Received 10 June 2025, accepted 5 July 2025, date of publication 10 July 2025, date of current version 17 July 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3587786

APPLIED RESEARCH

Ensuring Semantic Consistency in SysML v2 Models Through Metamodel-Driven Validation

EDUARDO CIBRIÁN[1], JOSE OLIVERT-ISERTE[1], CARLOS DÍEZ-FENOY[2], ROY MENDIETA[2], JUAN LLORENS[1], AND JOSE MARÍA ÁLVAREZ-RODRÍGUEZ[1]
[1]Computer Science and Engineering Department, Carlos III University of Madrid, 28903 Madrid, Spain
[2]The Reuse Company, 28919 Leganés, Spain

Corresponding author: Eduardo Cibrián (ecibrian@inf.uc3m.es)

This work was supported by the Universidad Carlos III de Madrid CRUE-Madroño under Grant 2025.

ABSTRACT Model-Based Systems Engineering (MBSE) relies on formal models for system lifecycle management, supporting model coherence and efficient reuse of components. Modelling languages, particularly SysML v2, provide a standardized framework for complex system modelling, overcoming key limitations of earlier versions. Unlike SysML v1, which was a UML profile and inherited UML's complexities, SysML v2 is designed to better handle multi-disciplinary, large-scale, and emergent system behaviours. However, the validation of SysML v2 models is still an emerging area, and while some methods are beginning to surface, comprehensive and standardized validation approaches are not yet fully established. This may pose challenges to ensuring correctness and reliability in engineering workflows. This paper presents a systematic, metamodel-based method for validating SysML v2 models, utilising the SysML v2 metamodel as a formal specification. By defining validation rules derived from this metamodel, the method facilitates automated detection of structural and semantic inconsistencies. A practical case study validates the method on SysML v2 models of aerospace, automotive, and software development domains, demonstrating the systematic identification and resolution of errors. This research advances SysML v2 model validation, contributing to broader MBSE objectives by ensuring correct SysML v2 models for complex system development in multi-disciplinary environments.

A direct, practical, and proven example of exactly what your first two hypotheses are about

INDEX TERMS Formal methods, MBSE, metamodel, model validation, model-based systems engineering, software engineering, SysML v2.

I. INTRODUCTION

Model-Based Systems Engineering (MBSE) has become a cornerstone methodology for managing the increasing complexity of modern engineering systems [1]. By emphasising the use of formal models throughout the system lifecycle, MBSE supports the design, analysis, and verification of both logical and physical system representations, promoting consistency, traceability, and reusability across engineering processes. Within this context, the adoption of SysML (Systems Modeling Language) [2] has been particularly influential, offering a standardised framework for modelling complex systems across diverse domains.

The recent release of SysML v2 represents a significant advancement over its predecessor, SysML v1. However, it is important to note that SysML v2 is still under active development [3], and its final specification is evolving. While SysML v1 laid a solid foundation for MBSE, it was inherently constrained by its dependency on UML (Unified Modeling Language), which limited its expressiveness and scalability for complex systems engineering tasks [4]. SysML v2 overcomes this limitation by introducing a stand-alone architecture, decoupled from UML, with improved model semantics, richer view support, and enhanced mechanisms for maintaining consistency [5]. However, despite these improvements, challenges still remain in fully validating SysML v2 models. This validation process, especially in terms of detecting both syntactic and semantic issues, requires further development and refinement, which opens up several possibilities for research and improvements in the validation framework. A metamodel guides implementation, but different realizations may vary in structure or strictness, still offering useful

The associate editor coordinating the review of this manuscript and approving it for publication was Claudia Raibulet.

perspectives for validation and interpretation. A key insight is that these variations often stem from differences in how relationships, constraints, and dependencies are enforced across implementations. For instance, SysML v2's first-class relationships may be explicitly stored as separate elements in one implementation, while another may embed them within the related objects, leading to differences in how constraints like cyclic dependencies are validated. Similarly, while JSON-based representations allow for straightforward syntactic validation, graph-based approaches may provide more natural traversal of relationships but require additional logic to enforce semantic constraints [6].

SysML is a family of languages based on a common metamodel, offering graphical, textual, and API-based representations. To enable interoperability and tool integration, SysML v2 supports standardized serialization formats like JSON and XMI, ensuring consistent model exchange and traceability across tools. It has been found that syntactic errors are much easier to detect when working with JSON representations, as their structured format allows for straightforward parsing and rule-based verification. However, enforcing semantic constraints, such as ensuring consistency in cross-relationships between model elements, remains a significant challenge [7], [8]. These constraints often require a deeper understanding of the model's context and dependencies, making their implementation more complex and demanding sophisticated validation mechanisms. Ensuring the validity of models is essential to guarantee their accuracy, consistency, and compliance with both the language specification and domain-specific requirements.

This variability highlights the need for a precise validation process to ensure that implementations remain consistent with the metamodel's specifications and intended use. Model validation plays a significant role in maintaining the reliability of engineering workflows [9]. Errors or inconsistencies in SysML models can propagate through downstream processes, resulting in inefficiencies, design flaws, or system implementation failures [10]. Although SysML v2 introduces powerful modelling capabilities, the lack of standardised and automated validation methods poses a risk to the quality and utility of these models.

Recent literature has started to address this challenge. For example, Molnár et al. [11] explore formal verification techniques for SysML v2; Sultan and Apvrille [12] propose AI-driven consistency checks; and Thorburn et al. [13] examine lightweight approaches to integrate formal methods with SysML. These works demonstrate promising directions, yet they either focus narrowly on specific diagram types or rely heavily on external formalisms without providing a complete validation strategy grounded in the SysML v2 metamodel itself.

This paper proposes a systematic, metamodel-based method for validating SysML v2 models. The approach leverages the SysML v2 metamodel as a formal specification of the language's structure, semantics, and rules. By defining validation rules derived from this metamodel, the proposed method facilitates automated detection of structural and semantic inconsistencies, ensuring that models conform to general and domain-specific requirements.

To demonstrate the effectiveness of the proposed method, the paper presents a practical case study focused on validating SysML v2 models of aerospace, automotive, and software development domains. This case study illustrates how the metamodel-based approach can systematically identify and resolve errors, enhancing the reliability and readiness of SysML v2 models for integration into engineering workflows.

In summary, this paper proposes:

- A systematic validation framework based on the official SysML v2 metamodel, covering both syntax and semantics.
- A set of formalized validation rules, directly derived from the metamodel's structure and constraints.
- An automated tool capable of parsing JSON-based SysML v2 models and reporting syntactic and semantic violations.
- A comprehensive case study across aerospace, automotive, and software domains to evaluate the method's effectiveness and generality.

This research advances the state of the art in SysML v2 model validation by providing a robust and extensible method. The findings contribute to the broader objectives of MBSE by ensuring that SysML v2 models serve as accurate artefacts, supporting the efficient development of complex systems in multidisciplinary environments.

II. STATE OF THE ART

Model-Based Systems Engineering (MBSE) has established its position as a cornerstone methodology for managing the complexities inherent in modern engineering systems [14], [15]. Within the MBSE paradigm, the Systems Modeling Language (SysML) is recognized as a foundational standard for system modelling [16]. The recent release of SysML v2 (beta 2) marks a significant shift from its predecessor, SysML v1 [17], introducing a decoupled architecture, enhanced model semantics, richer view support, and improved consistency management [5]. These advancements are critical in addressing limitations previously imposed by SysML v1's reliance on the Unified Modeling Language (UML). However, the lack of robust validation methods is a barrier to realizing the full potential of SysML v2 in practical applications [3].

Model validation is a process to ensuring the precision, coherence, and conformity of models with both the language specification and specific domain requirements [18]. Flaws or contradictions within SysML models are liable to propagate through subsequent development stages, resulting in inefficiencies, design defects, and implementation failures. Although SysML v2 provides powerful modelling features, the lack of new tools implementing validation techniques is a notable challenge for the consistent use and quality assurance of these models [19]. This highlights a critical need for precise validation methods to ensure the fidelity of SysML v2 models in engineering practice.

Existing research related to model validation often focuses on specialized aspects of model verification [20], [21], including syntactic correctness and semantic consistency [22]. However, a comprehensive perspective that encompasses the entire model validation process is missing. Furthermore, many model validation techniques are constrained to specific modelling environments, making their adaptation across different tools or even versions of the same language challenging [23]. These limitations underscore the need for more generalizable and versatile validation techniques.

Related with SysML v2 validation, comprehensive methods are still under active development. While dedicated SysML v2 validation tools are not yet available, some existing platforms, such as SysIDE [24], aim to provide integrated environments for SysML v2 modelling, with a particular emphasis on its textual notation. SysIDE is an open-source tool designed for editing and analysing SysML v2 textual models, offering syntax analysis and integration into popular development environments like VS Code. While it facilitates structured model creation and validation at a syntactic level, its focus remains on textual notation rather than deeper semantic validation. Another relevant tool is SysON [25], a platform centred on model execution and behavioural simulation. It provides validation capabilities through simulation-based checks and runtime constraint verification. However, its primary strength lies in executing SysML v2 models rather than performing exhaustive semantic verification. However, its primary strength lies in executing SysML v2 models rather than performing exhaustive semantic verification. In contrast, Cameo Systems Modeler [26] is a commercial tool that supports SysML modelling with partial compatibility for SysML v2 through plug-ins. It offers both syntactic and limited semantic validation, including rule checking and consistency analysis. While it supports simulation and traceability, and its environment enables further extensions such as probabilistic reliability analysis [27], its semantic validation depends heavily on user-defined configurations and does not fully cover the depth of SysML v2 specifications.

Recent implementations such as SysMD provide a validation environment for SysML v2 models by integrating syntactic and semantic consistency checks, enabling partial simulations and the automatic verification of block relationships. In contrast, tools like SAVVS focus on property verification through formal analysis and test scenario generation, thereby facilitating early detection of specification inconsistencies and ensuring that SysML v2 models satisfy the defined constraints [11].

The comparison of SysML v2 validation tools, see Table 1, reveals varying degrees of support for syntactic and semantic validation. SysIDE focuses solely on syntactic validation, verifying the basic structure and format of models but lacking the ability to detect logical coherence or advanced constraints, which can lead to semantic issues. SysON, on the other hand, prioritizes semantic validation but performs limited syntactic checks, often missing format errors and relying heavily on prior configurations, which may result in undetected semantic inconsistencies. Cameo Systems Modeler offers both syntactic and partial semantic validation; however, its effectiveness is influenced by user configurations and may fail to detect format and structure inconsistencies, especially when handling complex relationships. Moreover, its semantic validation capabilities are constrained when dealing with intricate interactions, and its overall reliability depends on the user's expertise. In contrast, the proposed methodology provides comprehensive syntactic and semantic validation, identifying errors that conflict with the SysML v2 specification and aiming to ensure both structural and logical correctness of the models.

TABLE 1. Comparison of SysML v2 validation tools.

Tool	Validation level	Syntax verification	Semantic verification	Logical consistency	Advanced constraints
SysIDE	Syntactic	**Yes**	**No**	**No**	**No**
SysON	Semantic	**No**	**Yes**	**No**	**No**
Cameo Systems Modeller	Syntactic and partial Semantic	**Yes**	**Partial**	**No**	**No**
SysMD	Syntactic and Semantic	**Yes**	**Yes**	**Partial**	**No**
SAVVS	Semantic	**No**	**Yes**	**Yes**	**Yes**
Proposed Solution	Syntactic and Semantic	**Yes**	**Yes**	**Yes**	**Yes**

In the field of Model-Driven Engineering (MDE), the use of metamodels as formal language specifications is advocated [28]. Metamodels serve as blueprints for producing concrete models that adhere to a specified syntax and semantics [29]. However, within MDE, the focus has mainly been directed towards transforming model specifications into different formats or implementation languages. Limited attention has been given to leveraging metamodel capabilities for automated model validation. Furthermore, metamodels can also be used for semantic retrieval, enabling the search and extraction of information based on the meaning of model elements, not just their structure [30], [31]. Recent studies, as outlined in [32], have highlighted the importance of context and semantics within metamodels as areas requiring further exploration, especially for more robust validation.

Moreover, a critical gap exists in methodologies incorporating domain-specific validation rules. While general rules derived from a metamodel address structural and semantic inconsistencies, domain-specific rules are needed to enforce constraints and knowledge particular to a given engineering context [33]. These rules are essential for ensuring that models are not only syntactically and semantically valid but also relevant for the specific target application. Recent studies, such as the work presented in [34], underscore the necessity of domain-specific validation methods in Model-Based Engineering. The absence of a holistic approach capable of addressing both general and domain-specific validation requirements limits the potential of SysML v2 in complex engineering scenarios.

In response to these limitations, a metamodel-based method for SysML v2 model validation is presented in this paper. The approach utilizes the SysML v2 metamodel as a formal description of the language's structure, semantics, and rules. While this metamodel offers improved semantic clarity and integration capabilities, the realization of its full potential critically depends on its implementation and the availability of tools to validate models. A well-defined and implemented metamodel not only ensures compliance with the standard but also facilitates the practical adoption of SysML v2 by tool vendors and practitioners. Without such implementations, the transition to SysML v2 risks being theoretical, limiting its impact on engineering practices. Therefore, establishing an implementation of the SysML v2 metamodel and associated validation techniques is essential to bridge the gap between the specification and its practical application in industrial contexts.

Several companies have developed their own implementations of the foundational metamodel to facilitate its usage and integration. A potential inconsistency has been identified in the documentation of the metamodel published in [35]. Specifically, the attribute "aliasIds," present in multiple data types, is defined as a string with a cardinality of [0..1]. This contradicts the official OMG (Object Management Group) SysML v2 specification, where "aliasIds" is defined as a string with a cardinality of [0..*], allowing for multiple IDs to be associated with this attribute [3].This inconsistency underscores the importance of a consistent metamodel implementation and may impact the fidelity of model validation tools. Furthermore, this cardinality inconsistency directly affects model interoperability; models adhering to a more restrictive interpretation may not be fully compatible with those allowing multiple "aliasIds." This can lead to data loss or misinterpretation during model exchange, undermining seamless integration. Therefore, resolving such inconsistencies is crucial for enabling a reliable SysML v2 ecosystem.

Validation rules are derived from the metamodel to facilitate the automated detection of both structural and semantic inconsistencies. By doing so, this method aims to support the broader objectives of MBSE by ensuring that SysML v2 models serve as accurate and reliable artifacts, thereby bridging the gap between advancements in modelling languages and comprehensive validation techniques. This contribution aligns with current trends, which indicate a growing focus on combining metamodels with automated validation techniques [36]. The proposed method in this paper represents a step forward in ensuring the reliability and usability of SysML v2 models in real-world applications.

Based on the analysis presented in the state of the art, it is evident that there is a lack of comprehensive and systematic methodologies for the validation of SysML v2 models, particularly approaches that integrate both syntactic and semantic dimensions while accounting for variability in model representation and implementation.

This work introduces a metamodel-driven validation mechanism for SysML v2 models, which enables the automated detection of structural and semantic inconsistencies, while also providing a suggestion module to assist users in resolving identified issues. The main contributions of this work can be summarized as follows:

- Leveraging the official SysML v2 metamodel as a formal specification to derive validation rules, rather than relying solely on superficial or implementation-dependent syntax checks.
- Identifying inconsistencies arising from different representations of the same concept, such as variations in how relationships are encoded within models.
- Integrating an intelligent suggestion component, based on the Levenshtein distance, to offer corrective recommendations for syntactic errors and enhance user experience.
- Providing a clear separation between syntactic and semantic validation, enabling more precise, modular, and targeted feedback for model correction.

This contribution advances the current state of the art by offering a systematic, extensible, and metamodel-based validation framework for SysML v2, aligned with the principals of MBSE and ready for application in real-world industrial environments.

III. A METHODOLOGY TO VALIDATE SysML v2 MODELS

In this section, the validation process will be described focusing on the schemas extraction and the validation steps involved in the process itself. This process focuses on the beta 2 version of the SysML v2 metamodel.

A. THE FAMILY OF THE SysML v2 LANGUAGE

- Elements: this component forms the building blocks of a system model, representing different aspects of the system's structure, behaviour and properties. Additionally, each type of this component represents key modelling constructs, such as blocks, activities, interfaces and parts, enabling the creation of precise system models and modellers to encapsulate detailed system characteristics while maintaining modularity and reusability.
- Relationships: this component defines how elements are connected, interact or depend on one another, ensuring a cohesive and comprehensive representation of the system. It includes associations, dependencies,

or hierarchies, ensuring a cohesive representation of system interactions.

- Constraints: this component enforces rules that govern the structure, behaviour and properties of the system, ensuring that the model remains valid and consistent. There are several types of constraints such as logical, behavioural, structural, etc. By embedding these constraints, it provides a robust framework for modelling complex systems in a standardized and extensible manner.

1) TOOLS

This section will try to highlight the use of formal languages like Meta-Object Facility (MOF) and UML Profiles, which play a crucial role in defining and structuring the metamodels used in the language.

MOF [37] is a standard developed by OMG that is used to define metamodels. In the context of SysML v2, MOF provides a basis to describe the structure of metamodels in such a way that they can be understood consistently and accurately across different platforms. MOF not only defines the metamodels for modelling languages like SysML, but also for any other type of metamodel, allowing the creation of abstract models on which domain-specific languages can be built for different applications. MOF is key to ensuring interoperability between different modelling tools and for the automatic generation of metamodels that define properties and relationships within a system.

However, UML profiles extend UML by enabling its adaptation to specific domains through the definition of stereotypes, constraints, and new constructs. In SysML v2, UML profiles are used to customize and extend UML's capabilities to address the needs of modelling complex systems [38]. UML profiles provide a framework for adding new notations and semantics to a model, allowing engineers to define and customize the elements of a model based on the particular needs of a specific domain, such as systems engineering [5]. For example, a UML profile may include stereotypes to represent specific concepts in a system, such as components, interfaces or relationships.

Regarding modelling tools, platforms such as MagicDraw of Cameo Systems Modeler are widely used in the SysML v2 environment for creating and manipulating models [39]. These tools offer comprehensive graphical environments that enable users to create system diagrams intuitively, ensuring that the relationships and elements of the model adhere to the SysML v2 guidelines. In addition, these platforms provide advanced capabilities, such as model validation and verification, integration with other simulation tools, and therefore automatic generation of technical documentation.

2) STANDARDIZED API

One of the most important features of SysML v2 is the definition of an standardized API that allows users to access, manipulate and integrate different models. This is also a key difference between the two versions of SysML, in SysML v1 the access to the models was made via specific tools.

This API is designed as an open standard, which means that:

- The models created from a SysML v2 tool can be accessible with any tool that has the API integrated.
- The interoperability between different modelling, analysis, and simulation tools is encouraged.
- Integration into Model-Based Systems Engineering (MBSE) workflows that combine multiple platforms is facilitated.

It is important to mention that this API is based in well established standards such as OpenAPI and exchange formats like XMI and JSON. As will be discussed later, this API allows developers to create different scripts to validate the different models ensuring that it aligns perfectly with the base metamodel.

B. SCHEMA EXTRACTION

Before starting the validation process, there is a earlier step that is crucial for this process to be successful. This step means to extract all the schemas depending on the type of elements.[1] In this context, a schema acts as a formal specification for a SysML v2 element type. Each schema defines the valid attributes for a specific element, along with the expected data type or enumeration values for each attribute. These schemas are programmatically extracted from the SysML v2 metamodel documentation [3], and are crucial for the automated validation process. Specifically, a schema refers to structured representation of the elements in a system, defining their types. This structured representation of elements serves as a blueprint for validating and processing data within the system, ensuring that elements adhere to the specified format and structure. The schemas were developed by thoroughly analysing the SysML v2 metamodel documentation [3]. Based on this documentation, two distinct files were created. The first file encapsulates all information regarding the different types of elements supported by SysML. To process this data effectively, a parser has been developed based on the official documentation, extracts the attributes and their corresponding types for each element type. This parser systematically analyses the structure of SysML elements, ensuring that the extracted information aligns with the specifications defined in the standard. By automating this extraction, we can accurately map SysML elements, facilitating further processing, validation, and integration with external tools or analysis frameworks. The second file serves as an auxiliary resource and contains definitions for enumerated types (enums). Enums are a data type that represent a fixed set of named values, often used to define a finite collection of possible states or options, ensuring clarity and reducing potential errors in the representation of categorical data.

[1]Available in: https://github.com/edcisa/SysMLv2_Models_Validation/tree/main/Schema%20Extraction

C. VALIDATION METHOD

In this work, two validation methods are presented. The first is designed to validate the JSON file that will be entered into the SysML v2 Server provided by the OMG Community. This server functions as a database, enabling clients to query different aspects or parts of their SysML v2 projects. However, in order for the projects to be queried, they must first be uploaded to the server. This validator ensures that the JSON file being inserted into the server is free of errors related to the metamodel definitions, as discussed in the preceding sections. However, the second validator refers to a syntactic SysML v2 validator, which allows users to verify whether the syntax and semantics of their SysML v2 file are correct.

Before initiating the validation process, it is crucial to design a user-friendly interface that facilitates seamless file uploads. Once a user uploads the required file, its extension will be automatically detected, and the appropriate validation mechanism will be applied to verify the file's contents.

The validation process can be dissected in four different steps. An overview of the process is shown in Figure 1.

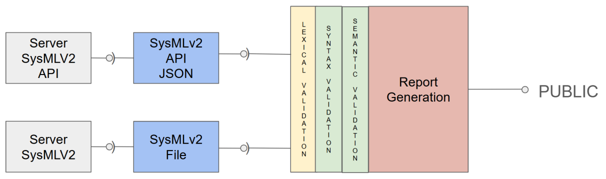


FIGURE 1. Overview of the validation process.

1) LEXICAL VALIDATION

Lexical validation is an important process while handling user-uploaded files through an interface, in this case, in the JSON and SysML v2 format. Upon receiving the file, this step ensures compatibility with the expected format by analysing its structure. JSON files are parsed to extract key-value pairs and hierarchical data, while SysML v2 files, often more complex, require specialized tools to interpret system modelling data effectively. Both of the validators treat the file as plain text during the validation process.

2) SYNTAX VALIDATION

Syntax validation refers to the process of ensuring that all the data comply with the predefined format or structural rules. This validation step is performed on both validators with its corresponding differences. There is an important component that facilitates the syntax and semantic validation of the SysML v2 files. This component is ANother Tool for Language Recognition (ANTLR) [40], a parser generator designed to read, process, execute, or translate structured text or binary files. It is widely used to create languages, tools, and frameworks. Based on a grammar,[2] ANTLR generates a parser capable of building and traversing parse trees.

[2]Available in: https://github.com/sireum/hamr-sysml-parser/blob/master/src/org/sireum/hamr/sysml/parser/SysMLv2.g4

In this validation process, ANTLR is used to automatically generate code derived from the grammar explained below, which is essential for subsequently validating input strings [41]. As a parser generator, ANTLR ensures syntactic correctness through syntax using lexical and syntactic analysis based on context-free grammars. Initially, it performs the lexical analysis, where an automatically generated lexer tokenizes the input text based on predefined rules of the grammar. Then, during parsing, a generated recursive descent parser applies grammar rules using LL(*) parsing. This technique enables ANTLR to efficiently detect syntax errors, enforce language constraints, and transform structured text into abstract syntax trees (ASTs) for further processing.

a: JSON SYNTAX VALIDATION

Recalling what was said at the beginning of this section, the JSON file comes from a SysML v2 API Server that is capable of this type of JSONs and will be inserted into another SysML v2 API Server. Also, as commented before, the base metamodel of SysML v2 extends from the SysML and KerML metamodels. In this section, a check to see if the corresponding attributes of the files comply with the format or types that are defined in those metamodels should be performed. When working with JSON files, the validator is designed to verify whether the file adheres to syntax rules by checking if the value of each attribute is either a string, a boolean, or a more complex data type defined by the base metamodel such as "Classifier." When an attribute is defined using a more complex data type, it typically establishes a relationship between two elements. Such attributes are often structured in a dictionary-like format, where each key corresponds to a specific identifier, and its associated value represents the linked data or elements. This approach facilitates the organization and retrieval of interconnected information in a systematic manner. The process is presented in Algorithm 1.

b: SysML v2 SYNTAX VALIDATION

When working with SysML v2 files, the process is slightly different. The grammar that defines SysML v2 language model is an agnostic grammar of the programming language in ANTLR format. When analysing the syntax of SysML v2 files, two rules will be followed:

- Lexic Rules (tokens): define the basic elements of the language, such as keywords, operators, identifiers, numbers, etc.
- Syntactic Rules: define how tokens are combined to form more complex structures.

Syntactic validation ensures that the input text complies with the grammatical rules defined in the ANTLR grammar (SysML v2 Lexer and SysML v2 Parser) [42]. This process involves transforming the input into a readable stream of characters, which serves as the basis for creating tokens. The lexer then divides the string into tokens based on the rules defined in the SysML v2 Lexer grammar file, with each token

Algorithm 1 Json Syntax Validation Process

```
Data: data, schema, enums
Result: errors list with invalid key-value pairs
begin
    Initialize an empty errors list;
    Extract keys from enums as enumKeys;
    foreach key-value pair in data do
        Retrieve the expected type;
        if the value does not match the expected type then
            Add key-value pair to errors;
        else if key is in enumKeys and value is not in enum list then
            Add key-value pair to errors;
        Handle missing or invalid keys by adding them to errors;
    end
end
```

representing a lexical unit such as keywords, identifiers, or symbols. These tokens are organized into a stream for the parser to process, ensuring that they are ordered and prepared for the next phase. The parser takes this token stream and verifies that it adheres to the syntactic rules of SysML v2 Parser, starting from a main rule (entryRuleRootNamespace) that defines the expected overall structure. A preconfigured error listener captures and returns syntax errors during parsing, allowing customized error reporting and handling. If no errors are found, a parse tree is generated that represents the hierarchical structure of the input text based on the grammar rules, which serves as the foundation for subsequent semantic validation.

3) SEMANTIC VALIDATION

Semantic validation refers to the process of ensuring that the meaning and intent behind data, models or processes align with their intended purpose, rules, or domain-specific semantics. It goes beyond syntactic validation to ensure that the content is logically consistent, contextually appropriate, and adheres to domain-specific rules or requirements. This process often involves rules, domain knowledge, or ontologies to determine whether the semantics of the subject meet the desired criteria. The semantic validation step is performed on both validators with its corresponding differences.

a: JSON SEMANTIC VALIDATION

To ensure that a JSON file adheres to domain-specific rules and has the appropriate context, the content is analysed to verify that all attributes align with the specified type of each element. This ensures that no attributes are incorrectly included or misplaced within the JSON structure. To delve deeper into the algorithm for this step, the process involves reading the type of the element from the input JSON and, based on this type, retrieving the corresponding schema that defines the valid attributes for that element. Subsequently, the attributes present in the JSON are analysed to verify that all of them are defined in the retrieved schema. It is important to note that the JSON is not required to contain all the attributes specified in the schema, given that the server would create these attributes as empty values when the JSON is inserted; the goal is to ensure that no attributes are incorrectly included for the given type of element. In addition, when an attribute of an enumeration type is encountered during the process, a validation is performed to ensure that the corresponding value is included within the predefined set of permissible values. This ensures consistency and prevents invalid assignments. The process is presented in Algorithm 2.

Algorithm 2 Json Semantic Validation Process

```
Data: data, schema, enums
Result: errors list with mismatched keys or invalid values
begin
    Initialize an empty errors list;
    foreach key-value pair in data do
        if key is not found in schema then
            Add key to errors;
        end
        else
            if the type is an enum then
                Validate against possible values in enums;
            end
        end
    end
end
```

As observed in the preceding algorithm, this process is relatively straightforward; however, it is essential to ensure that the content of the JSON file aligns with and adheres to the domain-specific rules defined by the underlying metamodel.

b: SysML v2 SEMANTIC SYNTAX

In the case of SysML v2 files, the tree returned by the syntactic validator is needed in order to perform this step, to analyse logic relationships, types, and consistencies in the SysML v2 model. ANTLR allows the use of listeners or visitors to implement semantic validation logic, which are registered to act on specific nodes of the parse tree. The previously described parser validates the structure and rejects anything that does not match the grammar rules, which constitutes syntactic validation, even if the rules seem to imply some "logical meaning" (e.g., that a requirement contains a subject). However, certain aspects cannot be handled by grammar alone, as they fall outside the strict definition of "structure." These correspond to semantic validation, such as ensuring that an identifier is unique within a set of requirements or verifying that a requirement references previously defined elements. These rules cannot be directly expressed in the grammar because they depend not only on the structure but also on the content and context. Therefore, it is necessary to traverse the tree generated and returned by the parser to apply this type of semantic validation.

4) ON THE FLY FIXING METHOD

The final stage of the validation process involves the generation of reports, during which the errors identified throughout the process are displayed in the User Interface (UI). As illustrated in Figure 1, the JSON and SysML v2 files are expected to be generated by servers capable of accurately producing these file types. Users should not be able to manually create or alter these files.

Internally, errors are collected during both the syntax and semantic validation phases. Even if an error is detected during the initial syntax validation step (e.g. An attribute is called "aliasIds" but "aliasId" was written instead), the process will continue with further validation to provide more comprehensive and precise information regarding the errors. This approach ensures a more thorough evaluation, offering valuable insights into the nature of the issues encountered. Additionally, a suggestion component has been developed to identify errors and provide users with potential solutions. This component leverages the Levenshtein distance [43], [44] that is used for spelling corrections because it quantifies the similarity between two strings by measuring the minimum number of edits —insertions, deletions, or substitutions— needed to transform one string into another. This metric is particularly useful for identifying the closest correct word in a dictionary or corpus when a user inputs a misspelled word. Since most spelling errors are the result of minor modifications, such as typos, omitted letters, or incorrect substitutions, the Levenshtein distance helps determine the most probable intended word by selecting the one with the smallest edit distance. This makes it a reliable and efficient method for automated spelling correction systems.

$$D(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0 \\ \min \begin{pmatrix} \text{D(i-1, j) + 1,} \\ \text{D(i, j-1) + 1,} \\ \text{D(i-1, j-1) + cost(i, j)} \end{pmatrix} & \text{otherwise.} \end{cases} \tag{1}$$

Being $cost(i,j)$ as it follows:

$$cost(i,j) = \begin{cases} 0 & \text{if } a[i] = b[j] \\ 1 & \text{if } a[i] \neq b[j] \end{cases} \tag{2}$$

The Formula 1 of the Levenshtein distance describes the process followed in the suggestions component. The base case ($min(i,j) = 0$) refers to the case where one of the strings is empty. This reflects the need to perform only insertions/deletions to transform one into the other. The recursive case refers to the case where no strings are empty, and the distance is computed by taking the minimum of the three operations.

An important component of the formula is the cost function cost(i,j) 2, which evaluates whether the characters at position i in the first string and j in the second string are equal. If they are the same, the cost is 0, indicating no substitution is needed; otherwise, the cost is 1, reflecting the need for a substitution. This ensures that only necessary changes are counted when computing the distance and that identical characters are preserved in the transformation.

When a syntactic validation process detects an error in one of the attributes of an element, the attribute in question is compared against all attributes within the schema using this metric. Each attribute is compared character by character using this formula. The computed $D(i,j)$ 1 reflects the number of transformations required to make the input attribute identical to a schema attribute. After all comparisons, the schema attribute with the smallest $D(i,j)$ 1 is chosen as a suggestion, since it has the highest similarity (requiring the fewest changes).

After evaluating the similarity of the attribute with each schema attribute, the suggestion provided corresponds to the schema attribute with the smallest Levenshtein distance. This indicates that the suggested attribute requires the fewest transformations to match the input attribute, making it the most likely intended match.

In summary, within the user interface, when an issue is detected, the system highlights the error and provides a corresponding suggestion to the user. This feedback mechanism is designed to enhance the user experience by not only identifying potential mistakes but also offering actionable recommendations to correct them. Using the Levenshtein distance, it is certain that the proposed correction is the closest match to the input, minimizing the number of transformations required.

By presenting both the error and its resolution in a clear and concise manner, the system streamlines the error-correction process, reducing user frustration, and improving overall efficiency.

IV. A CASE STUDY: SysML v2 VALIDATOR

In the following section, an evaluation of the implementation of the SysML v2 validation process[3] is analysed, focusing on its effectiveness, practical challenges, and alignment with the established standards and objectives. The validation process is applied to models generated using the official SysML v2 Implementation,[4] ensuring consistency with the current metamodel and language specifications.

A. RESEARCH DESIGN

As previously discussed, the validation process consists of several critical components, including file reading, syntax and semantic validations, and report generation. To ensure

[3]Available in: https://github.com/edcisa/SysMLv2_Models_Validation/tree/main/

[4]Available in: https://github.com/Systems-Modeling/SysML-v2-Release/tree/master/sysml/src/examples/Simple%20Tests

a thorough and accurate research design for evaluating the Validator, a series of well-defined stages has been established.

1) MODEL GENERATION

The Object Management Group (OMG) offers different SysML files in its official repository, known as SysML-v2-Release,[5,6] which serve as simple test cases[7] for users. This repository not only provides essential resources for understanding and experimenting with SysML v2 but also includes a feature to transform these SysML files into JSON format. This transformation capability is particularly significant as it enables users to interact with and test the SysML v2 API functionality, ensuring compatibility and ease of integration with different tools and workflows.

Furthermore, the repository encompasses a diverse set of domains, including aerospace, automotive, software development, and industrial automation, highlighting the adaptability of SysML v2 to different engineering challenges. This variety arises from the need to model complex systems across disciplines that require analytical specification, verification, and validation processes. For example, aerospace systems demand precise modelling of avionics, safety constraints, and mission-critical operations, whereas automotive applications focus on cyber-physical integration, embedded control systems, and independent functionalities. Similarly, software development leverages SysML v2 for requirements engineering, architecture specification, and model-driven development workflows. Additionally, the ability to convert SysML models into JSON facilitates seamless integration into software pipelines, enabling automated analysis, interoperability with other modelling tools, and efficient processing for performance and scalability testing in real-world applications. This capability is particularly valuable for industries that rely on simulation-driven design and consistent validation of evolving system architectures.

Additionally, the repository offers models with varying levels of complexity, ranging from simple examples with minimal elements to highly intricate models representing large-scale systems. This diversity in complexity is invaluable when analysing the validator, as it allows for testing its performance and functionality under different conditions. Simple models help validate the basic correctness and reliability of the API, while more complex models provide insights into scalability, efficiency, and the system's ability to handle intricate interdependencies. This range ensures a comprehensive evaluation of the validator, covering both fundamental operations and edge cases encountered in real-world applications.

This dual functionality, providing test models and enabling their transformation into JSON, makes the repository an indispensable resource for developers and engineers working on advanced system modelling and API testing. It serves not only as a learning tool but also as a foundation for building robust system modelling solutions.

A total of 25 models were randomly selected, as presented in Table 2. These models encompass different domains, as previously discussed, and are represented in the corresponding JSON files. Additionally, the models exhibit varying levels of complexity, which correlate with differences in the lengths of the JSON files.

It is important to note that the purpose of this validator differs from that of other tools. While some tools, such as Cameo Systems Modeler [26] or Capella [45], might directly load a SysML v2 model and prevent loading if it contains errors, effectively flagging format issues, this validator aims to provide a more detailed and comprehensive analysis of the model's structure and semantics. It goes beyond identifying whether a model can be loaded, analysing deeper into the specific inconsistencies and potential issues within the model.

TABLE 2. Model descriptions.

Model	Description
M_1	Simple model for importing an alias
M_2	Logical and Physical system decomposition
M_3	Automation Domain Model for vehicle dynamics computation
M_4	Text Annotation for analysis definition
M_5	Simple model for testing the counting capability
M_6	Mass Decomposition of vehicles
M_7	Definition of a Camera model
M_8	Modelling Cause and Effect
M_9	Comments inside a model
M_{10}	Modelling Conjugation and Connectivity
M_{11}	Modelling of Dynamics from MRU
M_{12}	Definition of Mass requirement
M_{13}	Modelling of battery life of a medical device
M_{14}	Modelling of the action of taking a picture
M_{15}	Definition of a Transmission Model
M_{16}	Modelling of the Thermodynamics of a vehicle
M_{17}	Modelling of a vehicle for demonstration propose
M_{18}	Example Vehicle definition model
M_{19}	Data of vehicles individually
M_{20}	Usages of elements from the vehicle definitions model
M_{21}	Simple model for testing actions
M_{22}	Modelling of Axle Assembly
M_{23}	Modelling of an individual
M_{24}	Modelling of a room
M_{25}	Example of a derivation in the Automation Domain

2) ERROR GENERATION

For the proposed case study, different errors have been deliberately introduced into the JSON files. The generation of these errors follows specific criteria. First, the errors are categorized into two distinct groups.

- Syntax Errors: these errors occur when the JSON structure is incorrect, regardless of the type of element being processed.
- Semantic Errors: these errors occur when the JSON structure is syntactically correct but the content does not conform to the intended meaning or constraints.

[5]Available in: https://github.com/Systems-Modeling/SysML-v2-API-Services

[6]Available in: https://github.com/Systems-Modeling/SysML-v2-Release

[7]Available in: https://github.com/Systems-Modeling/SysML-v2-Release/tree/master/sysml/src/examples/Simple%20Tests

Each type of error is introduced with a distinct purpose in JSON validation. On one hand, syntax errors are introduced to assess the system's or the parser's ability to detect and handle malformed JSON structures. Such errors can include misplaced commas, missing or improperly nested braces or brackets, and incorrect use of quotation marks as seen in preceding sections. Testing these errors helps verify the robustness of the syntax analyser and ensures that only well-formed inputs are processed, preventing unexpected failures or system vulnerabilities. Additionally, these errors demonstrate that even if a JSON file has been processed by a parser without immediately throwing an exception, it does not necessarily mean it is well-formed or fully interpretable. Some parsers may tolerate certain irregularities or make assumptions about the structure, which can lead to unintended behaviour. On the other hand, semantic errors are introduced to verify whether the system correctly validates the consistency and meaning of data within a syntactically correct JSON structure. Such errors can include values outside an allowed range, incorrect data types, or missing required attributes. Testing these errors ensures that the data meets the application's requirements and that logical validation mechanisms function properly, preventing misinterpretations or inconsistencies during processing.

With the different error categories defined, it is important to highlight that errors have been introduced randomly across the different models. As previously mentioned, the starting point consists of the JSON files generated through the transformation of SysML files into JSON by the Official Implementation. These JSON files were initially processed by the validation Module without detecting any errors. Out of the 25 available models, 20 were randomly selected, and errors were deliberately inserted into them.[8]

The randomly generated errors are adhered to these predefined criteria. Specifically, syntactic errors include incorrect tags, missing mandatory attributes, invalid connections (attributes not allowed in the given element), and ID inconsistencies. Meanwhile, semantic errors involve invalid values (e.g., mismatched data types such as string vs. integer, or incorrect enum values), values that are invalid but still parsed as strings (e.g., 0 vs. "0"), unused elements, and recursive references. Another important semantic error involves ensuring that all references created within an element of a model exist within the same model. This issue arises when an element references an entity that is either missing or belongs to a different model, leading to inconsistencies and potential failures in data integrity. Proper validation is required to verify that all referenced elements are present within the same model and that cross-references comply with the expected constraints. Failure to enforce this verification can result in incomplete or incorrect interpretations of the model's structure and relationships.

[8] Available in: https://github.com/edcisa/SysMLv2_Models_Validation/tree/main/SysMLAPIOM/Jsons/SysML%20Models

The validation module is capable of detecting some of these errors, particularly those related to structural inconsistencies and type mismatches. However, certain issues, such as subtle semantic inconsistencies or recursion in references, remain undetected because of the complexity of their validation.

B. ANALYSIS OF RESULTS

For the purpose of the evaluation of the validation process, different metrics have been analysed. With the observation of the results in those metrics, different conclusions of the efficiency and consistency of the validation process can be extracted. On one hand, the SyE (Syntax Errors), DSyE (Detected Syntax Errors) and SyE Ratio materialize the performance of the syntax validation steps. On the other hand, SeE (Semantic Errors), DSeE (Detected Semantic Errors) and SeE Ratio materialize the performance of the Semantic validation steps. All of these metrics provide a comprehensive assessment of the validation process's ability to detect both types of errors across different models, and the results can be shown in Table 3 and Table 4.

TABLE 3. SyE results of the validation process.

Model	SyE	DSyE	SyE Precision (%)
M_1	8	8	100.00
M_2	7	7	100.00
M_3	12	12	100.00
M_4	11	11	100.00
M_5	10	10	100.00
M_6	9	9	100.00
M_7	11	11	100.00
M_8	11	11	100.00
M_9	11	9	81.82
M_{10}	11	9	81.82
M_{11}	11	11	100.00
M_{12}	9	9	100.00
M_{13}	11	11	100.00
M_{14}	12	12	100.00
M_{15}	5	5	100.00
M_{16}	9	9	100.00
M_{17}	12	12	100.00
M_{18}	10	10	100.00
M_{19}	13	13	100.00
M_{20}	9	9	100.00
M_{21}	0	0	100.00
M_{22}	0	0	100.00
M_{23}	0	0	100.00
M_{24}	0	0	100.00
M_{25}	0	0	100.00
Total	202	198	98.02

A total of 428 errors have been randomly inserted in the 25 models available, alternating between syntax and semantic errors. A global measure can be extracted from calculating the mean between the both ratios, achieving a total of 97% of the errors detected, illustrating a high level of efficiency and consistency in the validation process.

As commented before, 5 of those models did not have any errors and the implementation has been able to successfully validate all of them, achieving a perfect score and parsing all the elements that those models have (from M_{20} to M_{25}).

TABLE 4. SeE results of the validation process.

Model	SeE	DSeE	SeE Precision (%)
M_1	13	12	92.31
M_2	15	14	93.33
M_3	8	8	100.00
M_4	14	14	100.00
M_5	12	11	91.67
M_6	13	12	92.31
M_7	15	14	93.33
M_8	6	6	100.00
M_9	14	12	85.71
M_{10}	11	11	100.00
M_{11}	11	11	100.00
M_{12}	14	13	92.86
M_{13}	8	7	87.50
M_{14}	11	11	100.00
M_{15}	5	5	100.00
M_{16}	12	11	91.67
M_{17}	12	12	100.00
M_{18}	11	11	100.00
M_{19}	11	11	100.00
M_{20}	10	10	100.00
M_{21}	0	0	100.00
M_{22}	0	0	100.00
M_{23}	0	0	100.00
M_{24}	0	0	100.00
M_{25}	0	0	100.00
Total	226	216	95.58

In addition to the accuracy-based metrics, the efficiency of the validation process has also been evaluated in terms of execution time. Figure 2 presents the evolution of the validation time as a function of the number of models present in each file. For this purpose, synthetic files have been created including three more models each time, just to evaluate the execution time.

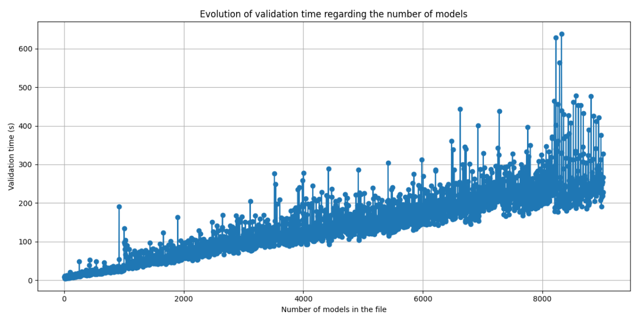


FIGURE 2. Validation time analysis.

The trend observed in the figure suggests a clear correlation between the number of models and the time required for validation. Although the growth is not strictly linear, showing a progressively increasing dispersion, the average validation time shows a steady upward trend as more models are included.

This behaviour highlights two key observations. First, the validation system scales reasonably well for smaller numbers of models, with execution times staying within acceptable bounds. Second, as the number of models approaches higher values (above 8000), the variance in validation time increases significantly, suggesting that certain models or interactions between models may introduce additional computational complexity.

Despite this variability, the system demonstrates robust performance for most instances, with the majority of the validation times remaining under 300 seconds even in the upper range. The outliers that exceed this threshold may be attributed to specific structural characteristics of the models being validated or the presence of particularly complex semantic constructs.

Overall, the validation time analysis, in conjunction with the error detection metrics, supports the conclusion that the validation pipeline is not only accurate but also reasonably efficient and scalable across a wide range of use cases.

One thing that is worth-mentioning is the slight difference between the SyE and SeE Ratios. As discussed in the preceding section, the validation process has no problem detecting errors related with structural inconsistencies and type mismatches, SyE. As a result of this, the score of both ratios slightly differ, obtaining a higher, almost perfect, score in the SyE Ratio. Delving into the Syntax Errors that the Validator is not able to detect, all of them are related to having the same attributes multiple times in an specific element. Related to SeE, the validation process has achieved a decent score, slightly lower than SyE, but now with robust handling of all previously problematic cases. Furthermore, the semantic rule set has been significantly extended. First, cardinality constraints are now enforced by traversing the model structure and applying rule-based checks derived from the metamodel specifications. Second, ID validation has been improved: the system now checks not only the syntactic validity of UUIDs but also ensures the uniqueness across the model and subcomponents. Previously, if the UUID followed a specific hexadecimal pattern, it used to pass validation even if it was meaningless (e.g. $fx8 - fx4 - fx4 - fx4 - fx12$). Third, we implemented cross-model reference validation by creating an internal index of all element identifiers during parsing, allowing the validator to detect unresolved references and ensure that all dependencies between model fragments are valid and consistent. These additions contribute to a more comprehensive and robust semantic validation pipeline, making the system more aligned with the needs of large-scale SysML v2 model verification.

However, it is important to note that a direct comparison between the implemented tool and existing tools analysed in previous sections, such as SysIDE, SysON, SysMD and SAVVS, was not feasible. These tools either do not perform direct and comprehensive model validation and verification, or they lack the capability to explicitly report the number of detected syntactic and semantic errors. Furthermore, some of them are still under active development or are not openly accessible, which prevents a uniform and reproducible evaluation. As a result, it is not possible to conduct a fair or meaningful comparison with the proposed validation methodology.

C. SUGGESTION GENERATION

As commented in section 4.2.4, a Suggestion Module has been developed to facilitate the end user different suggestions to the errors that the Validator has extracted. Nevertheless, it is important to mention that this Suggestion Module only applies to Syntax Errors, more precisely clerical errors in the attributes of a given entity.

The purpose of the proposed validator is helping the end user to understand better SysML v2 models without having an extremely technical background. By offering these suggestions, the system not only reduces the likelihood of recurring errors but also improves the overall modelling experience, making SysML v2 more accessible to users with varying levels of expertise. Future enhancements could extend its functionality to detect more complex inconsistencies or provide interactive guidance for model refinement. A graphical example of how the Suggestion Module works is shown in Table 5.

TABLE 5. Evaluation of clerical error handling in attribute matching.

Model Attribute	Suggested Attribute	Score
elementedId	elementId	2
isIndividuals	isIndividual	1
isSufficiented	isSufficient	2
isAbstracted	isAbstract	2
isImplyingIncluded	isImpliedIncluded	3
memberNamed	memberName	1
aliasssIds	aliasIds	2
ownedRelationshiping	ownedRelationship	3

As can be seen, different clerical errors were deliberately introduced into different attributes of multiple entities. As previously explained, the Suggestion Module computes the Levenshtein distance with all the attributes of each element. With this process, it is ensured that the returned attribute is the closest one, in terms of that distance, to the typing error itself, as can be seen in the "Suggested Attribute" column. In this example, all the suggestions correspond to the corrected attribute that the end user might have intended. However, in some cases, such as when the clerical error results in an attribute that does not belong to the given element, the suggestion can be incorrect.

V. CONCLUSION

The SysML v2 Validator developed in this work facilitates the verification and validation of models based on the beta 2 version of the SysML v2 Metamodel, as discussed in section III. This proposed methodology enables the early detection of both syntactic and semantic errors, ensuring adherence to the language specifications and established modelling best practices. Furthermore, the implementation of the validation process enhances user comprehension by providing structured feedback, thereby reducing the need for exhaustive prior knowledge of the language. Additionally, the whole process allows engineers in the MBSE context to automate different procedures that occur during different engineering processes.

The results obtained validate the effectiveness of the proposed validator in facilitating the automation of model evaluation. The integration of the Suggestion Module further enhances this process by providing a more comprehensive approach. In instances where syntax errors are detected, the module offers potential corrective suggestions, thereby improving the overall efficiency and usability of the validation framework.

The implementation of the validation process improves user comprehension by delivering structured feedback, thereby minimizing the requirement for extensive prior knowledge of the language. Furthermore, this process enables engineers in the Model-Based Systems Engineering (MBSE) domain to automate different procedures across different stages of the engineering workflow.

A. FUTURE WORK

The development of the SysML v2 validator still presents different opportunities for improvement and expansion in detecting both syntactic and semantic errors. This section outlines future lines of research and development aimed at enhancing the system's accuracy and robustness.

One of the main remaining challenges is the refinement of syntactic and semantic error identification, which requires a deeper analysis of the model's structure. Regarding syntactic errors, it is essential to conduct a thorough examination of the model to detect duplicate identifiers, as well as inconsistencies arising from the duplication of attributes within elements.

Concurrently, in industrial contexts, improving semantic error detection in systems models involves validating the presence and correctness of key attributes and relationships, particularly in complex domains like automotive, aerospace or defence. A major challenges relies in distinguishing between mandatory and optional elements and ensuring that critical dependencies are correctly defined. While SysML v2 Implementation provides a useful foundation, it is necessary to evaluate a broader range of industrial models to fully capture the variability and complexity found in real world systems. Case studies from industrial environments, such as embedded control or safety critical applications, highlight the need for careful analysis to identify missing references, redundant components and unused elements. Detecting such issues is crucial to reduce design errors and improve overall model quality. Furthermore, identifying cross-references helps uncover cyclic dependencies, which can compromise the consistency and reliability of the system architecture.

ACKNOWLEDGMENT

This work is part of the University Carlos III of Madrid (UC3M) research project CICLONES: Collaboration and smart continuous integration in Software and Systems Engineering with MBSE and DevOps.

REFERENCES

[1] T. Golgolnia, T. Kipouros, P. J. Clarkson, G. Marquardt, and M. Kevdzija, "Implementing the model-based systems engineering (MBSE) approach to develop an assessment framework for healthcare facility design," *Proc. Design Soc.*, vol. 4, pp. 1577–1586, May 2024.

[2] M. Hause, "The OMG modelling language (SYSML)," in *Proc. 5th Eur. Syst. Eng. Conf.*, vol. 638, Aug. 2007, p. 14.

[3] *OMG Systems Modeling Language (SysML) Version 2.0 Beta 2 Part 1: Language Specification*, Object Managment Group, Boston, MA, USA, 2024.

[4] J. Delsing, G. Kulcsár, and Ø. Haugen, "SysML modeling of service-oriented system-of-systems," *Innov. Syst. Softw. Eng.*, vol. 20, no. 3, pp. 269–285, Sep. 2024.

[5] M. Bajaj, S. Friedenthal, and E. Seidewitz, "Systems modeling language (SysML v2) support for digital engineering," *INSIGHT*, vol. 25, no. 1, pp. 19–24, Mar. 2022.

[6] L. Attouche, M.-A. Baazizi, D. Colazzo, G. Ghelli, C. Sartiani, and S. Scherzinger, "Validation of modern JSON schema: Formalization and complexity," *Proc. ACM Program. Lang.*, vol. 8, pp. 1451–1481, Jan. 2024.

[7] R. Saini, G. Mussbacher, J. L. C. Guo, and J. Kienzle, "Automated traceability for domain modelling decisions empowered by artificial intelligence," in *Proc. IEEE 29th Int. Requirements Eng. Conf. (RE)*, Sep. 2021, pp. 173–184.

[8] M. Zhang, C. Tao, H. Guo, and Z. Huang, "Recovering semantic traceability between requirements and source code using feature representation techniques," in *Proc. IEEE 21st Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Dec. 2021, pp. 873–882.

[9] A. Widner, V. Tihanyi, and T. Tettamanti, "Framework for vehicle dynamics model validation," *IEEE Access*, vol. 10, pp. 35422–35436, 2022.

[10] R. Baduel, M. Chami, J.-M. Bruel, I. Ober, S. Trujillo, A. Pierantonio, S. Trujillo, and A. Pierantonio, "Sysml models verification and validation in an industrial context: Challenges and experimentation," in *Modelling Foundations and Applications*, vol. 10890. Cham, Switzerland: Springer, 2018, pp. 132–146.

[11] V. Molnár, B. Graics, A. Vörös, S. Tonetta, L. Cristoforetti, G. Kimberly, P. Dyer, K. Giammarco, M. Koethe, J. Hester, J. Smith, and C. Grimm, "Towards the formal verification of SysML v2 models," in *Proc. ACM/IEEE 27th Int. Conf. Model Driven Eng. Lang. Syst.*, NY, NY, USA, Sep. 2024, pp. 1086–1095.

[12] B. Sultan and L. Apvrille, "AI-driven consistency of SysML diagrams," in *Proc. ACM/IEEE 27th Int. Conf. Model Driven Eng. Lang. Syst.*, Sep. 2024, pp. 149–159.

[13] R. Thorburn, V. Sassone, A. S. Fathabadi, L. Aniello, M. Butler, D. Dghaym, and T. S. Hoang, "A lightweight approach to the concurrent use and integration of SysML and formal methods in systems design," in *Proc. 25th Int. Conf. Model Driven Eng. Lang. Systems, Companion Proc.*, Oct. 2022, pp. 83–84.

[14] D. Dori, *Model-based Systems Engineering With OPM and SysML*, vol. 15. Cham, Switzerland: Springer, 2016.

[15] B. Keskin, B. Salman, and O. Koseoglu, "Architecting a BIM-based digital twin platform for airport asset management: A model-based system engineering with SysML approach," *J. Construct. Eng. Manage.*, vol. 148, no. 5, May 2022, Art. no. 04022020.

[16] S. Al-Fedaghi and H. Alahmad, "Integrated modeling methodologies and languages," in *Proc. 12th Int. Conf. Ubiquitous Inf. Manage. Commun.*, Jan. 2018, pp. 1–8.

[17] *Omg Systems Modeling Language (omg Sysml), V1. 0*, Object Manage. Group (OMG), O. A. Specification, 2007.

[18] A. M. Law, "How to build valid and credible simulation models," in *Proc. Winter Simul. Conf. (WSC)*, Dec. 2019, pp. 1402–1414.

[19] N. Jansen, J. Pfeiffe, B. Rumpe, D. Schmalzing, and A. Wortmann, "The language of SysML v2 under the magnifying Glass," *J. Object Technol.*, vol. 21, no. 3, pp. 1–3, 2022.

[20] W. Torres, M. G. J. van den Brand, and A. Serebrenik, "A systematic literature review of cross-domain model consistency checking by model management tools," *Softw. Syst. Model.*, vol. 20, no. 3, pp. 897–916, Jun. 2021.

[21] Y. Vanommeslaeghe, B. Van Acker, M. Cornelis, and P. De Meulenaere, "Towards continuous verification and validation of multi-domain system designs," in *Proc. ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. Companion (MODELS-C)*, Oct. 2023, pp. 495–499.

[22] M. A. Mohamed, G. Kardas, and M. Challenger, "Model-driven engineering tools and languages for cyber-physical Systems–A systematic literature review," *IEEE Access*, vol. 9, pp. 48605–48630, 2021.

[23] A. Tolk and J. Muguira, "The levels of conceptual interoperability model," in *Proc. Fall Simul. Interoperability Workshop 7*, Jan. 2003, pp. 1–11.

[24] J. Vaicenavičius, T. Wiklund, D. Kavolis, S. Draukčas, A. Kalkauskas, and R. Vaicenavičius, "SysIDE: SysML v2 textual editing and analysis system: Overview and applications," *CEAS Space J.*, vol. 17, pp. 1–7, Feb. 2025.

[25] Eclipse Found., Inc. (2025). *Welcome To Syson Docs*. [Online]. Available: https://doc.mbse-syson.org/syson/v2025.1.0/index.html

[26] No Magic, Inc. (2025). *Cameo Systems Modeler*. Accessed: Mar. 31, 2025. [Online]. Available: https://www.nomagic.com/products/cameo-systems-modeler

[27] S. Cvijic, S. Hookway, S. Harrison, and A. Strelzoff, "Probabilistic AI reliability analysis in MBSE and SysML," in *Proc. Annu. Rel. Maintainability Symp. (RAMS)*, Jan. 2025, pp. 1–6.

[28] A. Idani, "Formal model driven engineering," Ph.D thesis, Dept. Comput. Sci., Université Grenoble-Alpes, Grenoble, France, 2023.

[29] Z. Mao, *Model Validation and Uncertainty Quantification, Volume 3*. Cham, Switzerland: Springer, 2021.

[30] E. Cibrián, J. M. Álvarez-Rodríguez, R. Mendieta, and J. Llorens, "Towards a method to enable the selection of physical models within the systems engineering process: A case study with simulink models," *Appl. Sci.*, vol. 13, no. 21, p. 11999, Nov. 2023.

[31] C. Fu, J. Liu, and S. Wang, "Building SysML model graph to support the system model reuse," *IEEE Access*, vol. 9, pp. 132374–132389, 2021.

[32] A. Abouzahra, A. Sabraoui, and K. Afdel, "Model composition in model driven engineering: A systematic literature review," *Inf. Softw. Technol.*, vol. 125, Sep. 2020, Art. no. 106316.

[33] R. T. Geraldi, S. Reinehr, and A. Malucelli, "Software product line applied to the Internet of Things: A systematic literature review," *Inf. Softw. Technol.*, vol. 124, Aug. 2020, Art. no. 106293.

[34] Q. Ma, M. Kaczmarek-Heß, and S. de Kinderen, "Validation and verification in domain-specific modeling method engineering: An integrated life-cycle view," *Softw. Syst. Model.*, vol. 22, no. 2, pp. 647–666, Apr. 2023.

[35] Powered by Starion Group. (2024). *OMG SysML Version 2 Ecore-based Meta Model Documentation*. [Online]. Available: https://modeldocs.sysml2.net/#Namespace

[36] J. V. Soares do Amaral, J. A. B. Montevechi, R. D. C. Miranda, and W. T. D. S. Junior, "Metamodel-based simulation optimization: A systematic literature review," *Simul. Model. Pract. Theory*, vol. 114, Jan. 2022, Art. no. 102403.

[37] I. Poernomo, "The meta-object facility typed," in *Proc. ACM Symp. Appl. Comput.*, Apr. 2006, pp. 1845–1849.

[38] A. A. AlRababah, "Assessing the effectiveness of UML models in software system development," *Int. J. Appl. Sci. Res.*, vol. 7, no. 1, pp. 13–24, 2024.

[39] I. N. Magic, *Model-Based Systems Engineering With MagicDraw*. Waltham, MA, USA: No Magic, 2021.

[40] T. J. Parr and R. W. Quong, "ANTLR: A predicated-LL(k) parser generator," *Software, Pract. Exper.*, vol. 25, no. 7, pp. 789–810, Jul. 1995.

[41] A. D. Hristozov and E. T. Matson, "DynADL–Dynamic architecture description language for system of systems," in *Proc. IEEE Int. Syst. Conf. (SysCon)*, Apr. 2024, pp. 1–8.

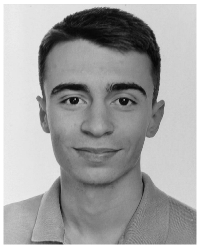
[42] T. J. Parr, "The definitive ANTLR 4 reference," Pragmatic Bookshelf, Dallas, TX, USA, 2013.

[43] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. ACM*, vol. 21, no. 1, pp. 168–173, Jan. 1974.

[44] L. Yujian and L. Bo, "A normalized Levenshtein distance metric," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1091–1095, Jun. 2007.

[45] Eclipse Found. (2025). *Capella*. Accessed: Mar. 31, 2025. [Online]. Available: https://www.eclipse.org/capella/

EDUARDO CIBRIÁN is currently an Assistant Professor with the Carlos III University of Madrid and a Researcher of software engineering. His academic background is focused on information retrieval from engineering artifacts, model-based systems engineering (MBSE), and the application of standards, such as SysML v2 for the development of cyber-physical systems. He has participated in European projects, such as H2020 AHTOOLS and H2020 IREL4.0, integrating theory and practice to improve engineering methods and their application in industry. His academic career culminated with the defense of his doctoral thesis in April 2024, receiving the highest distinction: Summa Cum Laude. His teaching work is centered on creating dynamic and up-to-date learning experiences, combining technological innovation with active teaching methodologies. He has taught courses, such as programming, software engineering, application programming, and advanced excel.



JOSE OLIVERT-ISERTE received the B.S. degree in data science from Valencia Polytechnic University. He is currently pursuing the M.S. degree in computer science and technology with the Carlos III University of Madrid (UC3M). He has prior experience as a Junior Data Scientist at NTT DATA and ALTUM Proyectos de Ingeniería, where he contributed to the development of machine learning models, cloud-based microservices, and full-stack applications. He is a Researcher at UC3M, where he works on model-based systems engineering (MBSE), with a focus on SysML v2. His research interests include deep learning, computer vision, and intelligent systems engineering.

CARLOS DÍEZ-FENOY received the degree in computer engineering from the Carlos III University of Madrid (UC3M), where he is currently pursuing the master's degree in computer science and technology. He is a Researcher with The Reuse Company, where he works in the field of model-based systems engineering (MBSE), with a particular focus on SysML v2. His research interests include machine learning, deep learning, and intelligent systems engineering. Independently, he has developed projects involving convolutional neural networks for Braille language detection, exploring the use of artificial intelligence in accessibility applications.

ROY MENDIETA received the degree in computer science engineer from the Catholic University of Honduras, the master's degree in computer science and technology from the Carlos III University of Madrid, the master's degree in economics and finance from the Catholic University of Honduras, and the Ph.D. degree in computer science and technology from the Carlos III University of Madrid. He is a Senior Software Engineer with The Reuse Company. Before these role, he was the IT Project Manager of Central Bank of Honduras, where he was responsible for the implementation of systematization projects of economic statistics generation processes, using as a base, international methodologies for the generation of economic statistics. Additionally, he has a Project Management Professional Certification (PMP) by the Project Management Institute (PMI). His main lines of research are knowledge reuse and interoperability.

JUAN LLORENS received the M.S. degree in industrial engineering from the ICAI Polytechnic School, UPC University, Madrid, in 1986, Spain, and the Ph.D. degree in industrial engineering and robotics from the Carlos III University of Madrid, Spain, in 1996.

He started his own company in 1987, dealing with artificial intelligence applied to database systems, named Knowledge Engineering SL (KE). He worked as the Technical Director of KE and, afterwards, as the Marketing Director of Advanced Vision Technologies, until 1992, when he joined Carlos III University of Madrid. Since 2003, he has been a Professor at the Carlos III University of Madrid. His main subject is knowledge centric systems engineering and reuse, which he teaches in the informatics studies at the University. He is the Leader of the Knowledge Reuse Group (KR Group), within the University. In 1998, he was invited to the Åland University of Applied Sciences (HÅ), Åland, Finland. From 1998 to 2008, he split his educational activities between Madrid's University and the HÅ, where he taught different software engineering subjects. He is currently a Professor with the Computer Science and Engineering Department, Carlos III University of Madrid. His current research interests include knowledge technologies and system engineering techniques integration toward software, systems and knowledge quality, traceability, interoperability, reasoning, and reuse.

Dr. Llorens is a Full Member of the International Council on Systems Engineering (INCOSE) and the past President of INCOSE Spain (AEIS). He is the creator of the INCOSE Knowledge Management and Ontologies Working group as well as a member of the Requirements Working Group. He holds a Certified Systems Engineering Professional (CSEP) accreditation and an Expert Systems Engineering Professional (ESEP). His CV is presented in http://www.linkedin.com/pub/juan-llorens/b/857/632.

JOSE MARÍA ÁLVAREZ-RODRÍGUEZ was born in Oviedo, Asturias, Spain, in 1983. He received the B.S. and M.S. degrees in computer science and engineering and the Ph.D. degree in software engineering from the University of Oviedo, Spain, in 2005, 2007, and 2012, respectively.

From 2005 to 2010, he was a Research Engineer with the CTIC Foundation in the Semantic Technologies area and an Assistant Professor with the University of Oviedo, from 2008 to 2009 and from 2011 to 2012. In 2013, he joined as a Visiting Professor with the Computer Science and Engineering Department, Carlos III University of Madrid, Spain. Since 2018, he has been an Associate Professor with the Computer Science and Engineering Department. He is the author of more than 100 works in high-impact factor journals, conferences, and book chapters. His research interests include systems engineering, service-oriented computing, interoperability semantic technologies, and linked big data. He was a recipient of the HPC2-Europe Transnational Access Program grant at SARA (The Netherlands), hosted by the University of Amsterdam. In 2013, he was granted a Marie Curie Research Fellow (Experienced Researcher) at SEERC (Thessaloniki, Greece) within the RELATE-ITN FP7 Project researching in ''quality management in service-based systems and cloud applications.'' He is also a member of ISO, LOTAR, INCOSE, OMG, and ProSTEP. He has also served as a guest editor in three high-impact factor journals.