SysIDE: SysML v2 textual editing & analysis system – overview and applications

Juozas Vaicenavičius[1⋆], Tilo Wiklund[2], Daumantas Kavolis[1], Simonas Draukšas[1], Antanas Kalkauskas[1], and Rimantas Vaicenavičius[1]

[1] Sensmetry Lithuania
[2] Sensmetry Sweden
https://sensmetry.com/

Abstract. SysIDE is an open-source SysML v2 textual editing and analysis system that provides a convenient and effective textual editing experience. SysIDE was developed by Sensmetry as SysML v2 language support extension for VS Code. SysIDE can be used immediately as a standalone application, providing SysML v2 textual syntax analysis capabilities, or it can be integrated into larger software projects. SysIDE editor extension has already been successfully used for prototyping space applications involving failure analysis and fault detection, isolation, and recovery (FDIR) modelling. SysIDE also proved to be useful in providing expressive SysML v2 language support for systems engineering tools of broader functionality. SysIDE is conveniently accessible as an extension of VS Code, a popular development environment. The paper provides details on SysIDE use cases, functionality, and experience gained through specific applications.

Keywords: SysML v2, MBSE, LSP, SysIDE
Workshop Objectives: Optimising MBSE Applications and Sustainability in Space Mission Design, Enhancing Downstream Applications and Collaborative MBSE Efforts

1 Introduction

SysML v2 [10], a new standard of systems modelling language, has introduced a standardised textual syntax alongside the improved graphical syntax of SysML v1. The addition of the textual representation creates new use cases and induces a new demand for textual SysML editors. A SysML v2 Pilot Implementation [5] was implemented by the SysML v2 Working Group to let engineers play around with the new textual syntax. However, from testing the Eclipse Pilot Implementation, it quickly became apparent that it was only meant as an introduction, since the tool lacks the functionality and performance to be an effective SysML v2 tool for real-world systems engineering applications the company needed. This led to the dilemma, how can Sensmetry be an early adopter of SysML v2 textual syntax without appropriate tooling available neither paid nor free?

To resolve the dilemma without holding back, Sensmetry started the development of SysIDE – a SysML v2 textual editing and analysis system. To facilitate faster adoption of SysML v2 and understanding that the situation Sensmetry faced was not unique in the industry, Sensmetry decided to make SysIDE open-source, releasing it on 26 January 2023 and

⋆ Corresponding author: juozas.vaicenavicius@sensmetry.com

supporting it since then. For systems engineers, SysIDE provides a convenient and effective textual editing experience when used as a SysML v2 language support extension for VSCode (the most popular development environment). Moreover, SysIDE can be used as a standalone application, providing SysML v2 textual syntax analysis capabilities that can be integrated in other applications. Together with VS Code, SysIDE aims to serve as an Integrated Developer Environment (IDE) for SysML v2. This explains the origin of the name SysIDE standing for Systems Integrated Developer Environment.

SysIDE is available to download for free from the VS Code marketplace [4] and Open VSX Registry [3]. SysIDE's source code can be found on GitLab [12].

2 *SysIDE* Overview

SysIDE [12] is an open-source SysML v2 textual editing and analysis system. Currently, it comes as an extension for VS Code that provides SysML v2 language support. Together with VS Code, SysIDE becomes an Integrated Development Environment, with functionality that would be expected by a software engineer when working with code. Examples of such functionality are:

- Semantic highlighting – different symbols are highlighted in different colours, which makes the distinction between different element meanings easier and quicker even if the elements are syntactically similar. This feature is especially useful for SysML v2 due to its large number of different element types.
- Auto completion – suggestions for keywords and symbol references are shown automatically to the user as they type.
- Code navigation – references to the selected symbol can be easily navigated to by using a keyboard and mouse shortcut, e.g. `Ctrl+Click` on Windows/Linux and `Cmd+Click` on MacOS.
- Formatting – automatically formats the text file according to prescribed rules, such as the amount of indentation. This ensures that a consistent style is maintained even when multiple people work on the model.
- Syntax validation – syntax is validated on every keystroke, giving real-time feedback.
- Reference search – search for explicit symbol usages and display every location where that specific symbol is used.
- Folding – file regions can be folded and hidden away, e.g. long documentation or comment blocks, letting users focus on what is important.
- Document symbols – displays named symbols in the currently open document, which may be replaced by the model tree in the future.
- Renaming – renaming a symbol also renames its textual references to maintain the same model structure similarly to how graphical editors maintain element usages when editing.
- Documentation on hover – hovering over a symbol name displays associated documentation, owned or inherited, removing the need for separate documentation in other places.
- Semantic validation – models are checked on every edit using the standard SysML v2 validation rules to give near real-time feedback on the model.

All this functionality greatly decreases the time needed to create, understand, and navigate through the model. Additionally, by checking model's syntax and semantics in near-real time, common mistakes can be caught immediatelly.

(a) Semantic highlighting. (b) Documentation on hover.

Fig. 1: Language server features in action.

Additionally to the VS Code extension, SysIDE can be used as a standalone language server. Language servers are independent applications providing language features such as auto completion, going to definition, finding all references, and other standard features found in code editors. They are usually separate from any development tool using it, and instead the communication between the server and the development tool happens through JSON-RPC [1] protocol using a standard Language Server Protocol (LSP) [9].

It takes a significant amount of effort to implement each of the language features found in code editors. Before LSP was standardised, each feature would need to be reimplemented for each different code editor or development environment due to non-standard APIs. Now, LSP lets the developers implement the features only once and easily deploy them in many different code editors, development environments, or even integrate them with other processes.

Due to the wide adoption of LSP, integration of SysIDE with tools other than VS Code is simple, and contributions to enable this are very welcome.

3 Example *SysIDE* Use Cases

We present two use cases of how SysIDE could be applied. In the first use case, we outline how SysIDE could be used to model a cyber-physical system and perform reliability analysis. The second use case illustrates how SysIDE can be integrated into and used by other software products.

3.1 Using *SysIDE* for reliability engineering

We at Sensmetry focus on system safety and reliability, and SysIDE was born from the need to reduce costs associated with carrying out safety and reliability analyses, such as SOTIF [7] and Failure modes, effects, and criticality analysis (FMECA) [8]. We believe that adopting model-based engineering is the main force that can reduce the costs of said analyses. Therefore, we

devised an internal project to perform FMECA analysis and define a fault detection, isolation, and recovery (FDIR) strategy for a simplified EPS system of a satellite in SysML v2 textual syntax, which required the development of SysIDE as the Pilot Implementation did not meet our requirements.

To perform FMECA analysis, we chose to follow the ECSS-Q-ST-30-02C standard [8]. First, we needed to specify physical and logical architectures. Physical and logical components were specified making extensive use of SysML v2's `part` element. We used `action` elements to specify functions of the logical components. The next step was to perform FMECA analysis for the logical and physical architectures. At first, we tried using the existing RAAML [11] standard by OMG to define the FMECA structure in SysML. However, we discovered an incompatibility between the ECSS and RAAML standards: ECSS asks for the *Failure Mode* to become a *Failure Effect* of a lower level failure, and such inheritance is not possible in RAAML, since *Failure Mode* and *Failure Effect* are two different entities. This led to us developing our own SysML v2 FMECA library, which created a *Failure Type* entity that could become a *Failure Mode*, *Failure Effect*, or *Failure Cause*, depending on the context. These were all combined into *FMECA Items* (FMECA table rows) and linked to physical and functional architectures.

```
#fmecaItem item 'L1.1.2_F1_FM4_FC1' : FMECAItem {
    :>> function = 'L1'::'L1.1'::'L1.1.2'::'F1';
    :>> impactedSubsystem = 'L1'::'L1.1';
    :>> missionPhases = ('Wakeup', 'Deployment', 'Operations', 'EOL');
    :>> failureMode = 'L1.1.2_F1_FM4';
    :>> cause = 'L1.1.2.1_F1_FM4';
    :>> finalEffect = 'L1.1_F1_FM1';
    :>> CN = 3;
    :>> isCritical = false;
}

#fmecaItem item 'L1.1.2_F1_FM4_FC2' : FMECAItem {
    :>> function = 'L1'::'L1.1'::'L1.1.2'::'F1';
    :>> impactedSubsystem = 'L1'::'L1.1';
    :>> missionPhases = ('Wakeup', 'Deployment', 'Operations', 'EOL');
    :>> failureMode = 'L1.1.2_F1_FM4';
    :>> cause = 'L1.1.2.3_F1_FM2';
    :>> finalEffect = 'L1.1_F1_FM1';
    :>> CN = 6;
    :>> isCritical = true;
}
```

Fig. 2: Example of FMECA in SysML v2 using SysIDE.

Following FMECA, we developed the FDIR strategy to observe and mitigate the identified system failures. We are not aware of any standardised ways to define the FDIR strategy and logic in SysML. Therefore, we developed our own SysML v2 library to encode FDIR information. We can define existing signals and their types, both measured by sensors and synthesised from other signals, signal transformations, diagnostic logic and configuration. We also developed an internal FDIR validator for SysIDE to report errors and warn about potential issues during editing for specification mistakes such as invalid thresholds in configuration, mismatched signal types, usage of non-existent signals, and FMECA items without specified related diagnostics.

One of the biggest advantages that we found of using SysML v2 textual notation was the ease of version control. A common way in the industry is to use Excel sheets to specify reliability engineering information. However, Excel sheets are known to be difficult to version control through Git. SysML v2 textual notation, which is similar to programming languages, works with Git and Git providers out-of-the-box. This enables us to review changes directly

in Merge Requests without the need for more complicated versioning tools and procedures. Additionally, this makes version history much more transparent and traceable.

Another advantage of using SysML v2 with SysIDE compared to Excel is a simplified data export. For any Excel sheets, one would need to write custom parsers that would parse the unstructured data into a more useful format that could then be used in other tools. SysIDE, on the other hand, is already a parser that internally builds an Abstract Syntax Tree (AST) of the model, which can then be easily searched, filtered and manipulated. We have developed another internal extension for SysIDE that can export JSON, YAML, and CSV files with user-specified data from the SysML v2 model. This extension allowed us to carry out our FMECA and FDIR activities in SysML v2 and then export them to JSON files that can be consumed by our other tools that do not yet have a SysML v2 interface.

3.2 Integration of *SysIDE* into other SysML v2 tooling

In the previous use case, we used the SysIDE extension together with the VS Code editor to create, modify, and validate SysML v2 models. However, using VS Code is not a requirement if SysIDE functionality is needed. Since SysIDE uses a standardised Language Server Protocol (LSP) to communicate with other processes, it can easily integrate with other tools. Combined with Eclipse Public License 2.0 of SysIDE, it is the perfect candidate to be integrated into any other tool that needs to interact with SysML v2's textual notation in any way.

One such tool in development is SysON [6], an ambitious open-source project aiming to provide a rich web-based visual SysML v2 editing experience as well as achieve interoperability with existing established open-source tools Capella and Papyrus. As part of the 2024 roadmap, the SysON project aims to provide both visual and textual editing capabilities of the same SysML v2 model. 2024.03 release of SysON introduced the feature of importing textual SysML v2 files into SysON graphical editor utilising SysIDE, an example of a textual model imported into SysON is shown in fig. 3. While the lead SysON project developer OBEO focuses on visual editing functionality, SysML v2 textual syntax as well as editing support is to be provided by SysIDE. Sensmetry and OBEO, the lead developers behind SysIDE and SysON projects respectively, are closely coordinating their efforts in an effort to bring a convenient and effective SysML v2 editing experience to all – free and open-source.

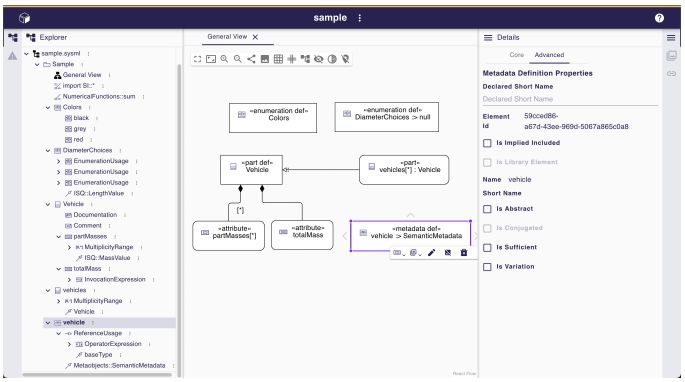

Fig. 3: Example model in 2024.03 SysON release imported from textual file in fig. 1a.

We strongly encourage other developers to also consider integrating SysIDE in their SysML v2 tooling. It will save a lot of development effort, reduce time-to-market, and will support

the interoperability between SysML v2 tools. Additionally, since SysIDE is open-source, any company or an individual developer is very welcome and invited to contribute to the development of SysIDE, thus contributing to the quality and transparent SysML v2 ecosystem to benefit model-based systems engineering.

4 Learnings

Our main motivation for developing SysIDE was to improve SysML v2 parsing performance compared to the Pilot Implementation in Eclipse. While SysIDE provides a significant performance improvement, it is still not at the level we would like it to be and it limits the size of the models we can comfortably work with. This is mostly due to us using Langium [2], a TypeScript-based parser generator. This generator also struggles with some more ambiguous SysML v2 textual syntax constructs. Additionally, since SysIDE is still early in its development, it is lacking some features that we would like to see, such as showing the model tree on the side, and a table-like editor. Lastly, we need to improve the conformance of our internal model with SysML v2 specification.

While working on SysIDE, we have had a chance to deepen our understanding of SysML v2, and we have found some drawbacks that make SysML v2 harder to reason about and maintain. One of the biggest ones is the possibility of cyclical inheritance and cyclical source file dependencies. This is in contrast to most programming languages which usually forbid both of these cycles since they easily lead to unstructured, hard-to-maintain, and difficult to understand as a whole source code. Another drawback is the large proliferation of syntactic sugar, which allows expressing the same meaning in multiple different ways, but it complicates the analysis and parsing of the textual syntax. Lastly, one feature that is absent from SysML v2 is the ability to easily assign static, and easily user-accessible unique IDs to model elements. Unique IDs would greatly simplify the linking of model data.

We have also had SysIDE users from outside of Sensmetry provide us with feedback. In general, SysIDE users leave positive feedback. However, two features are being asked for the most: the display of the model tree, and REST API implementation.

5 Conclusions and Next Steps

While SysIDE is still in early development, it has already been tried out on a more complex model than the toy examples given with the SysML v2 standard. Additionally, it is already being used as part of another SysML v2 tool and has been tried out by people outside of Sensmetry, who have supplied relevant and much needed feedback.

As part of SysIDE development work, we have identified several issues with the SysML v2 standard and we are addressing those through being active members of OMG's Systems Modeling Community Organisation that is responsible for the further evolution of SysML v2 standard.

SysIDE will continue to be developed, with the current focus areas being better performance, better conformance with SysML v2 standard, and implementation of more Language Server Protocol features. SysIDE is available to download for free from the VS Code marketplace [4] and Open VSX Registry [3]. SysIDE's source code can be found on GitLab [12].

Acknowledgements

We would like to thank the OBEO team working on SysON for the provided feedback.

References

1. JSON-RPC 2.0 Specification, https://www.jsonrpc.org/specification
2. Langium, https://langium.org/
3. SysIDE on Open VSX Registry, https://open-vsx.org/extension/sensmetry/sysml-2ls
4. SysIDE on VS Code Marketplace, https://marketplace.visualstudio.com/items?itemName=sensmetry.sysml-2ls
5. SysML v2 Pilot Implementation, https://github.com/Systems-Modeling/SysML-v2-Pilot-Implementation
6. SysON, https://mbse-syson.org/
7. ISO 21448:2022, Road vehicles — Safety of the intended functionality (Jun 2022)
8. ECSS: Space product assurance: Failure modes, effects (and criticality) analysis (FMEA/FMECA) (Mar 2009)
9. Microsoft: Language Server Protocol Specification – 3.17, https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/
10. OMG: OMG System Modeling Language, https://www.omg.org/spec/SysML/
11. OMG: Risk Analysis and Assessment Modeling Language, https://www.omg.org/spec/RAAML/1.0/About-RAAML
12. Sensmetry: SysIDE GitLab repository, https://gitlab.com/sensmetry/public/sysml-2ls