Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Thesis for the Degree of Master of Science (60 credits) in Computer Science with Specialization in Software Engineering - 15.0 credits

GENERATING SYSMLV2 MODELS FROM STRUCTURED NATURAL LANGUAGE

Petar Đukić
pdc22001@student.mdu.se

Examiner: Federico Ciccozzi
Mälardalen University, Västerås, Sweden

Supervisors: Malvina Latifaj,
Robbert Jongeling
Mälardalen University, Västerås, Sweden

June 14, 2025

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my wonderful supervisors, Malvina Latifaj and Robbert Jongeling. Their patient, insightful guidance, and attention to detail helped shape this work from its earliest ideas to its final form.

Additionally, I thank Jagadish Suryadevara for providing the necessary feedback, crucial to the outcome of this work.

Furthermore, I would like to thank Mälardalen University and my home institution University Mediterranean for granting me the opportunity to embark on this unforgettable and rewarding educational journey.

Lastly, I would like to thank my friends and family for being understanding and supportive throughout this entire experience.

Abstract

Model-based systems engineering (MBSE) is increasingly gaining momentum in industries that are developing and managing complex systems, such as Volvo Construction Equipment (VCE). Transitioning from informal architecture documentation, such as ad-hoc sketches or diagrams, remains a challenge, especially due to steep learning curve and complexity of modeling languages such as SysMLv2. This study explores how a natural language template can help domain experts intuitively capture system architecture information while enabling transformation of these templates into formal SysMLv2.

To achieve this, two artefacts were introduced: a natural language template and an LLM-driven transformation pipeline. The natural language template was iteratively evaluated and refined through a collaboration with a company representative, ensuring that it is both intuitive and sufficiently structured. The transformation pipeline uses the provided template as input and generates SysMLv2 code that is then rendered into a graphical representation of the system. The results show that the template successfully captures system architecture information while maintaining informality and intuitiveness. The transformation pipeline consistently generates SysMLv2 models, albeit with a missing variability concept due to the current limitation of the LLM training on SysMLv2 syntax.

This work demonstrates the feasibility of bridging the gap between informal and formal modeling, lowering the entry barrier to SysMLv2 and enforcing MBSE practices by enabling a flexible modeling approach.

Table of Contents

1. Introduction

Model-based systems engineering (MBSE) is increasingly becoming the industry standard for the development of complex systems and the management of their design [1]. The growing complexity of modern systems can often exceed the cognitive limits of their designers. This calls for evolution of modeling practices so that they can meet the demand of understanding and managing such systems [2]. By transitioning from document-centric to model-centric approach, MBSE improves traceability and integration throughout the design, analysis, and verification process of a system [3]. MBSE improves communication between stakeholders and addresses issues associated with document-centric approaches, such as the inconsistency and data inaccessibility often seen in document-based workflows [4].

Volvo Construction Equipment (VCE) [1] is exploring MBSE to improve modularity and reusability in its machine designs. However, the transition to formal modeling poses some challenges. Currently, in companies that are developing complex systems, a common practice is using a combination of informal graphical models that are being created in MS Visio[2], complemented by data stored in MS Excel[3] sheets. Despite the intuitive appeal of this informal documentation, these practices lack the rigor, a well-defined syntax, and semantics required in the later stages of development, and do not represent formal models. This means that these informal diagrams could not be parsed or checked by tools and automated analyses, resulting in errors remaining until very late in the development. Instead, the interpretation of informal models relies primarily on the implicit knowledge of engineers, making it more difficult to enforce consistency between spreadsheets and specifications, or parallel diagrams (diagrams created by different engineers, describing different views or parts of the same system). Additionally, the standardization becomes hindered as some domain experts document system architecture in a different way than others, making it more difficult to adopt MBSE practices.

A key aspect of MBSE is to capture system information in models conforming to a modeling language. However, a potential challenge in adopting these languages could be their complexity and learning curve, in particular for domain experts who lack expertise in these languages. One such language is the newly developed systems modeling language SysMLv2 [5], the successor to SysML, which is the de facto standard for the modeling of complex systems in the industry [6]. SysMLv2 introduces notable advancements in formal modeling, with one of the most relevant for the purpose of this thesis being the support for both graphical and textual notations. However, to directly define SysMLv2 models, domain experts require knowledge of the language's syntax, semantics and underlying principles. This is significant as domain experts already possess extensive expertise in their field, so making SysMLv2 accessible to them could make sure that different designs could be combined into one coherent system overview with far less rework. Different subsystem models using the same SysMLv2 format from the start could be automatically merged.

In this thesis, the aim is to bridge the gap between informal documentation and formal modeling by developing and leveraging natural language templates in combination with Large Language Models (LLMs) to convert these templates into formal SysMLv2 models. The templates are predefined formats, acting as a tool for capturing necessary system architecture information in a way that is intuitive for domain experts, while maintaining consistency. In this way, we aim to reduce the learning curve normally required to learn SysMLv2.

The rest of the thesis is structured as follows. Section 2. summarizes the concepts that support this work, such as Model-Based Systems Engineering, informal modeling, SysMLv2, language workbench technology and large language models, providing the technical vocabulary used in this study. Section 3. discusses research done related to the topic of bridging informal and formal modeling. Section 4. defines problem statement and presents the research questions aimed to be answered by this study. Section 5. details the adopted constructive research methodology. Section 6. reports on the implementation and its outcomes. Section 7. discusses the findings and contribution of the study, interprets the results with respect to the research questions and discusses the encountered limitations and the potential broader applications of the study. Section 8. concludes the study and outlines possibilities for future research.

[1] VCE: https://www.volvoce.com
[2] Microsoft Visio: https://www.microsoft.com/en/microsoft-365/visio/flowchart-software
[3] Microsoft Excel: https://www.microsoft.com/en-us/microsoft-365/excel

2. Background

The following section provides more information on concepts and tools that underpin this thesis.

2.1 Model-Based Systems Engineering (MBSE)

International Council on Systems Engineering (INCOSE) has defined model-based systems engineering as *"formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases"* [2]. MBSE is expected to replace the document-centric approach that has been used by companies so far, and instead integrate the model-centric approach that would be fully integrated into the system engineering process.

2.2 Systems Modeling Language v2 (SysMLv2)

Object Management Group (OMG) defined SysML as a *"general-purpose modeling language for modeling systems that is intended to facilitate a model-based systems engineering (MBSE) approach to engineer systems"* [7]. SysMLv2[4] brings forth many improvements, aiming to provide enhanced modeling aid to systems engineers. One of said improvements would be that, in contrast to SysML that supported only graphical representation, SysMLv2 supports textual notation as well. This is possible as SysMLv2 is built on top of Kernel Modeling Language[5] (KerML). The version of SysMLv2 Release used in this thesis is 2024-12.

2.3 Informal Modeling

In software engineering practice, most of the system architecture models are in fact informal, despite containing at least some elements of the unified modeling language (UML) [8, 9]. Informal modeling refers to the use of informal diagrams or ad-hoc sketches to capture and communicate software architecture concepts without adhering to formal modeling languages, such as UML or systems modeling language (SysML). These sketches and diagrams are most commonly created to design, explain, understand, but also analyze requirements [9]. Software architects often rely on informal diagrams for design and communication in early stages of development [10]. Some of the tools used for informal diagramming are the aforementioned MS Visio, Excalidraw[6], Diagrams.net[7] and others. Informal diagrams would often be accompanied by additional information stored in, for example, MS Excel spreadsheets.

Informal diagrams are flexible and easy to adopt, and are often preferred over the models with strict syntax or semantics as defined by the aforementioned formal modeling languages. However, the lack of syntax limits their use in later stages of development as architects often revisit informal diagrams to analyze the system. For example, these later-stage tasks include analysis tasks such as consistency checking or change impact analysis [10].

2.4 Flexible Modeling

Flexible modeling has emerged as a bridge between informal notation and the rigor required for MBSE. It targets the space between the two extremes: ad hoc, human-friendly, free-form sketches and the fully conformant models required for analysis and other automated steps in the later stages of development, thus combining the best of both worlds [11].

The workflow proposed in this thesis aligns with this philosophy, as it is allowing domain experts to work in a informal style, while the transformation pipeline translates their input into a formal representation on their behalf. Engineers describe an architecture in a lightweight, natural language template, being able to reorder sections, insert comments and slightly deviate outside of the template rules, and the LLM would still produce a satisfactory result. This enables low entry barrier for domain experts, yet produces a formal SysMLv2 model.

[4] SysML v2: https://github.com/Systems-Modeling/SysML-v2-Release
[5] KerML: https://www.omg.org/spec/KerML/1.0/Beta2/About-KerML
[6] Excalidraw: https://excalidraw.com/
[7] Diagrams.net: https://www.drawio.com/

2.5 Large Language Models (LLMs)

An LLM is a generative mathematical model trained on billions of parameters and training data, able to generate and comprehend human-based text [12, 13]

Currently, new models are being released rapidly, each few months bringing a new generation of models that outperforms its predecessors. For this reason, this study is not focused on benchmarking the LLM itself. Therefore, the LLM chosen for the purpose of this study is OpenAI's o3[8] model. It is OpenAI's latest model, and it pushes the boundaries in coding ability, setting new state-of-the-art results across programming benchmarks such as Codeforce and SWE-Bench, surpassing prior models.

[8] OpenAI o3: https://openai.com/index/introducing-o3-and-o4-mini/

3. Related Work

This section provides an overview of previous work conducted on the topic of flexible modeling, with the goal of formalizing informal sketches or diagrams.

3.1 Review of identified studies

Peltonen et al. [14] point out the need for flexible modeling tools that would support modeling in earlier phases of the work, in contrast to only documenting the work later on. The authors propose a Visio plug-in that maps every free-form shape onto a typed element in a shared repository. This approach would allow engineers to keep using their familiar tool while the underlying model is stored in a version-controlled database. They point out that creating a modeling tool based on already existing graphics tools is much more practical, as creating a modeling tool from scratch would take much more time.

In business transformation projects, process modeling is a key activity [15]. Mukherjee et al. [15] highlight that although free-form diagramming tools such as Visio or PowerPoint are preferred by experts, these designs are informal sketches that do not support automated analysis. In order to overcome this limitation, the authors propose idiscover, an end-to-end approach to convert informal flow diagrams to formal process models [15]. This pipeline can turn informal Visio sketches into correct Business Process Modeling Notation (BPMN) models ready for simulation and compliance checking. By formalizing the sketches, the authors can keep using lightweight tools for early designs while still benefiting from the automation later on in the process.

Sottet and Biri [16] tackle flexible modeling at language infrastructure level with JSMF. JSMF provides a lightweight proto-metamodel written as a DSL directly in JavaScript, where every attribute, cardinality, and reference type can be switched from strict to relaxed on demand. During the early stages, the rules stay relaxed, so JSMF logs any mismatches it finds, but does not block the user. In later stages, when the structure has settled, rules are set to strict, so the checker raises real errors. This follows the same "start informal, finish formal" journey our paper and the ones talked about in this section are investigating. However, unlike the previous solutions, JSMF keeps the flexibility inside of the plain JavaScript code and giving teams power to dial the formality up or down as they need to.

Gogolla et al. [17] suggest the introduction of informal elements into a formal tool, thus supporting flexible modeling. This would be done by introducing sliders that let developers adjust the level of formality of the UML and Object Constraint Language (OCL) models. In their prototype of the tool, the users can relax constraints such as attribute and object typing, role typing, multiplicities and others. This adjustment allows for temporary incomplete or inconsistent models in the early design phases. This enables faster prototyping and refinement. Later these constraints can be tightened. This approach makes it possible to work with informal models while benefiting from the formality in the later stages of development.

Finally, DeHart [18] proposes leveraging LLMs as an interface to work with SysMLv2 models through natural language. The idea is to let domain experts describe changes in plain English, which the LLM will then translate into Python code that will directly manipulate the model. Similarly to our work, the idea is to simplify the complexity and learning curve of SysML. The results of this paper suggest greatly decreased effort and complexity. The LLM acts as an intermediary that hides the technical side of the SysML API, creating a more intuitive and accessible modeling process.

3.2 Summary of findings

The reviewed studies collectively demonstrate a trend that is the gravitation toward flexible modeling, making the process more accessible and adaptable. The inclusion of formality is especially important for the later stages of development where precision, automation, and validation become critical. In the early stages, maintaining a low barrier to entry is crucial, allowing domain experts to sketch and iterate without being hindered by complex syntax. Our work aligns with this vision by bridging informal design and formal modeling in SysMLv2. By using natural language templates, the user can describe the system components and interactions in plain English, avoiding

the complex syntax of SysMLv2.

4. Problem Formulation

This section outlines the central problem addressed by this study, presents the research questions that guide the investigation, and describes the expected outcomes.

4.1 Research Questions (RQ)

This study addresses the growing need for formalized modeling in system design. Domain experts have extensive knowledge of machine development, but often lack expertise in formal modeling languages such as SysMLv2. By having natural language templates on the input side and a formal SysMLv2 code as the output, we can enable flexible modeling, capturing both informality and speed, as well as formality and standardization.

Based on the described problem, the study will aim to answer the following research questions:

- *RQ1*: How can textual templates be designed to capture system architecture information?
- *RQ2*: How can LLMs be leveraged to transform the textual templates with structured natural language inputs into SysMLv2 models?

4.2 Research Outcomes

The expected outcomes of this study are the following:

- RQ1 aims to develop a textual template that allows domain experts to capture system architecture information. With the goal of preserving flexibility, the templates should be sufficiently structured for machine parsing, but also allow free-form additions and minor deviations from the provided syntax.
- RQ2 aims to develop a transformation pipeline for transforming the textual templates with structured natural language inputs into SysMLv2 models, by leveraging LLMs.

5. Method

This study will follow a constructive research approach to investigate, develop, and evaluate a framework to generate SysMLv2 models from structured natural language inputs.

K. Lukka defines constructive research as a "*research procedure for producing innovative constructions, intended to solve problems faced in the real world and, by that means, to make a contribution to the theory of the discipline in which it is applied*" [19]. The main elements of constructive research approach are:

- It focuses on relevant real-world problems.
- It produces an innovative solution that is meant to solve the initial real-world problem.
- It attempts to implement the developed solution and evaluate it for its practical applicability.
- It implies a co-operation between researcher and practitioners.
- It is linked to prior theoretical knowledge.

Additionally, Lukka et. al. describe a constructive research typical process as follows:

1. **Find a practically relevant problem, which also has potential for theoretical contribution.**

   The practically relevant problem was already desribed in the previous sections. It carries a potential for theoretical contribution as it is exploring an intersection between structured natural language, MBSE, and AI-driven transformation of structured natural language into SysMLv2 models, which is currently insufficiently researched.

2. **Examine the potential for long-term research co-operation with the target organization(s).**

   This study is addressing the problem encountered at Volvo Construction Equipment (VCE). Domain experts will participate in the research process, as their insights into current workflows and challenges are needed. The outcome of this study aims to open the doors and provide the building blocks for the future research on the topic allowing for further co-operation with VCE.

3. **Obtain deep understanding of the topic area both practically and theoretically.**

   A comprehensive literature review is performed in order to better understand the newly developed SysMLv2, MBSE, informal and flexible modeling, and LLMs. This provides theoretical understanding needed to conduct this study. Insights of domain experts at VCE provide the required practical understanding and clear motivation for the study.

4. **Innovate a solution idea and develop a problem solving construction, which also has potential for theoretical contribution.**

   In the scope of this study, this solution consists of:

   - *Textual templates:* Textual templates are designed to capture system architecture information.
   - *Transformation pipeline:* The pipeline consists of leveraging LLMs to transform the textual templates with structured natural language inputs into SysMLv2 models.

5. **Implement the solution and test how it works.**

   After the natural language templates are developed, the pipeline will be implemented to transform the templates into SysMLv2 models. The templates will be refined and evaluated through iterations and talking to the company representative.

6. **Ponder the scope of applicability of the solution.**

   In this step, applicability and how the proposed solution can be generalized will be considered. In particular, how the developed template and transformation pipeline might be transferred to other domains or systems beyond the motivating example used for the purpose of this study.

7. **Identify and analyze the theoretical contribution.**

   The theoretical contribution lies in outlining how MBSE can be incorporated in an industrial setting without the requirement for extensive knowledge of the formal SysMLv2.

   By developing natural language templates to capture the required system architecture while still preserving informality and allowing flexibility in their usage. Additionally, the transformation pipeline will be used to generate formal SysMLv2 models.

6. Implementation and Outcomes

This section describes the implementation efforts from the design of natural language templates to the generation of SysMLv2 models. It outlines how system architecture information is captured in a flexible way and transformed into formal representations via the transformation pipeline.

The following sections (6.1-6.4) together provide a detailed description of the workflow (Figure 1), as well as its outcomes. Section 6.1 introduces a VCE system architecture example and captures core concepts needed to describe a system. Section 6.2 defines the template and describes its initial, verbose version, reasoning about its structure. Section 6.3 presents a workflow in the transformation pipeline (Figure 4) that feeds a filled out template into an LLM, converting it to SysMLv2 textual syntax. Section 6.4 reports on generated models, syntax issues, template refinement, and the flexibility of developed natural language templates.

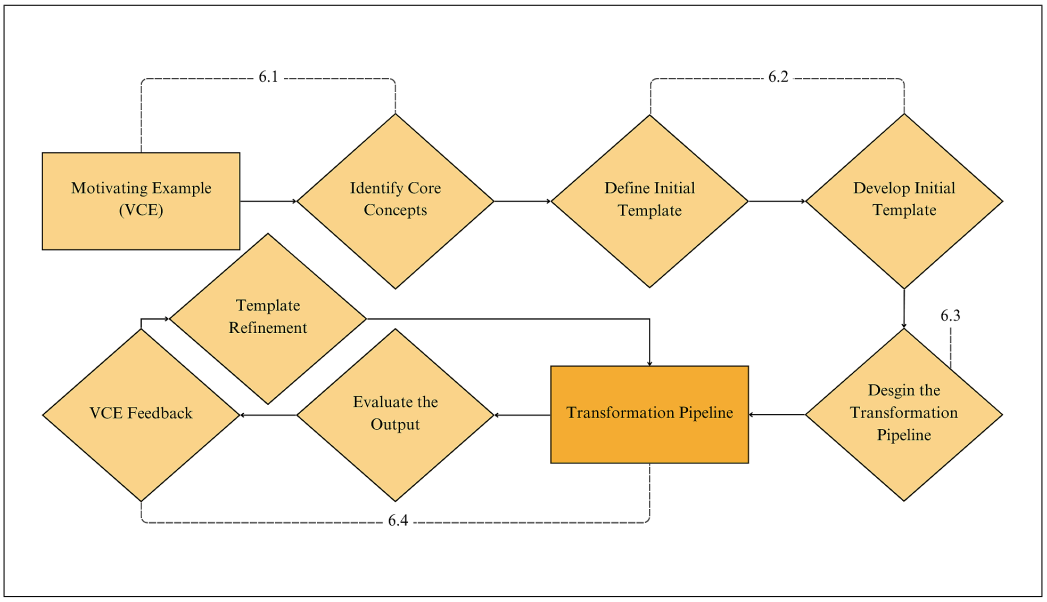


Figure 1: Visualization of workflow

6.1 Motivating Example and Identifying Concepts

To enable domain experts to capture system architecture information in a more flexible manner, this work introduces an approach that accepts informal descriptions in the form of natural language templates and generates conformant SysMLv2 models.

Before initiating the definition and development of the template, it was necessary to define core concepts (components of the system) that would be used throughout the template. A small example of a system (Figure 2) was provided by the company representative that will be used to define these core concepts. As elaborated by the company representative, this example is a scaled-down version of bigger systems that contain the same concepts, the only difference being in their number.

In the provided example, we can distinguish the following concepts:

- **Module** - A container that groups other lower-level modules and/or design units.
- **Design Unit** - An architectural building block declared inside a module. It holds a specific capability or responsibility and can be instantiated one or many times.
- **Interface** - An interaction point that in this example specifies connections between design units. The use of interfaces was clarified in the subsequent meeting with the company representative. As such, interfaces were not considered in the first iteration of the template and were only added in a later refinement.

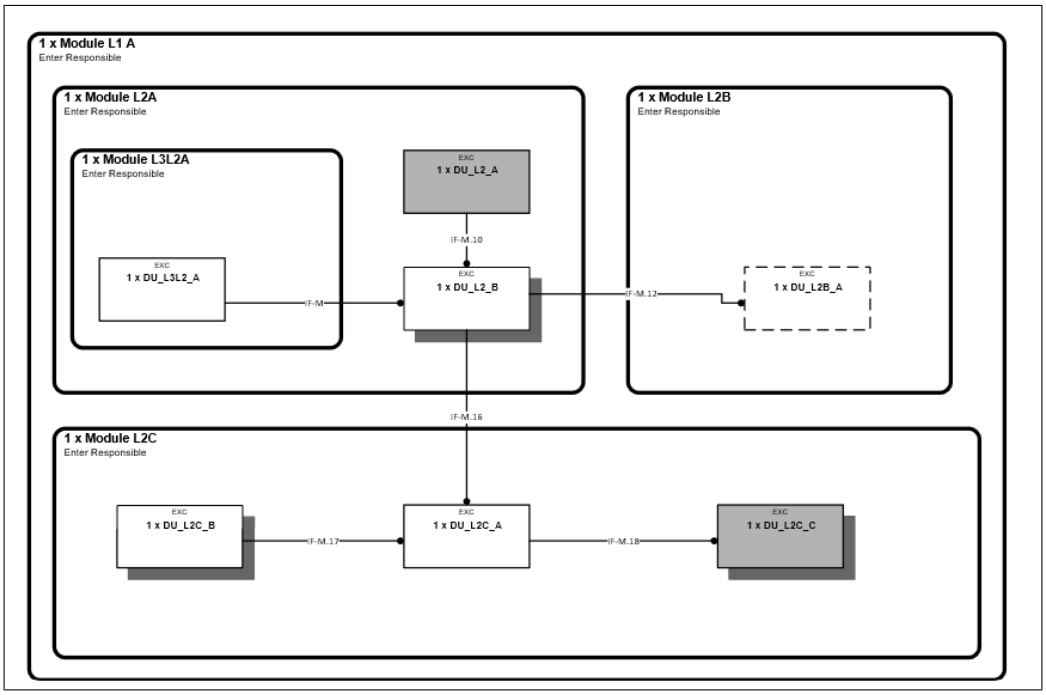


Figure 2: System Example

- **Connection** between design units - A link between two design units established through an interface.
- **Multiplicity** of design units - An attribute that describes how many instances of a design unit a module may contain (e.g. 0..1 - none or one, 1..* - one or many). In the provided example, gray boxes are optional units, whereas white boxes are mandatory.
- **Variability** of design units - An attribute that marks a design unit as a variation point. In the provided example, boxes with a shadow background are points of variability.

The same hierarchy present in this example is used in later chapters. However, the specific multiplicity or variability attribute for each design unit is chosen ad-hoc for illustration.

6.2 Initial Template Definition and Development

The following section defines the natural language template and describes the development of its initial verbose version. Additionally, it introduces the template requirements that will serve as a benchmark in later evaluations.

6.2.1 Template Definition

The structured natural language template can be defined as a short textual template that domain experts fill out instead of writing SysMLv2 code themselves. It plays a similar role as a form, having defined keywords that capture the most important architectural concepts of the system, while still leaving enough freedom in the text surrounding these keywords, preserving the informality.

Using natural language templates, system architects would be able to describe components, hierarchies, and relationships between them. This approach absolves them from having to write any SysMLv2 code themselves. These templates should be easily understood by humans while being clear enough for an LLM to transform them into their corresponding SysMLv2 textual notation, acting as a lightweight intermediate between informal sketches and formal models. The development of natural language templates was an iterative process with the goal of balancing simplicity, intuitiveness for domain experts, and machine interpretability while maintaining completeness and flexibility. These attributes have been defined as template requirements, serving as

a benchmark for a later assessment of the refined template. These requirements are the result of iterative refinement cycles, guided by evaluations from the company representative. The resulting set of requirements is presented in Table 1.

Table 1: Template Requirements

Template Requirement	**Rationale**
Completeness	All essential system concepts are initially defined and ready to be used in the template.
Simplicity	The syntax is concise, avoiding unnecessary verbosity.
Intuitiveness	Layout follows a natural language flow so the structure looks more like a description of a system than code. Domain experts can understand and fill in the template unassisted.
One-to-One Translation	Every component maps to a SysMLv2 construct, enabling automatic translation into a valid model.
Flexibility	Minor deviations (order changes, inline comments) do not break parsing, preserving informality.

6.2.2 Initial Template Development

Initially, the templates were slightly verbose, resembling natural English sentences. The idea was to be consistent in writing by repeating keywords such as "Module:" and "Design Unit:", ensuring that the LLM always saw the same pattern before each identifier. This was also thought to improve human readability. Another reason behind the initial form was that initially it was not known how flexible the LLM would be to simplified syntax and deviations from it. This verbose wording also served as a starting point. Once the LLM produced the correct SysMLv2 code, all future refinements and simplification of the template could be compared to the initial baseline output.

Initially, the only template requirements considered (Table 1) were 'completeness' and 'intuitiveness'. Interfaces were not yet introduced, variability was not the main focus, and those concepts were introduced and focused on more in the later iterations. This resulted in an evolution of the template requirements.

The initial template in its entirety can be seen in Figure 3.

The goal of the templates is to achieve balance between informality and precision, describing the system architecture using natural language, while containing enough structure to allow for one-to-one translation into valid SysMLv2 models.

As a result of an evaluation from the company representative, this structure was later found to be unnecessary, which demanded further refinement.

System Description

The system contains **Module: L1A**.

Module: L1A
Module: L1A, contains **Modules**: **Module: L2A**, **Module: L2B**, and **Module: L2C**.

Module: L2A
Module: L2A, contains **Module: L3L2A**, **Design Unit: DU_L2_A** (**multiplicity: 1..1**, **variability: no**) and **Design Unit: DU_L2_B** (**multiplicity: 0..1**, **variability: yes**).

Module: L3L2A
Module: L3L2A, contains **Design Unit: DU_L3L2_A** (**multiplicity: 1..1**, **variability: no**).

Module: L2B
Module: L2B contains **Design Unit: DU_L2B_A** (**multiplicity: 0..***, **variability: yes**).

Module: L2C
Module: L2C contains **Design Unit: DU_L2C_B** (**multiplicity: 1..1**, **variability: no**), **Design Unit: DU_L2C_A** (**multiplicity: 1..***, **variability: yes**), and **Design Unit: DU_L2C_C** (**multiplicity: 0..1**, **variability: yes**).

Connections
The following connections exist between design units:
Design Unit: DU_L3L2_A is connected to **Design Unit: DU_L2_B**.
Design Unit: DU_L2_A is connected to **Design Unit: DU_L2_B**.
Design Unit: DU_L2_B is connected to **Design Unit: DU_L2B_A**.
Design Unit: DU_L2_B is connected to **Design Unit: DU_L2C_A**.
Design Unit: DU_L2C_B is connected to **Design Unit: DU_L2C_A**.
Design Unit: DU_L2C_A is connected to **Design Unit: DU_L2C_C**.

Figure 3: Initial template

6.3 Transformation Pipeline

The transformation pipeline is designed to receive the user-provided natural language template and convert it into SysMLv2 models, leveraging the capabilities of an LLM. The process is as follows (Figure 4):

1. **Natural Language Template input:** The user provides a filled-out template with a described system architecture, describing modules, design units, interfaces, and hierarchy and connections between them.

2. **LLM:** The template is then processed by an LLM. The LLM used for the purpose of this study is OpenAI's ChatGPT o3 model. LLM then generates the SysMLv2 textual notation, ensuring that the user's informal descriptions of the system's architecture are correctly translated into a formal SysMLv2 textual notation.

3. **Textual model input into the Eclipse environment:** The LLM-generated code is then imported directly into Eclipse (at the time of writing of this thesis, version 4.31.0). This file will have a ".sysml" extension.

4. **Graphical model visualization:** Graphical visualization is then made possible with the open source PlantUML[9] tooling, which immediately renders the diagrams from the provided textual notation.

[9]PlantUML: https://plantuml.com/

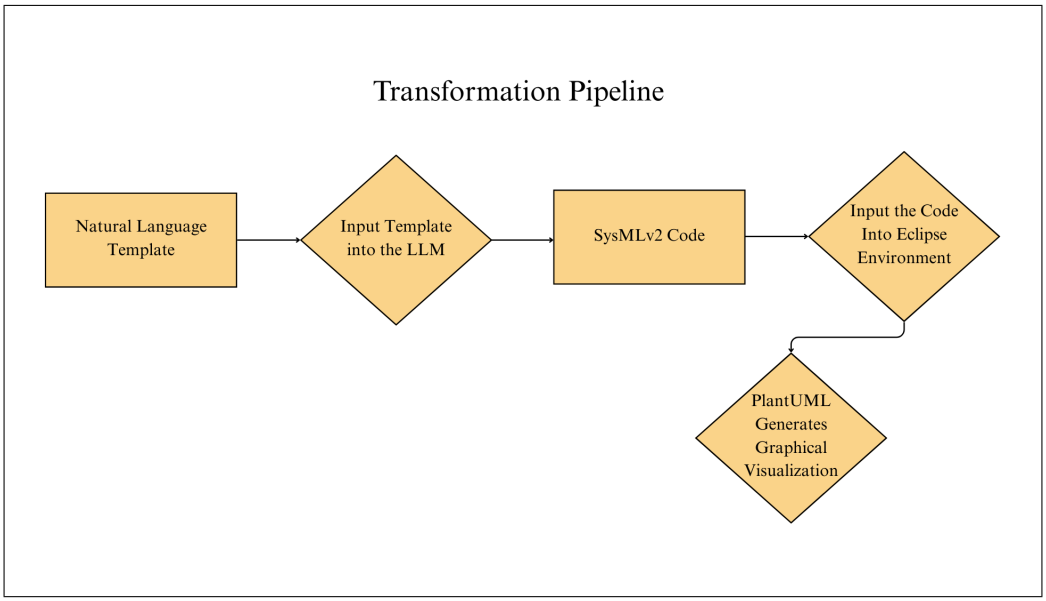


Figure 4: Transformation Pipeline

6.4 Transformation Pipeline Output & Template Refinement

The following section outlines the outcomes produced by leveraging the transformation pipeline and describes how the initial template was refined based on those outcomes. It also covers the development of the mapping of each concept to its corresponding SysMLv2 element, and briefly evaluates the flexibility of the template.

6.4.1 Initial template

The initial prompt, prior to providing the system description, was

> "Generate SysML-v2 textual notation for the following system description:
> [System Description]"

without further instructions. The generated code looked promising, correctly defining the components, hierarchy, and relationships between them. The code is presented in Figure 5. Connections, however, were not always as consistent. In some attempts, connections would be declared outside of any part, which would have resulted in paths to the design units not being recognized.

The PlantUML graphical visualization has then instantly generated the following representation of the system (Figure 6):

Although complex at first glance, the generated visualization holds true to the system description. From both textual and graphical notation it can be noticed that both modules and design units are translated into **part def** block in the SysMLv2 code. A part definition defines a new type of part (a reusable component) within the model. A **part** instance, however, creates an instance of a previously defined part definition, representing an actual part in the model. The connections between parts and part definitions can be seen in the graphical visualization.

The connections were given unique identifiers (c1, c2, c3, etc.) for clarity. The source (src) and destination (dst) are correctly identified based on the provided system description. However, the source and destination can seemingly only be seen in the code. In the graphical representation this connection is represented as a bold straight line between the two components without any directional indications.

Multiplicities described in the template were correctly translated into the code correctly. In the graphical representation, multiplicities can as well be noticed next to each component. In contrast to multiplicities, the generated code did not address variability in any way.

Furthermore, PlantUML offers different styles and diagrams when visualizing the generated code. One such diagram is the "interconnection" diagram, and it can present the Figure 6 in a way

```
package SystemModel {


    part def DU_L2_A;
    part def DU_L2_B;
    part def DU_L3L2_A;
    part def DU_L2B_A;
    part def DU_L2C_A;
    part def DU_L2C_B;
    part def DU_L2C_C;


    part def L3L2A {
        part du_l3l2_a : DU_L3L2_A[1];        // mandatory
    }


    part def L2A {
        part l3l2a     : L3L2A[1];            // mandatory
        part du_l2_a   : DU_L2_A[1];          // mandatory
        part du_l2_b   : DU_L2_B[0..1];       // optional / variable
    }

    part def L2B {
        part du_l2b_a  : DU_L2B_A[0..*];      // 0 … many, variable
    }

    part def L2C {
        part du_l2c_b  : DU_L2C_B[1];         // mandatory
        part du_l2c_a  : DU_L2C_A[1..*];      // ≥1, variable count
        part du_l2c_c  : DU_L2C_C[0..1];      // optional
    }


}
```

(a) Generated code part 1

```
    part def L1A {

        part l2a : L2A[1];
        part l2b : L2B[1];
        part l2c : L2C[1];

        connection c1 : Connection
            connect src references l2a.l3l2a.du_l3l2_a
                to dst references l2a.du_l2_b;

        connection c2 : Connection
            connect src references l2a.du_l2_a
                to dst references l2a.du_l2_b;

        connection c3 : Connection
            connect src references l2a.du_l2_b
                to dst references l2b.du_l2b_a;

        connection c4 : Connection
            connect src references l2a.du_l2_b
                to dst references l2c.du_l2c_a;

        connection c5 : Connection
            connect src references l2c.du_l2c_b
                to dst references l2c.du_l2c_a;

        connection c6 : Connection
            connect src references l2c.du_l2c_a
                to dst references l2c.du_l2c_c;
    }


    part SystemInstance : L1A;
}
```

(b) Generated code part 2

Figure 5: Initial generated SysMLv2 textual syntax

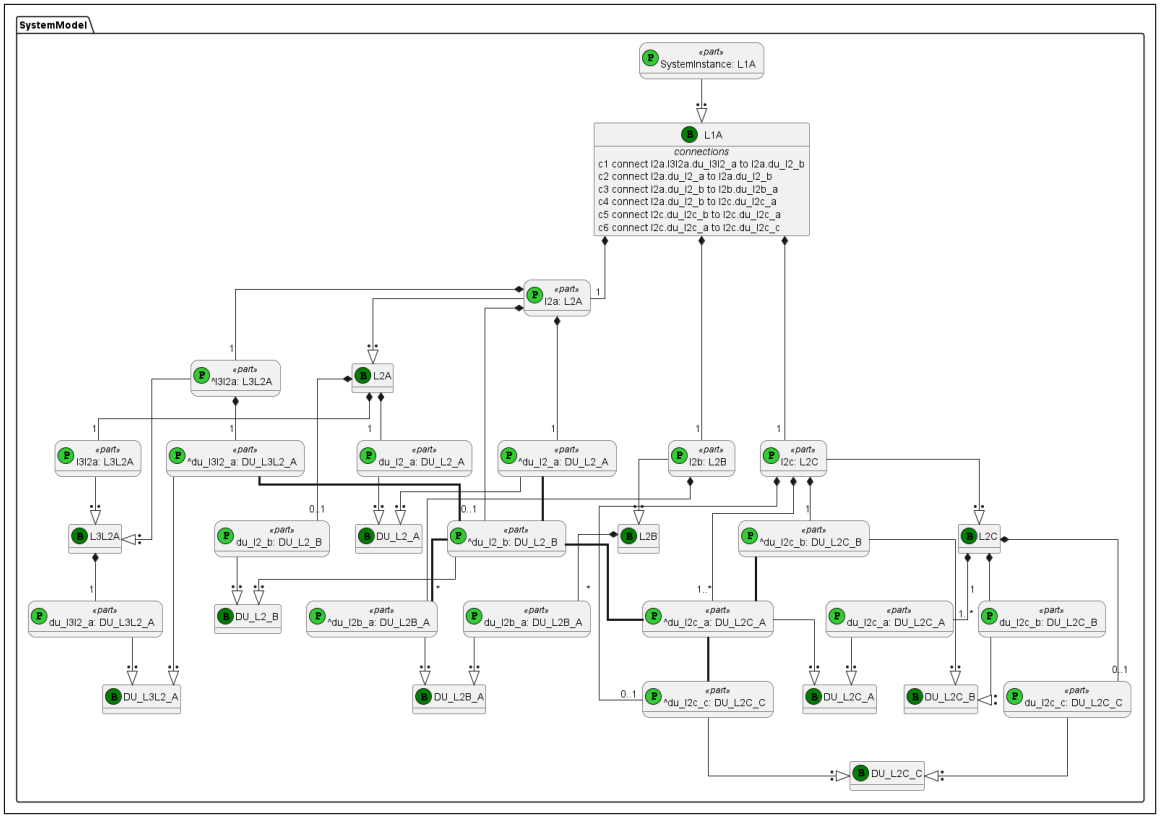


Figure 6: Initial template SysMLv2 graphical notation

that highlights the hierarchy and connections in a clearer way (Figure 7), making it more similar to the example provided by VCE.

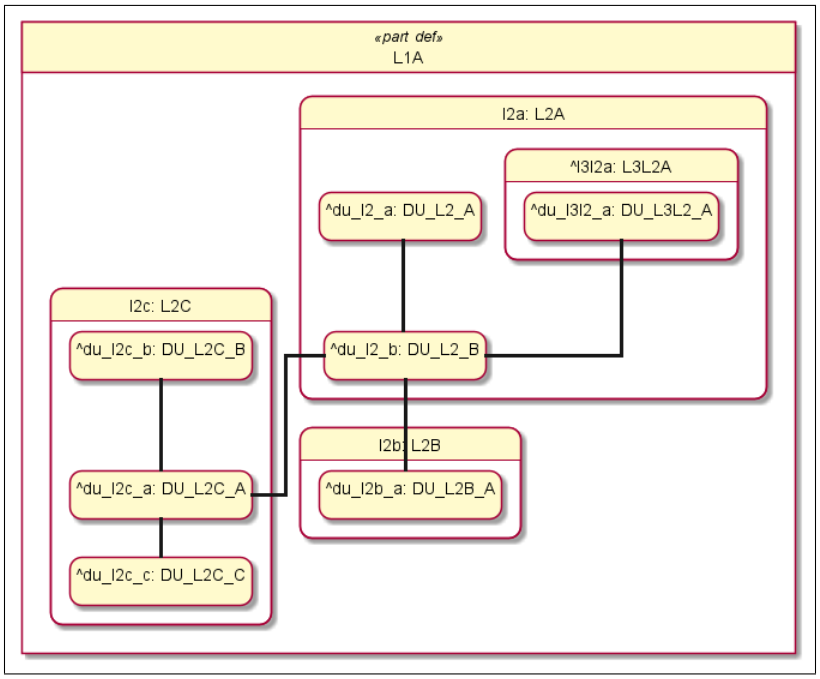


Figure 7: Interconnection diagram of the initial generated code

Lastly, the generated code has produced an error message related to the defined connections which can be seen in Figure 8. According to the official SysMLv2 Release GitHub[10], connections are defined using a specific type and need to be defined. Connection definition is a part definition whose usages are connections between its ends. If connection definition is not specified, a generic connection definition is used. In the generated code, no connections were defined, and the syntax in this case was not correct. This error highlighted the LLM's limitation in perfectly interpreting and implementing SysMLv2 syntax. Nevertheless, with further adjustment of the prompt and introduction of interfaces and mapping, the results became more consistent.

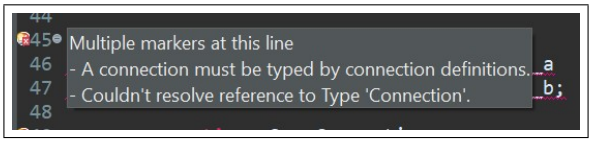


Figure 8: Initial template code error

6.4.2 Template refinement

The initial template has captured all of the necessary system information, however, the evaluation from the company representative revealed that it was too verbose and would benefit from simplification. This has led to the design of a lighter list-based structure, removing redundant text, making the templates easier to fill in while maintaining readability. Additionally, further clarification of the interface component resulted in this new addition to the next template iteration.

After refinement, the same example from Figure 3 is displayed in Figure 9. Additionally, the template in its unfilled form can be seen in Figure 10.

In this version of the template, multiplicity and variability are using a consistent and shorthand notation, avoiding redundant text. Even without any additional instruction, the LLM correctly interpreted what was written as multiplicity and variability. Furthermore, the refined template now included a dedicated section for interfaces and connections expressed by using an identifier and directionality (e.g., IF-M.10: DU_L2_A to DU_L2_B).

[10]SysMLv2 Release: https://github.com/Systems-Modeling/SysML-v2-Release/tree/master

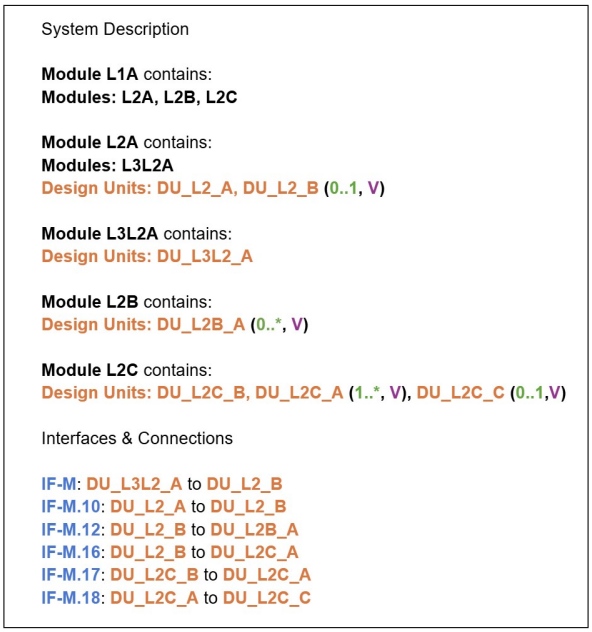


Figure 9: Template after refinement

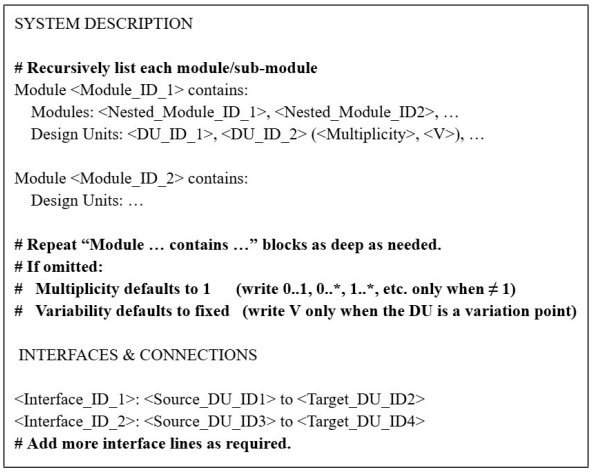
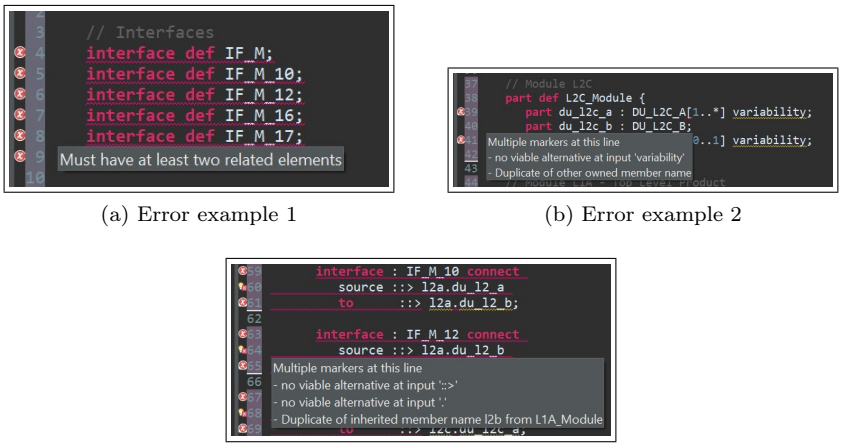


Figure 10: Unfilled template

The introduction of interfaces was not taken lightly by the LLM. Through many tries and models (o3, o4, SysML v2 codeGEN), error-filled code remained. Some of the example errors are shown in Figure 11.



(a) Error example 1

(b) Error example 2

(c) Error example 3

Figure 11: Initial errors after introducing interfaces

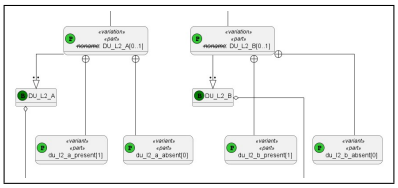
Syntax for interfaces was not correct, and the LLMs could not translate variability into SysMLv2 as well. According to the official SysMLv2 Release GitHub, an interface definition is a connection definition whose ends are ports. So, in order to use interfaces, ports need to be previously defined. To overcome this issue, a precise mapping of each component was needed, so that an example could be provided to the LLM before giving it the template.

6.4.3 Mapping of concepts

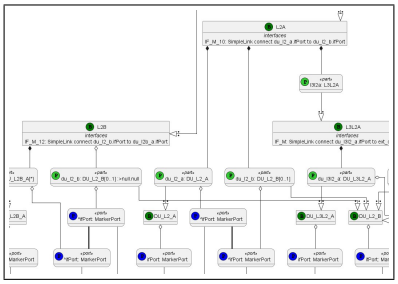
Some components, such as modules and design units, were initially generated correctly with zero-shot learning. To correctly map interfaces, few-shot learning was needed, providing minimal training data to the LLM. To provide examples and correct explanations, the GitHub page for the textual notation of SysMLv2 Release was consulted. Seeing that interfaces contain ports, which in our example are used to connect the two design units, both interface and port examples with the explanations were provided to the LLM (ChatGPT o3). Alongside these examples, the refined template (Figure 9) was provided, asking to generate the SysMLv2 code. The generated code contained appropriate syntax for the interface connections that was then used to define the correct mapping for future use. Additionally, as ports are parts of interfaces, the LLM recognized that in our example the ports did not need any attributes, since they are just connection points, so they were defined as empty.

Lastly, the remaining concept was variability. In the first iterations, with zero-shot learning, variability was never acknowledged by the LLM. The only sign of variability would be in the comment next to the design unit, which was not correct as variations do exist in SysMLv2, which was confirmed in the official documentation. Even specifying in the prompt to generate actual variants made no difference.

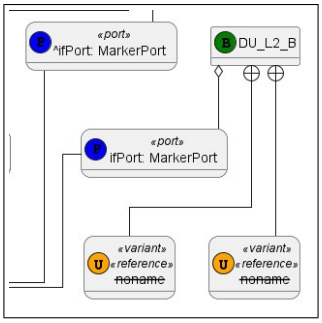
Next, we tried few-shot learning, similar to the interface. Examples and definitions from official documentation were provided. However, this did not yield any meaningful results. Additional examples were provided from the SysMLv2 Release examples and training folders that contain variation code examples. This, too, did not produce valid results. By valid results, it is meant that the syntax present in the generated code did not correspond to the encountered examples. This evaluation was performed manually. Furthermore, in the graphical visualization, the variants seemed to not be represented as in the acquired examples from the GitHub repository. Some of the issues are presented in Figure 12. In Example 1, the LLM was generating variants that were never specified and that do not make sense (absent, present). In Example 2, the generated code produced duplicates of the DU_L2_B design unit, making it a part of both the L2A and L2B modules. The design unit belonging to L2B module can be seen having null in its description. In Example 3, DU_L2_B had two variants, however, names could not be compiled. Additionally, it was effectively a 'dangling' design unit, connecting to no other components of the system. The visible ifPort is connected to only its definition.



(a) Example 1



(b) Example 2



(c) Example 3

Figure 12: Wrong variant syntax examples

As a result, the final mapping is presented in Table 2.

User-level construct / modelling feature (template keyword)	**Equivalent SysML v2 language element (example text)**
module concept	`part def Module1 { }`
module contains sub-modules	`part def Module2 { part module3: Module3; part module4: Module4; }`
design-unit concept	`part def DU1 { port ifPort : MarkerPort; }`
module contains design units	`part def Module3 { part du2 : DU2; part du3 : DU3; }`
design-unit multiplicity	`part du3: DU3 [0..1];`
interface concept	`interface def SimpleLink { end port endA : MarkerPort; end port endB : MarkerPort; }`
interface connects design units	`interface IF_M : SimpleLink connect endA ::> module1.module2.du2.ifPort to endB ::> module2.du3.ifPort;`
port concept	`port def MarkerPort { }`

Table 2: Mapping between template keywords and representative SysML v2 syntax

After introducing the mapping, the generated models were mostly correct and more consistent. An issue that would occasionally occur is that the interface connections would be emitted outside the intended top-level module (Figure 13). This presented an issue for the current mapping, as

the paths to each port would not be recognized.

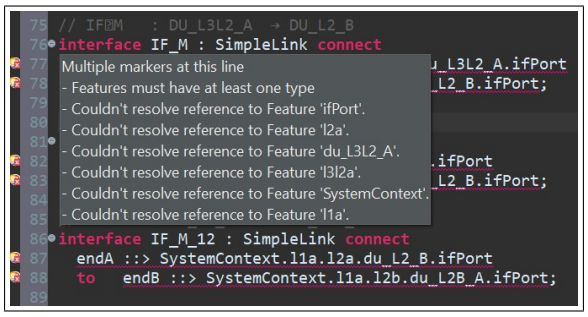


Figure 13: Interface outside of it's module

This means that each interface would need to be inside a module that hosts all of the ports used by this connection. To mitigate this issue, a prompt was introduced that establishes a few rules (Figure 14), the relevant rule to this issue being the second rule. The first rule of the prompt is there to make sure the entire code is wrapped in a package, reason being the ease of import and reuse. Other projects could import the package directly. The second rule is there to ensure consistency when defining interfaces. The third rule is there simply to prevent unnecessary comments generated by the LLM.

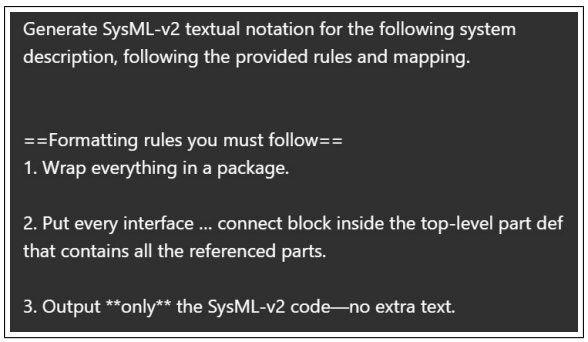


Figure 14: Prompt

The new output after the introduction of the prompt would define each connection in the module that contains the involved ports. This would sometimes generate all the connections in the top-level module (L1A), so the results would still vary.

Changing the second rule of the template to *"2. Put every interface ... connect block inside the part def that contains all the referenced parts of that connection."* made the results more consistent, defining all the connections in the first parent module that contained all the referenced parts of that connection (Figure 15).

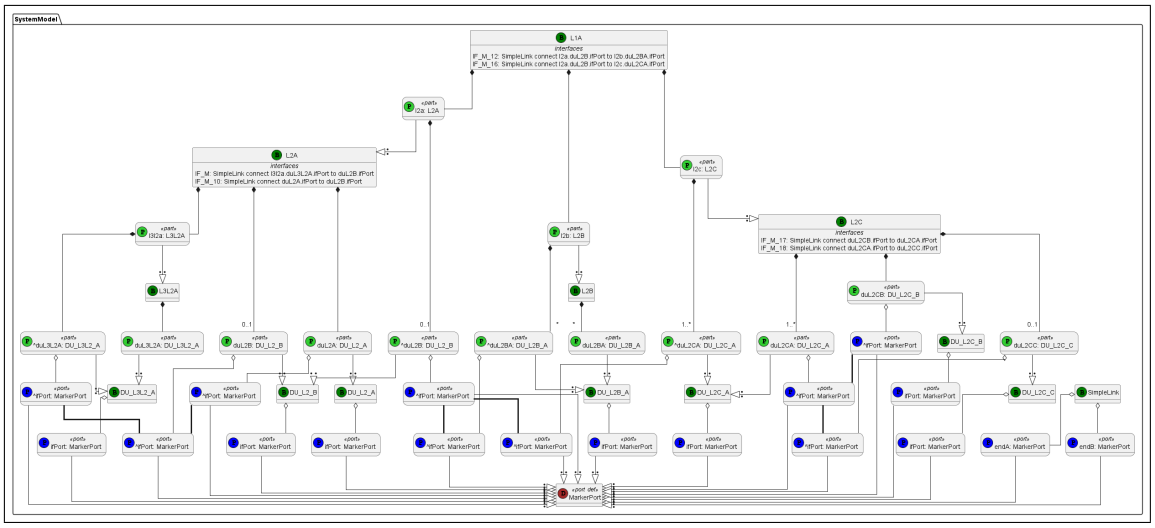


Figure 15: Graphical notation after introducing interfaces, mapping and a prompt

6.4.4 Template's flexibility

A concern during the study was avoiding making the template too strict, turning it into a sort of a domain-specific language. If every keyword, comma, or line break had to appear in the exact specified place in the template, the initial goal of preserving the freedom of informal modeling would be lost.

Therefore, several variations of deviations from the original specified template were explored. The aspects of freedom in the Table 3 were tested and the LLM provided consistent results, the results being the same as the one provided in the Figure 15. The only difference that would sometimes occur is that, as mentioned before, the interface connections would all be declared inside of the top module (L1A), and other times in the first part def that would contained the referenced parts.

Aspect of flexibility	**Example variations that parsed successfully**
Wording	*"Module L3L2A has a design unit DU_L3L2_A"* or *"Module L2A contains Module L3L2A and Design Units DU_L2_A DU_L2_B (0..1, V)"*
Order of sections	Interfaces may appear *before* module declarations.
Comments between the lines	Comments such as "This module is responsible for sensing" can be inserted anywhere.
Synonyms	*Module X has Design Unit DU_A* vs. *Module X contains a design unit DU_A.*
Punctuation errors	Missing commas or using "and" / "&" are all accepted as long as the component identifier is contiguous (DU_L2_B, DU-L2-B, DUL2B).

Table 3: Flexibility of the natural-language template

7. Discussion

This section will reflect on the work done in this study, interpreting the results and discussing the significance and limitations of those answers.

7.1 How the Research Questions Were Answered

The main goal of this thesis was to investigate how natural language templates can be designed to capture system architecture information and how these templates can be transformed into SysMLv2 models by leveraging LLMs.

1. **RQ1 - How can textual templates be designed to capture system architecture information?**

   The study showed that a small, keyword-centered template is sufficient to capture all necessary concepts needed for a generation of a SysMLv2 model. Performed iterations have reinforced the defined template requirements (Table 1), which were considered in the evaluations by the company representative. To answer this research question, each requirement had to be addressed:

   - **Completeness** - The first draft of the template did not include interfaces. The reason behind this oversight is that initially the connections in the example were simply not understood correctly. After a meeting with the company representative, usage of interfaces was clarified, explaining that design units are not simply connected with each other; in fact, it is the interface that connects them. After the refinement of the template based on this feedback, the company representative confirmed that the template contains all of the necessary concepts required to generate the appropriate SysMLv2 model, satisfying the completeness requirement.
   - **Simplicity** - The initial version of the template was intentionally verbose to ensure that the LLM generates an appropriate code. After receiving the code, we could start testing the limits to which the template can be simplified while still maintaining satisfactory output. The feedback from the company representative also pointed out the verbosity of the template, which contains redundant repetitions and words. After refinement, the template became much simpler, acquiring a list-based structure. In the subsequent meeting with the company representative, the template was evaluated as concise, satisfying the simplicity requirement.
   - **Intuitiveness** - During the second review, the company representative described the template as easy to follow, which indicated that the chosen layout is intuitive, satisfying this requirement.
   - **One-to-One Translation** - When only the modules, design units, and connections between the design units were present, the translation to SysMLv2 code was done easily with zero-shot learning. Each defined concept successfully mapped to a SysMLv2 construct. Introduction of interfaces and demanding variability created error-filled code, code that would not compile, wrong syntax. In order to satisfy this requirement, we had to perform precise mapping of each element via the few-shot learning to ensure the correct translation of each concept.
   - **Flexibility** - The refined template could have its sections reordered, switch between 'contains' and 'has', have inline comments, and have minor punctuation errors. While these were the only deviations performed, the template would still be successfully translated into SysMLv2 code, satisfying the flexibility requirement.

   In summary, the final version of the template meets the defined requirements, demonstrating how can a natural language template be designed to successfully capture system architecture information for model generation.

2. **RQ2 - How can LLMs be leveraged to transform the textual templates with structured natural language inputs into SysMLv2 models?**

   The study shows promising results in generating accurate SysMLv2 models from the provided templates. However, they must be leveraged through a prompt that provides mapping between template concepts and SysMLv2 syntax alongside the system description in the form of the developed template.

   Feeding a filled-out template to the LLM with a single instruction of generating the SysMLv2 code left the model guessing and producing incomplete outputs: interface syntax was wrong, ports were missing, variability not recognized, connections would often be defined where reference targets could not be resolved, and model would fail to compile. This shows that in its current state, the LLM (ChatGPT - o3) is still poorly trained on the newly released SysMLv2.

   Once the prompt was extended, eliminating inconsistency, and was mapping introduced, the same model produced accurate code on every run. The generated graphical representation in Figure 15 matched the motivating example in Figure 2. The only component left to be desired is the variability attribute. Repeated prompt tweaks and additional examples were enough to make the model mention variability, but the syntax or placement was seemingly not correct (Figure 12). Until the underlying LLM is exposed to more SysMLv2 training data, the variation points would likely need to be manually defined.

   Finally, as a conclusion, LLMs can indeed be leveraged to transform natural language templates into SysMLv2 models by utilizing the developed transformation pipeline. However, currently only when guided by precise mapping and a purposeful prompts.

7.2 Limitations

Some important limitations to consider that might have impacted the outcome of this work:

1. **LLM support for SysMLv2** - As a result of this study's outcome, it is assumed that ChatGPT - o3 was trained on insufficient SysMLv2 training data. Both the language itself and the LLMs are still evolving, so the latter models are likely to deliver better results; however, for now the transformation pipeline relies on explicit mapping.

2. **Narrow expert validation** - The developed templates and generated models were reviewed and evaluated by a single company domain expert. Although feedback is valuable, evaluation by multiple domain experts would further strengthen the confidence of the acquired results.

3. **Single case study** - The study revolved around the provided motivating example by the company representative, who confirmed that larger systems follow the same structural patterns. However, the approach has not been stress tested with systems comprised of hundreds of parts and interfaces.

7.3 Broader applicability

Although the study was based around the motivating example provided by VCE, the method itself is generic. The template used a defined set of concepts that have been mapped for the purpose of this study, however, these concepts could be swapped with different ones such as components of a medical device or something similar. By doing this, with minor edits, the pipeline should still work. Additionally, the approach used in this study, that is natural language templates being translated into SysMLv2 models by the LLM, could generate any other formal artefacts. Therefore, this approach could be generalized to any project that must deliver a formal SysMLv2 representation from a simple description.

8. Conclusions and future work

This study explores a way to bridge informal system architecture documentation and formal system modeling, easing Volvo Construction Equipment's (VCE) transition to model-based systems engineering (MBSE). We investigate how natural language templates could be designed to capture system architecture information, and how can they be transformed into conformant SysMLv2 models by leveraging LLMs. To achieve this goal, two artefacts were introduced:

1. **Natural language template** - The natural language template captures system architecture information, having previously defined necessary concepts such as module, design unit, interface, multiplicity, variability, connection. These concepts were derived from the provided motivating example by VCE. Iterative evaluations and refinements with the company representative confirmed that the final version of the template met all previously defined requirements (Table 1), answering our first research question. The final template is shown in Figure 9.

2. **Transformation Pipeline** - The transformation pipeline is linear and concise. The developed template is fed to the LLM, which in turn generates the corresponding SysMLv2 code. This code is then opened in the Eclipse-based SysML v2 environment, where the PlantUMLl plug-in instantly renders a graphical visualization of the architecture. To get the most accurate result, only the template was not enough. A prompt was introduced to increase consistency, and the mapping of each defined concept to its corresponding SysMLv2 element was carried out. As a result, the transformation pipeline produced working SysMLv2 code and a graphical visualization of the system, thus answering our second research question. The remaining gaps are confined to variability, as this attribute was not successfully mapped due to the LLM being still poorly trained on the newly released SysMLv2.

The two introduced artefacts meet their primary goal. Domain experts can fill out the template, feed it to the transformation pipeline, and generate the SysMLv2 model, therefore bridging the gap between informal documentation and formal MBSE artefacts. These templates have precisely defined concepts; however, they preserve their informality in their aspect of flexibility (Table 3), being able to deviate from their initial form. This study thus demonstrates and contributes to the topic of flexible modeling, allowing domain experts to work in an informal way, while the transformation pipeline translates their input into a conformant SysMLv2 model.

Although this approach has proven its functionality for a provided motivating example, there are several paths for future research that can be explored. One direction is to build on this study as there are gaps such as variability not being properly mapped. Another future work possibility is training the open-source model on SysMLv2 code examples data so that the mapping becomes less critical and results more consistent. This system example was quite small, scaling to larger architectures could be another direction for future research. Finally, transferring this approach to a different domain, such as medicine or industrial IoT networks, is also a possible future work direction.

Given that SysMLv2 is relatively new and LLM capabilities are only going to improve over time, this line of research represents a high-potential field that deserves further investigation.

References

[1] A. L. Ramos, J. Ferreira, and J. Barcelo, "Model-based systems engineering: An emerging approach for modern systems," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 42, pp. 101–111, 01 2012.

[2] I. C. on Systems Engineering (INCOSE), "Systems engineering vision 2020." [Online]. Available: https://sdincose.org/wp-content/uploads/2011/12/SEVision2020_20071003_v2_03.pdf

[3] M. Blumenfeld, E. Stewart, and E. Larios, "Using model based systems engineering to increase the robustness of monitoring systems in railways," in *7th IET Conference on Railway Condition Monitoring 2016 (RCM 2016)*. Birmingham, UK: Institution of Engineering and Technology, 2016, pp. 12 (6 .)–12 (6 .). [Online]. Available: https://digital-library.theiet.org/content/conferences/10.1049/cp.2016.1199

[4] O. Ginigeme and A. Fabregas, "Model based systems engineering high level design of a sustainable electric vehicle charging and swapping station using discrete event simulation," in *2018 Annual IEEE International Systems Conference (SysCon)*. Vancouver, BC, Canada: IEEE, Apr. 2018, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/8369606/

[5] O. Management Group (OMG), "SysMLv2 GitHub Repository," Accessed: 2025-01-10. [Online]. Available: https://github.com/Systems-Modeling/SysML-v2-Release

[6] 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 *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS Companion '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1086–1095. [Online]. Available: https://doi.org/10.1145/3652620.3687820

[7] O. Object Management Group, "OMG System Modeling Language," Accessed: 2025-01-05. [Online]. Available: https://www.omg.org/spec/SysML

[8] R. Jongeling, A. Cicchetti, and F. Ciccozzi, "How are informal diagrams used in software engineering? an exploratory study of open-source and industrial practices," *Software and Systems Modeling*, Dec 2024. [Online]. Available: https://doi.org/10.1007/s10270-024-01252-3

[9] S. Baltes and S. Diehl, "Sketches and diagrams in practice," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 530–541. [Online]. Available: https://doi-org.ep.bib.mdh.se/10.1145/2635868.2635891

[10] R. Jongeling, F. Ciccozzi, A. Cicchetti, and J. Carlson, "From informal architecture diagrams to flexible blended models," in *Software Architecture*, I. Gerostathopoulos, G. Lewis, T. Batista, and T. Bureš, Eds. Cham: Springer International Publishing, 2022, pp. 143–158.

[11] R. Jongeling and F. Ciccozzi, "Flexible modelling: a systematic literature review," *J. Object Technol.*, 2024.

[12] Google Developers, "Machine learning resources: Introduction to large language models," 2024, [Online; accessed 1-February-2025]. [Online]. Available: https://developers.google.com/machine-learning/resources/intro-llms

[13] M. Shanahan, "Talking about large language models," *Commun. ACM*, vol. 67, no. 2, p. 68–79, Jan. 2024. [Online]. Available: https://doi.org/10.1145/3624724

[14] J. Peltonen, M. Felin, and M. Vartiala, "From a freeform graphics tool to a repository based modeling tool," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*. Copenhagen Denmark: ACM, Aug. 2010, pp. 277–284. [Online]. Available: https://dl.acm.org/doi/10.1145/1842752.1842804

[15] D. Mukherjee, P. Dhoolia, S. Sinha, A. J. Rembert, and M. Gowri Nanda, "From informal process diagrams to formal process models," in *Business Process Management*, R. Hull, J. Mendling, and S. Tai, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 145–161.

[16] J.-S. Sottet and N. Biri, "JSMF: a flexible JavaScript Modelling Framework."

[17] M. Gogolla, R. Clariso, B. Selic, and J. Cabot, "Towards Facilitating the Exploration of Informal Concepts in Formal Modeling Tools," in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. Fukuoka, Japan: IEEE, Oct. 2021, pp. 244–248. [Online]. Available: https://ieeexplore.ieee.org/document/9643627/

[18] J. K. DeHart, "Leveraging Large Language Models for Direct Interaction with SysML v2," *INCOSE International Symposium*, vol. 34, no. 1, pp. 2168–2185, Jul. 2024. [Online]. Available: https://incose.onlinelibrary.wiley.com/doi/10.1002/iis2.13262

[19] K. Lukka, *The Constructive Research Approach*, 01 2003, pp. 83–101.