Rethinking Model Differencing for SysML v2: Challenges and Future Directions

Gvidas Ambrozaitis [1] and Mantas Jurgelaitis [1]

[1] *Department of Information Systems, Faculty of Informatics, Kaunas University of Technology, Studentų St. 50, Kaunas, 51368, Lithuania*

Abstract

SysML v2 introduces new challenges in model differencing because of its textual notation, reduced dependence on tool-specific exchange formats, and explicit structural relations between model elements. Existing differencing approaches designed mainly for UML, XMI-based models, or identifier-dependent versioning scenarios are not fully suitable for SysML v2. This paper analyzes the characteristics of SysML v2 relevant to model comparison and reviews existing differencing approaches and tools in order to identify the main requirements for SysML v2 model differencing. Based on this analysis, the main concepts of a graph-based SysML v2 differencing solution are proposed. The proposed process includes transformation of models into abstract syntax trees, matching of model elements, generation of a delta script, and presentation of detected changes in textual and graphical form. In addition, optional summarization of differencing results using a large language model is considered to improve the readability of comparison output. An illustrative example demonstrates how the proposed approach can organize the results according to SysML v2 specific concepts.

Keywords

MBSE, SoS, SysML v2, LLM, model differencing, model comparison.

1. Introduction

In Model Based System Engineering (MBSE) various models are created in order to describe the system being designed from different perspectives, levels of abstraction, and development phases [1]. Depending on the specifics of the system under development, an appropriate modeling language is selected. Among the many system modeling languages applied in model based systems engineering, one of the most widely used is SysML (Systems Modeling Language) [2]. It is a graphical modeling language intended for modeling Systems of Systems (SoS), whose metamodel is based on the UML (Unified Modeling Language) metamodel [3]. Using this language in MBSE makes it possible to specify system requirements, behavior, structure and parameters. The newest version of this language – SysML v2 offers a new metamodel and not only a graphical but also a textual language notation for modeling systems. The functionality of the new notation enables describing models using a formal, human-readable textual format, which makes it possible to analyze models in new ways, share them, and apply machine learning algorithms [2].

In collaborative MBSE practice, maintaining consistency between evolving models is a significant challenge. As models are modified, differences may arise through element addition, deletion, renaming, restructuring, or independent development. This creates a need for model differencing methods capable of identifying correspondences between model elements, detecting relevant changes, and supporting activities such as version control, integration, and review [4]. Existing model comparison and merge solutions already address similar problems for other modelling languages, typically relying on rule-based analysis of syntax, structure, and element attributes, while semantic aspects are often handled separately [5].

*IVUS 2026: Information Society and University Studies 2026, May 12, Kaunas, Lithuania

gvidas.ambrozaitis@ktu.lt (G. Ambrozaitis); mantas.jurgelaitis@ktu.lt (M. Jurgelaitis)

0009-0007-6033-9777 (G. Ambrozaitis); 0000-0003-2221-0765 (M. Jurgelaitis)

Copyright © 2026 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-06

CEUR-WS.org/Vol-4317/paper2.pdf

However, SysML v2 introduces new conditions for differencing, particularly because its textual notation reduces dependence on tool-specific exchange mechanisms and enables models to be processed as text-based artifacts [6], [7]. For this reason, it is important to examine how existing differencing approaches and tools can be applied to SysML v2.

At the same time, when comparing models, for the result to be useful in modeling practice, it would be beneficial to present a report in an understandable textual and graphical form [8]. In most model comparison cases, the resulting report is too extensive to be presented simply [9]. When facing this problem, it is useful to analyze the application of large language models to summarize [10], [11] the results of comparison algorithms, emphasizing significant design differences in the model rather than minor technical deltas.

The goal of this paper is to outline the principles of a prospective model differencing approach for SysML v2 that exploits the capabilities of its textual and graphical notations. Therefore, we analyzed the characteristics of SysML v2 and existing model differencing approaches and tools in order to identify the main requirements for comparison of SysML v2 models. On this basis, we proposed the main concepts of a graph-based differencing solution that supports model element matching, delta script generation, and interpretation of comparison results in textual, graphical, and summarized form.

The rest of the paper is organized as follows. Background covers the background on SysML v2 and various approaches to model differencing. Proposal for SysML v2 describes the main concepts of the proposal for a SysML v2 model differencing process. Conclusions concludes and presents future work.

2. Background

2.1. SysML v2 Modeling Language

SysML v2 [2] is a next-generation systems modeling language developed by the Object Management Group (OMG) to support the modeling of complex systems. It is intended to describe system requirements, structure, behavior, and relationships in a more precise and flexible way than its predecessor, SysML v1 [3]. In practice, SysML v2 helps engineers represent how a system is organized, how its parts interact, and whether it satisfies defined requirements.

One of the major changes introduced in SysML v2 is its new metamodel foundation. Unlike SysML v1, which was built on top of UML, SysML v2 is based on the KerML [12] language. This new metamodeling foundation removes several limitations inherited from UML and enables richer semantic representation of systems and their elements. As a result, SysML v2 provides a more consistent and extensible basis for systems engineering modeling [2].

From the perspective of model differencing, one of the most important advances in SysML v2 is the support for both graphical and textual notations that are semantically equivalent. The same model may be developed graphically or textually and transformed between these representations while preserving a unified underlying meaning. This enables more flexible modelling workflows and, importantly, creates new opportunities for model exchange and automated processing.

Unlike SysML v1, where model exchange commonly depended on XMI [13] and tool-specific representations, SysML v2 allows models to be shared as textual artefacts. Such a shift reduces dependence on a single modelling environment and makes it more practical to apply techniques inspired by source code differencing, text processing, and language-aware analysis [7].

These SysML v2 characteristics are especially relevant when considering model comparison. Existing differencing approaches developed for UML and SysML v1 often assume that models are exchanged in XMI form and that model elements can be matched using generated unique identifiers. Such assumptions are suitable in controlled versioning scenarios within a single tool, but they become less reliable when models are created independently, exchanged across tools, or represented through textual notation. For this reason, SysML v2 raises the need to reassess established model differencing methodologies and to investigate approaches that rely not only on persistent identifiers, but also on structural, syntactic, and semantic similarity between model elements.

2.2. Principles of Model Differencing Algorithms

All systems of systems or software models that involve more than one model developer, or that are developed iteratively, need to be checked regularly in order to maintain model consistency. In software engineering, this problem is commonly addressed using version control tools such as Git [14], which compare textual artefacts and help manage changes between project versions. Git's default diff behavior is based on the Myers algorithm [15], and text-based differencing approaches more broadly are commonly framed as sequence comparison problems related to the longest common subsequence (LCS) [16]. Such approaches are effective for source code and other sequential text files, where changes are represented as insertions, deletions, and nearby modifications in a linear sequence. However, modelling artefacts in MBSE are typically structured as interconnected elements and relationships rather than plain text. Because of this, text-based differencing is often insufficient for modelling practice: it may report moved or reorganized model content as unrelated additions and deletions, while failing to capture structural correspondences between model elements. Algorithms like GumTree [17] addresses that problem and tries to achieve more accurate source code differencing by transforming code into graphs and utilizing graph-based differencing. Graph-based model differencing [17], [18] addresses this limitation by operating on model structure and relationships, allowing changes such as additions, deletions, moves and modifications to be analyzed in terms of the model itself rather than only its textual representation. In MBSE projects, as bigger models are created, the likelihood of inconsistencies increases, such as duplicated elements, renamed concepts, or structurally similar fragments created independently. To address this problem, various model version management tools [19], [20], [21] are used; their functionality often includes model comparison algorithms that identify similarities and differences between models, as well as merge algorithms that support combining several versions into one.

The model comparison problem itself is usually decomposed into several parts [20], [22]: calculation, which identifies corresponding elements and computes the differences between models; representation, which defines the internal structure of the comparison result; and visualization, which presents the result in a human-understandable form. Such a decomposition makes it possible to analyze and evaluate different comparison methodologies by treating these parts separately. However, it is worth noting that the visualization part is not always integral, and in further analysis more attention is given to calculation and representation.

Model differencing methodologies are also classified according to additional properties, such as their dependence on a specific modelling language metamodel and the principle used to find correspondences between model elements, alternatively known as model matching. Model matching approaches are commonly divided into several categories [22]. One of the most straightforward is static identity-based matching, in which correspondences are established using unique and stable element identifiers assigned at creation time. This strategy was effective when comparing different versions of the same UML or other XMI compatible model [13], but it is less suitable when models have been created independently, since their identifiers do not coincide [23]. Another category is signature-based matching, where elements are compared using a similarity score derived from their signatures, that is, combinations of properties such as type, name, stereotype, and other attributes. Unlike identifier-based matching, this approach can also be applied to independently created models [24]. A broader category is similarity-based matching, in which the model is treated as a graph of elements and their properties, and correspondences are established by aggregating and comparing node characteristics. In such cases, different properties may contribute unequally to the comparison result, which means that appropriate weights often need to be assigned to them [18], [22]. Finally, custom language-specific matching algorithms rely on rules tailored to a particular modelling language or notation [25]. Such algorithms often provide more accurate results for their target language, but their applicability is usually limited outside that specific context.

Existing research shows that many current solutions are oriented toward UML based models and are often most effective when comparing different versions of the same model. However, the characteristics of SysML v2 motivate re-examining how such graph-based differencing methodologies can be applied to this language. Static identity-based matching is not well suited to SysML v2 textual notation and its reduced dependence on a single modelling tool, since model exchange is no longer inherently tied to tool-generated persistent identifiers. As a result, approaches that rely more strongly on structural properties, element signatures, or other forms of similarity appear more relevant for SysML v2 model comparison. For this reason, this paper focuses on graph-based model comparison,

since SysML v2 models contain explicit structural relations between elements that cannot be captured adequately by text-based differencing alone.

2.3. Existing Model Differencing Approaches and Tools

This section reviews relevant model differencing approaches and comparison capabilities provided by existing modelling (or model differencing) tools. One of the more straightforward approaches is the identifier-based method proposed by Alanen [23], intended for MOF-based models [26]. In this method, the difference between two models is represented as a sequence of operations, including creation, deletion, and modification. The comparison process begins by establishing traceability between elements through their unique identifiers. Elements that appear only in one of the models compared are then interpreted as additions or deletions, depending on the versioning context, while matched elements are checked for changes in their values. The resulting sequence can be used not only to describe model evolution, but also to support model merging and reconstruction of previous states through inverse transformations. This approach is mainly suitable for comparing different versions of the same model, since independently created models usually do not share static identifiers.

In contrast, similarity-based differencing approaches attempt to identify corresponding elements even when unique identifiers are unavailable. SiDiff [18], for example, compares recursively structured model elements using element signatures and weighted similarity functions, and classifies the detected changes into categories such as structural, attribute, reference, and move differences. UMLDiff [25] similarly combines signature-based and structural similarity, using unchanged names as anchors and then identifying more complex modifications such as renaming and moving. EMF Compare [20] applies similar principles in a practical modelling framework by evaluating similarity based on element type, name, attributes, values, structural context, and relationships with other elements. It also applies filtering and scoping mechanisms to reduce informational noise in the comparison results. These approaches show that model differencing is not limited to exact identifier matching and that more flexible comparison can be achieved by relying on structural similarity. At the same time, they highlight that useful differencing depends not only on accurate matching, but also on clear classification and presentation of the detected changes.

These principles can also be observed in existing modelling tools. Established environments such as CATIA Magic [21] and Enterprise Architect [27] provide broad support for MBSE languages and frameworks, while SysML v2 support is still emerging or limited to newer product versions. Their comparison capabilities are primarily oriented toward version management within the same modelling lineage and in practice appear to rely heavily on internally generated unique identifiers. As a result, such tools are effective when comparing successive versions of the same project but are less suitable for comparing independently created yet structurally similar models. Plugins such as LemonTree [19] extend these environments with improved comparison visualization and merge support, but they still inherit the underlying dependence on stable element identities. Other tools like Papyrus [28], together with EMF Compare [20], enables model comparison at the level of model elements and relationships rather than only file text, thereby providing more informative and model-aware difference visualization. However, strong SysML v2 support is not evident. In contrast, newer SysML v2-focused tools such as SysIDE [29] reflect the shift toward textual modelling and tool interoperability. SysIDE, for example, supports textual SysML v2 modelling and validation, while diagram viewing can be achieved through integration with external viewers [30]. However, it does not currently provide dedicated comparison or merge functionality.

Overall, the analyzed approaches and tools suggest that current differencing solutions are either highly effective in controlled versioning scenarios or more flexible through similarity-based matching, but that robust comparison support for SysML v2 remains limited. This is particularly important in cases where models may be created across different tools, evolve independently, or include renaming and restructuring modifications that cannot be captured reliably through identifier-based matching.

3. Proposal for SysML v2 Model Differencing

3.1. The Main Concepts of the Proposed Model Differencing Solution

Based on the limitations of existing differencing approaches discussed in the previous section, we propose a SysML v2 model differencing solution intended to support comparison of SysML v2 models in a way that is less dependent on tool-specific identifiers and aligned with the textual and structural characteristics of the language. The proposed process consists of four main stages: transformation of SysML v2 models into an abstract syntax tree representation, matching of corresponding model elements, generation of a delta script, and presentation of detected differences in textual and graphical form.

The overview of requirements of the proposed solution is presented in Figure 1. After creating models in the modelling environment, the systems of systems engineer performs the comparison of the differencing solution. Model comparison includes transformation of models into ASTs, matching of model elements, generation of a delta script, and presentation of differences. This decomposition reflects the main parts of the differencing process, namely calculation, representation, and visualization. The process also introduces an optional summarization for differencing results, where an LLM can summarize the produced comparison output when the systems engineer requests a more concise interpretation of the detected differences. This additional functionality is intended to reduce the complexity of raw differencing output and to emphasize the most significant model modifications.

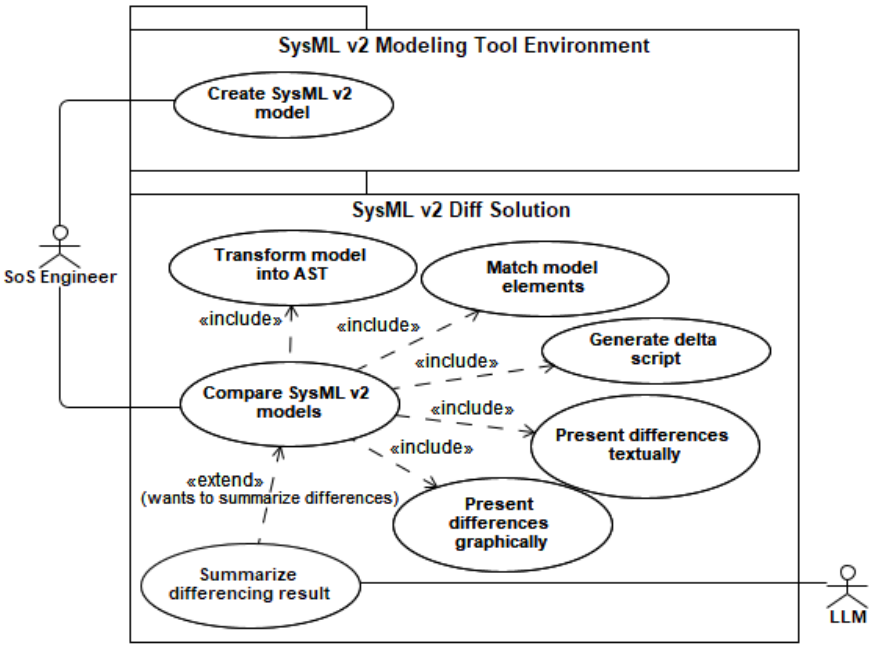


Figure 1: Overview of requirements of the proposed SysML v2 model differencing solution

The proposed process is presented in Figure 2. The process starts with the systems engineer selecting two SysML v2 models for comparison. Each model is then transformed into an abstract syntax tree (AST), which serves as the intermediate structure. Since SysML v2 provides a textual notation, the language can be processed using parsing-based techniques that produce structured representations of model contents. For this purpose, a parser generator such as ANTLR [31] could be used, since it supports construction of parse trees from formal grammars and provides a practical basis for deriving AST-like structural representations of textual models. Such representations make it possible to analyze containment, hierarchy, element properties, and other structural relationships. This is important when comparing models that may have been created independently or edited in different tools, where identifier-based traceability cannot be assumed.

Once both model representations are obtained, the matching stage can rely on descriptive properties of model elements rather than only on static identifiers. A tree differencing framework such as GumTree [17] could be applied to perform graph- or tree-based element matching and to generate the corresponding delta script describing the identified changes between the compared models. GumTree is designed around descriptive property differencing and delta script generation, which makes it a suitable approach for matching SysML v2 model elements represented in AST form.

Finally, the produced differencing results are presented in two complementary ways: both graphically and textually.

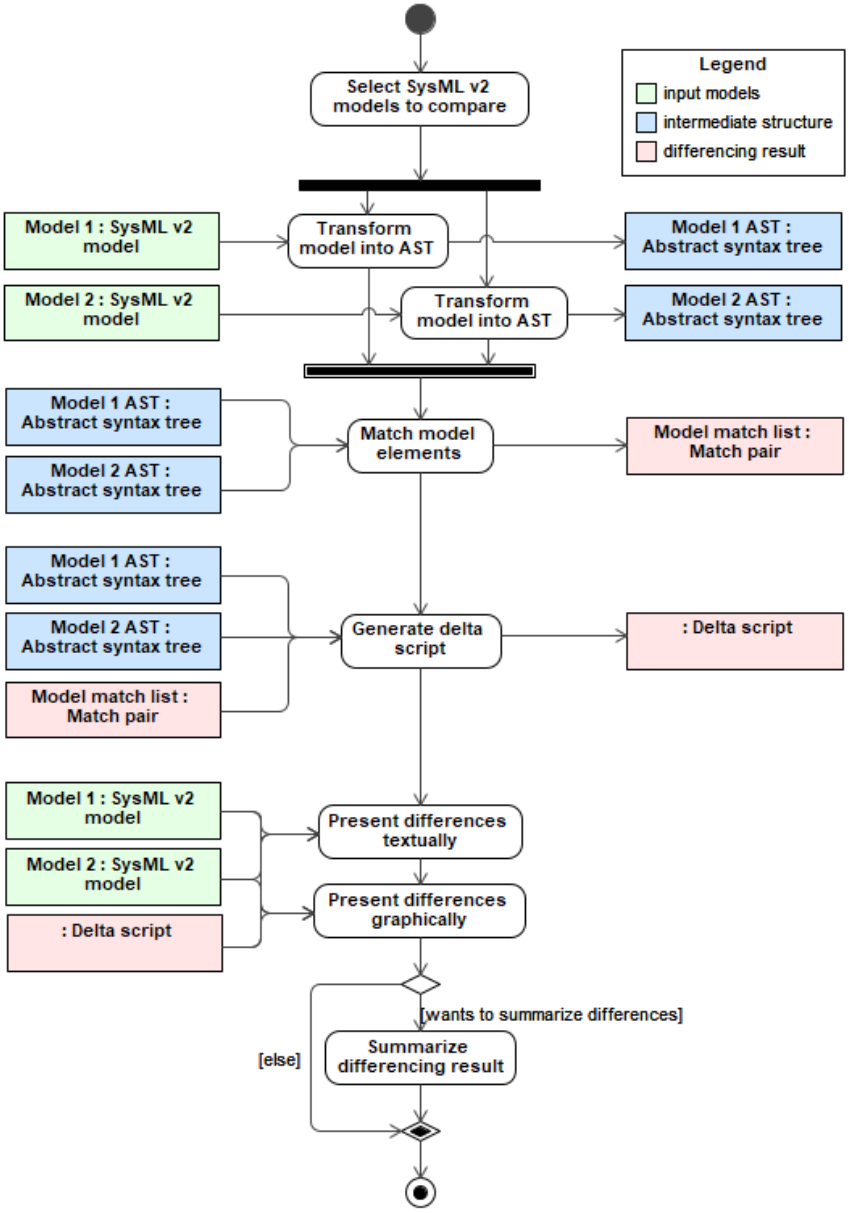


Figure 2: General overview of the proposed SysML v2 model differencing solution

The conceptual model of the proposed solution is presented in Figure 3. In this conceptual view, a SysML v2 model is transformed into an abstract syntax tree composed of vertices and edges representing model elements, thereby making the structural organization of the model explicit. The solution identifies model matches represented as match pairs between corresponding elements. These matches are then used to derive edit operations, which are collected into a delta script. This conceptual data model shows the main principles of the proposal: SysML v2 models are first represented as graph-like structures, then compared through element matching, and finally expressed through a sequence of edit operations suitable for further representation and analysis.

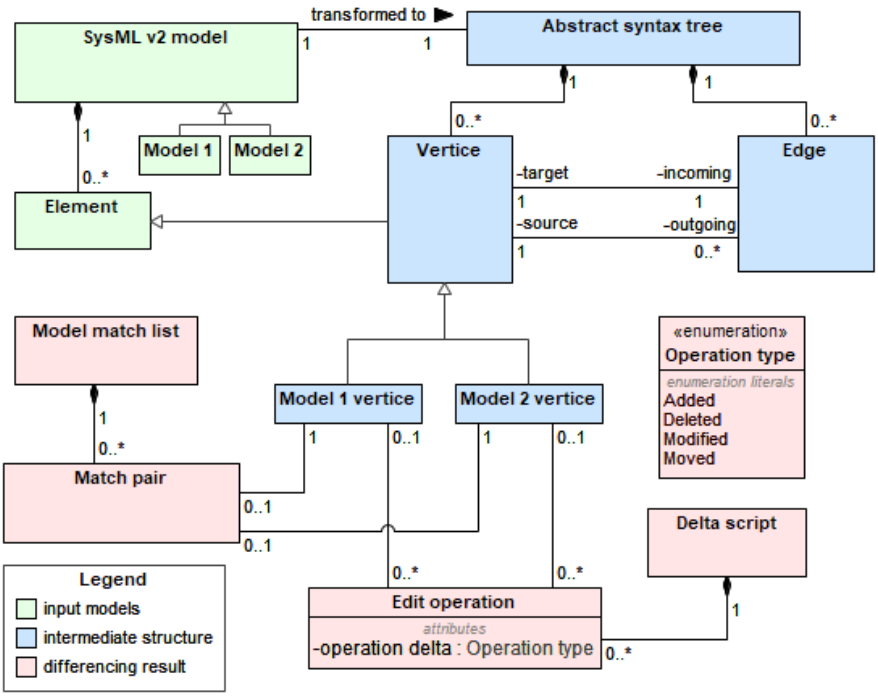


Figure 3: Conceptual model of the proposed SysML v2 model differencing solution

The proposal supports comparison of models without relying on static identifiers, which is particularly important in contexts where models may be created independently, exchanged across tools, or modified through restructuring operations. We also aim to present the detected changes in a human-readable form. The proposed process is intended to emphasize significant design-level differences between models rather than minor technical deltas, and thus to provide a more informative basis for review, interpretation, and further model refinement.

3.2. Example Illustrating Model Differencing Principles

To demonstrate the main principles of the proposed differencing solution, Figure 4 presents an example of two SysML v2 models. The example illustrates how the proposed approach can identify structural differences between model elements and present the detected changes in both textual and graphical form. The upper part of the figure contains the compared models in textual notation, whereas the lower part shows their graphical representations generated using the Tom Sawyer SysML v2 Viewer [30]. Displaying both views side by side helps relate textual changes to structural modifications in the model and thereby provides a clearer basis for interpreting the differencing results.

This example demonstrates that the proposed differencing solution is intended to support interpretation of SysML v2 changes at more than one level. In addition to identifying modified fragments in textual notation, it relates these changes to the structural arrangement of model elements in the compared models. This is important because the engineering significance of a detected change often depends on its context within the model rather than on the textual edit alone.

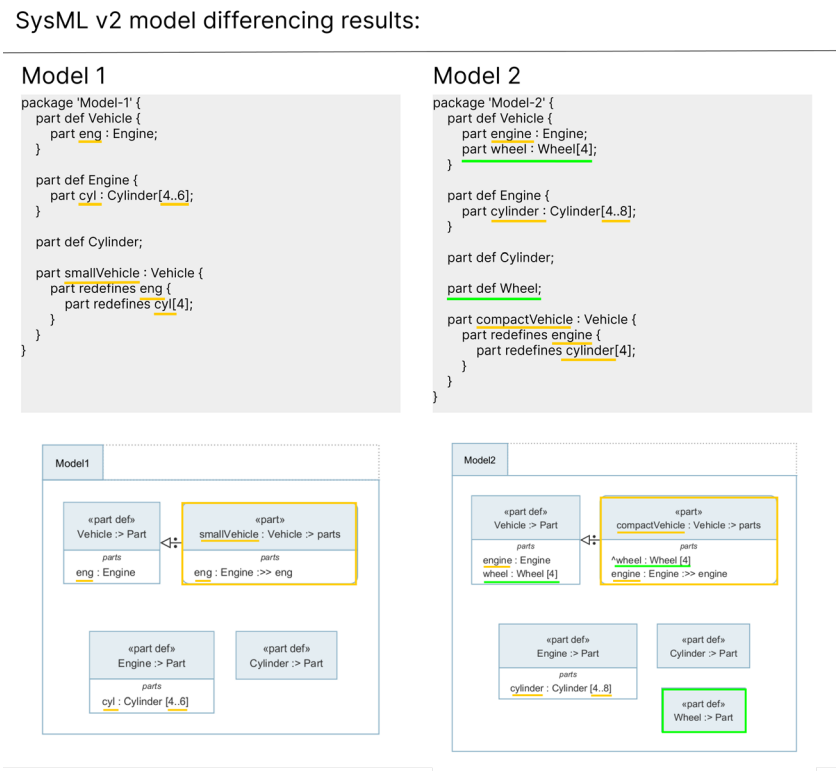


Figure 4: Example view for results of the proposed model differencing solution

To further support interpretation of the detected changes, GPT-5.4 [32] was used to summarize the differencing results in a more structured and human-readable form. In this case, the prompt explicitly instructed the model to present the output as a table and to cluster the detected changes according to whether they affected definition elements or usage elements. Such grouping is particularly relevant in the context of SysML v2, where the distinction between element definitions and their usages is an important part of the language structure. As illustrated in Figure 5, this prompt design made it possible to transform lower-level differencing results into a concise summary that highlights the main modifications more clearly. Instead of listing changes only as a sequence of technical edit operations, the generated summary organizes them according to SysML v2-specific modelling concepts, thereby making the results easier to interpret and more suitable for subsequent engineering analysis.

Category	Detected differences
Definition elements	• Modified `Engine` definition by: ◦ renaming `cyl` to `cylinder` ◦ changing `Cylinder` multiplicity from `[4..6]` to `[4..8]` • Added `Wheel` definition
Usage elements	• Modified `Vehicle` usage by: ◦ renaming `eng` to `engine` ◦ adding `wheel : Wheel[4]` • Modified `smallVehicle` usage by: ◦ renaming it to `compactVehicle` • Modified `compactVehicle` usage by: ◦ updating nested redefinition from `eng` to `engine` ◦ updating nested redefinition from `cyl[4]` to `cylinder[4]`

Figure 5: Summary of example view of model differencing by GPT-5.4 Thinking (OpenAI) LLM

This example indicates that the usefulness of SysML v2 differencing depends not only on accurate detection of changes, but also on their representation to the analyst. When textual and graphical views are complemented by a structured summary aligned with SysML v2 concepts such as definition and usage elements, the comparison results become more suitable for engineering interpretation. This

suggests that future SysML v2 differencing solutions should also support summarization of model evolution through representations that are closer to the modelling concepts used by engineers.

4. Conclusions

This paper addressed the problem of model differencing for SysML v2. The analysis of related approaches and tools showed that many existing differencing solutions are primarily oriented toward UML, and identifier-based comparison scenarios and therefore tend to be most effective when comparing successive versions of the same model within a single tool environment. However, SysML v2 introduces different conditions for model differencing because of its textual notation, reduced dependence on tool-specific exchange formats, and explicit structural relations between model elements. These characteristics make it necessary to reconsider established differencing methodologies and suggest that graph-based comparison is more suitable than purely text-based or strictly identifier-based approaches.

To address this problem, the main concepts of a proposed SysML v2 model differencing solution are outlined. Our proposal is based on transformation of models into abstract syntax tree, matching of corresponding elements, generation of a delta script, and presentation of detected changes in textual and graphical form. In addition, the proposal includes optional summarization of differencing results using an LLM in order to provide a more concise interpretation of detected model changes. In this way, the proposed solution is intended not only to support comparison of SysML v2 models beyond a single tool-specific version history, but also to make the resulting differences more useful for practical engineering analysis.

Future work will focus on implementation and evaluation of the proposed solution. This includes defining the concrete parsing process for transforming SysML v2 textual models into AST, selecting and adapting a suitable matching strategy, and specifying delta generation mechanisms for relevant change types such as additions, deletions, moves and modifications.

5. Declaration on Generative AI

The authors have not employed any Generative AI tools.

6. References

[1] A. L. Ramos, J. V. Ferreira, and J. Barcelo, "Model-Based Systems Engineering: An Emerging Approach for Modern Systems," *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 42, no. 1, pp. 101–111, Jan. 2012, doi: 10.1109/TSMCC.2011.2106495.

[2] "SysML v2." Accessed: Oct. 26, 2025. [Online]. Available: https://www.omg.org/sysml/sysmlv2/

[3] "SysML v1." Accessed: Oct. 26, 2025. [Online]. Available: https://www.omg.org/sysml/sysmlv1/

[4] R. F. Paige, N. Matragkas, and L. M. Rose, "Evolving models in Model-Driven Engineering: State-of-the-art and future challenges," *J. Syst. Softw.*, vol. 111, pp. 272–280, Jan. 2016, doi: 10.1016/j.jss.2015.08.047.

[5] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, and M. Wimmer, "An Introduction to Model Versioning," in *Formal Methods for Model-Driven Engineering*, vol. 7320, M. Bernardo, V. Cortellessa, and A. Pierantonio, Eds., in Lecture Notes in Computer Science, vol. 7320. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 336–398. doi: 10.1007/978-3-642-30982-3_10.

[6] E. Cibrián, J. Olivert-Iserte, C. Díez-Fenoy, R. Mendieta, J. Llorens, and J. María Álvarez-Rodríguez, "Ensuring Semantic Consistency in SysML v2 Models Through Metamodel-Driven Validation," *IEEE Access*, vol. 13, pp. 121444–121457, 2025, doi: 10.1109/ACCESS.2025.3587786.

[7] J. Vaicenavičius, T. Wiklund, D. Kavolis, S. Draukšas, A. Kalkauskas, and R. Vaicenavičius, "SysIDE: SysML v2 textual editing and analysis system: overview and applications," *CEAS Space J.*, Feb. 2025, doi: 10.1007/s12567-025-00595-x.

[8] S. Maoz, J. O. Ringert, and B. Rumpe, "A Manifesto for Semantic Model Differencing," 2014, doi: 10.48550/ARXIV.1409.2485.

[9] S. Maoz, J. O. Ringert, and B. Rumpe, “Summarizing Semantic Model Differences,” 2014, doi: 10.48550/ARXIV.1409.2307.
[10] A. M. Dakhel, A. Nikanjam, F. Khomh, M. C. Desmarais, and H. Washizaki, “An Overview on Large Language Models,” in *Generative AI for Effective Software Development*, A. Nguyen-Duc, P. Abrahamsson, and F. Khomh, Eds., Cham: Springer Nature Switzerland, 2024, pp. 3–21. doi: 10.1007/978-3-031-55642-5_1.
[11] C. Mannari, T. Turchi, M. Bacco, and A. Malizia, “Assisting Stakeholders in Class Diagram Interpretation with LLMs: a Work in Progress,” in *2025 IEEE 33rd International Requirements Engineering Conference Workshops (REW)*, Valencia, Spain: IEEE, Sep. 2025, pp. 298–303. doi: 10.1109/REW66121.2025.00045.
[12] “KerML Specification 1.0.” Accessed: Mar. 18, 2026. [Online]. Available: https://www.omg.org/spec/KerML
[13] “XMI Specification 2.5.1.” Accessed: Mar. 18, 2026. [Online]. Available: https://www.omg.org/spec/XMI/2.5.1/About-XMI
[14] “Git documentation.” Accessed: Dec. 03, 2025. [Online]. Available: https://git-scm.com/docs
[15] E. W. Myers, “AnO(ND) difference algorithm and its variations,” *Algorithmica*, vol. 1, no. 1, pp. 251–266, Nov. 1986, doi: 10.1007/BF01840446.
[16] Y. S. Nugroho, H. Hata, and K. Matsumoto, “How different are different diff algorithms in Git?,” *Empir. Softw. Eng.*, vol. 25, no. 1, pp. 790–823, Jan. 2020, doi: 10.1007/s10664-019-09772-z.
[17] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, “Fine-grained and accurate source code differencing,” in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, Vasteras Sweden: ACM, Sep. 2014, pp. 313–324. doi: 10.1145/2642937.2642982.
[18] U. Kelter, J. Wehren, and J. Niere, “A generic difference algorithm for UML models,” *Softw. Eng. 2005*, pp. 105–116, 2005.
[19] LieberLieber, “LemonTree 3.0.” Accessed: Dec. 18, 2025. [Online]. Available: https://www.lieberlieber.com/en/lemontree-3-0-2/
[20] A. Pierantonio, “Model differences in the eclipse modeling framework,” *UPGRADE Eur. J. Inform. Prof.*, Apr. 2008, Accessed: Dec. 15, 2025. [Online]. Available: https://www.academia.edu/2953239/Model_differences_in_the_eclipse_modeling_framework
[21] “CATIA MAGIC Documentation.” Accessed: Dec. 18, 2025. [Online]. Available: https://docs.nomagic.com/
[22] D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige, “Different models for model matching: An analysis of approaches to support model differencing,” in *2009 ICSE Workshop on Comparison and Versioning of Software Models*, Vancouver, BC, Canada: IEEE, May 2009, pp. 1–6. doi: 10.1109/CVSM.2009.5071714.
[23] M. Alanen and I. Porres, “Difference and Union of Models,” in *«UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications*, vol. 2863, P. Stevens, J. Whittle, and G. Booch, Eds., in Lecture Notes in Computer Science, vol. 2863. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 2–17. doi: 10.1007/978-3-540-45221-8_2.
[24] R. Reddy, “Model composition-a signature-based approach,” *Asp. Oriented Model. AOM Workshop*, Oct. 2005, Accessed: Dec. 17, 2025. [Online]. Available: https://www.academia.edu/100009057/Model_composition_a_signature_based_approach
[25] Z. Xing and E. Stroulia, “UMLDiff: an algorithm for object-oriented design differencing,” in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, Long Beach CA USA: ACM, Nov. 2005, pp. 54–65. doi: 10.1145/1101908.1101919.
[26] “MOF 2.5.1.” Accessed: Jan. 13, 2026. [Online]. Available: https://www.omg.org/spec/MOF
[27] “Enterprise Architect 17.1 Documentation.” Accessed: Jan. 13, 2026. [Online]. Available: https://sparxsystems.com/enterprise_architect_user_guide/17.1/welcome/index.html
[28] “Papyrus.” Accessed: Dec. 18, 2025. [Online]. Available: https://eclipse.dev/papyrus/
[29] “Syside.” Accessed: Dec. 18, 2025. [Online]. Available: https://docs.sensmetry.com/
[30] “SysML v2 Viewer | Tom Sawyer Software.” Accessed: Jan. 14, 2026. [Online]. Available: https://www.tomsawyer.com/sysml-v2-viewer
[31] “ANTLR.” Accessed: Mar. 19, 2026. [Online]. Available: https://www.antlr.org/

[32] “GPT-5.4 Model | OpenAI.” Accessed: Mar. 19, 2026. [Online]. Available: https://developers.openai.com/api/docs/models/gpt-5.4