool during early system development process (e.g. the system specification process), to reduce the problem size.

4) Inheritance & Redefinition: SysMLv2's complex inheritance mechanics in the form of specialization, subsetting and redefinition exacerbate the checks wether or not the Liskov principle still holds. Especially redefinition of expressions can break the principle. To offset this, specific constraints are introduced that check for the adherence to the Liskov Principle in a static way (as introduced in subsection III-E).

5) Contract-based Design & Solver agnosticism: The possibility to map the underlying abstract syntax model into different problem formulations of different frameworks facilitates solver agnosticism: Specific solvers can be chosen according to specific problem types (e.g. SAT-solver for problem instances that are limited to pure propositional logic). This enables the export to domain-specific languages for Validation & Verification with a specialized tool and subsequent re-import of results, which are then communicated through constraint propagation.

By providing such a framework for contract-based design through the propagation of solver-agnostic results, scalability and efficiency can be improved, further mitigating the issue of NP-hardness.

VI. Conclusion

We introduced the semantics of anyOf, oneOf and allOf to ranges and constraints. Subsequently, we demonstrated how these semantics map to different domain-specific problem formulations. We also demonstrated how these semantics can be used to integrate constraints into definitions and expressions.

To support the new semantics we proposed a changed representation of data type definitions to suffice range-based semantics. Also we analyzed inheritance and possible disadvantages of the new semantics with regards to the Liskov Principle and SysMLv2's redefinition capabilities and as a result proposed specific constraints which are evaluated statically.

By examples we have shown how the new semantics and their integration into constraints can be leveraged to analyze single models or composite system models. While we acknowledge that the proposed approach can lead to exponential runtime in the worst case, we present means such as over-approximations or bounded ranges to mitigate this potential problem.

References

[1] M. Queva, "A framework for constraint-programming based configuration," Ph.D. dissertation, Technical University of Denmark, 2011.

[2] J.-C. Roger and P. Dissaux, "Aadl modelling with sysml v2," *Ada Lett.*, vol. 43, no. 1, p. 42–45, Oct. 2023. [Online]. Available: https://doi.org/10.1145/3631483.3631486

[3] G. Rajić and V. Sruk, "Definitions and computational properties of ocl: A systematic review," *IEEE Access*, 2024.

[4] M. Kölbl, S. Leue, and H. Singh, "From sysml to model checkers via model transformation," in *Model Checking Software: 25th International Symposium, SPIN 2018, Malaga, Spain, June 20-22, 2018, Proceedings 25*. Springer, 2018, pp. 255–274.

[5] C. Capozucco, "Constraints for avoiding sysml model inconsistencies," 2019.

[6] N. Przigoda, M. Soeken, R. Wille, and R. Drechsler, "Verifying the structure and behavior in uml/ocl models using satisfiability solvers," *IET Cyber-Physical Systems: Theory & Applications*, vol. 1, no. 1, pp. 49–59, 2016.

[7] 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. 2, p. 175, 2023.

[8] H. P. de Koning, "Sysml version 2-final stretch," 2022.

[9] 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.

[10] J. P. A. Almeida, L. Ferreira Pires, G. Guizzardi, and G. Wagner, "An analysis of the semantic foundation of kerml and sysml v2," in *International Conference on Conceptual Modeling*. Springer, 2024, pp. 133–151.

[11] H. Schichl, A. Neumaier, M. C. Markót, and F. Domes, "On solving mixed-integer constraint satisfaction problems with unbounded variables," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings 10*. Springer, 2013, pp. 216–233.

[12] B. H. Liskov and J. M. Wing, "A behavioral notion of subtyping," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 6, pp. 1811–1841, 1994.

[13] A. Ratzke, S. Post, J. Koch, and C. Grimm, "Constructive model analysis of sysmlv2 models by constraint propagation," in *2024 19th Annual System of Systems Engineering Conference (SoSE)*. IEEE, 2024, pp. 239–244.

[14] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier, 2006.

[15] C. Lecoutre, *Constraint Networks: Targeting Simplicity for Techniques and Algorithms*. John Wiley & Sons, 2013.

[16] C. Zivkovic, C. Grimm, M. Olbrich, O. Scharf, and E. Barke, "Hierarchical verification of ams systems with affine arithmetic decision diagrams," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 10, pp. 1785–1798, 2018.