34th Annual INCOSE international symposium
hybrid event
Dublin, Ireland
July 2 - 6, 2024

Leveraging Large Language Models for Direct Interaction with SysML v2

John K. DeHart
AVIAN Inc.,
2211 Three Notch Road, Lexington Park, MD 20653
301-866-2070
jdehart@avian.com

Abstract. This paper examines the potential integration of Large Language Models (LLMs) with the Systems Modeling Language version 2 (SysML v2), proposing a novel methodology for systems engineering by capitalizing on the enhanced readability and human-friendly syntax of SysML v2. Given the emergent sophistication of LLMs and the coincidental development of SysML v2—an endeavor that presents a pivot toward naturally articulated model interaction—we explore the possibilities and implications of such an intersection. Our investigation posits that LLMs can serve not only as an interpretive layer, allowing for the syntactically simplified manipulation of system models, but also as a catalyst for a knowledge-driven design approach.

We highlight the efficiencies gained by deploying LLMs for SysML v2 interactions, which reduce the dependency on technical expertise traditionally needed for API navigation and model management. Through case studies and analysis, we demonstrate that the conversational engagement with system models facilitated by LLMs can lead to a democratized and accelerated design process. However, this advent is tempered by a critical awareness of potential pitfalls, such as automation bias and overreliance on automated systems—underscoring the need for continued human oversight and the examination of ethical considerations.

Emphasizing the chance of SysML v2 being inherently English-based and the parallel maturation of LLMs, this paper suggests that the collaborative utilization of these concurrent advancements may offer an opportune fusion, potentially revolutionizing the way systems are modeled and managed. Future work involves the empirical validation of these approaches and a deeper investigation into interoperability with existing and future systems engineering ecosystems. The ultimate goal is to ensure that this fusion not only complements human expertise but also propels systems engineering into a new era of innovation and holistic design.

Keywords. API, AI, Assistant, Large Language Model, LLM, MBSE, SysMLv2

Introduction

The Systems Modeling Language (SysML) version 2 (Object Management Group, 2024) marks a significant step forward in modeling capabilities for complex systems. Its revision aims to enhance usability for systems engineering practitioners by introducing more intuitive language constructs and improved model organization. SysML v2's evolution reflects an ongoing effort to address the challenges inherent in representing and understanding the multifaceted nature of systems in various domains.

However, the interaction with SysML v2 is typically intermediated by an Application Programming Interface (API), which can introduce a layer of complexity that potentially detracts from the language's user-centric design advancements. Recognizing this gap, there is a unique opportunity to use LLMs like the Generative Pre-trained Transformer (GPT) (Tom B. Brown, 2020) to create a more seamless integration into systems engineering workflows.

This paper suggests a novel approach to interfacing with SysML v2 through direct communication with LLMs. The goal is to leverage the natural language processing strengths of LLMs to read, interpret, and generate SysML v2 constructs, thus reducing the reliance on APIs for model manipulation. Such a direct interaction could simplify the model management process, leaving systems engineers free to focus on the core tasks of design and analysis.

The objective of this investigation is twofold: first, to evaluate the feasibility of using LLMs to directly interact with SysML v2 models, and second, to understand the potential benefits and limitations of this approach. We aim to provide evidence-based insights into how this integration can improve model accessibility and effectiveness within the systems engineering process.

Through the initiative detailed in this paper, we intend to contribute to the systems engineering community by exploring alternative methods for model management that increase efficiency and enhance the usability of SysML v2. We will look closely at the technical implications of this method, considering how it can fit within existing systems engineering practices and potentially lead to improved ways of model-based engineering.

The anticipated outcomes include a detailed analysis of the correspondence between LLMs and SysML v2, a practical exploration of the LLM's effectiveness in model interaction, and a discussion on the integration with other tools commonly used within the systems engineering domain. Ultimately, our study seeks to provide a substantial technical foundation for integrating advanced language models as tools into the practice of systems engineering (Estefan, 2008). The intrinsic goal of this academic endeavor is to advance systems engineering by harnessing the latest developments in artificial intelligence. This approach aims to merge human intellect with machine comprehension to augment technical intelligence and enhance system modeling and design, as discussed by (Rouse, 2020).

SysML v2 and Human Readability

SysML has long been a cornerstone in systems engineering for its ability to model complex systems across various disciplines. With the introduction of SysML version 2, the language has seen significant enhancements aimed at improving human readability, model expressiveness and interoperability with other engineering models (Object Management Group., (n.d.), 2023). These enhancements are designed to bring closer to natural language, making the models more accessible and easier to comprehend for engineers and stakeholders alike.

SysML v2's intuitive design is evident in its refined grammar and syntax, which enable a more straightforward description of system behaviors, structures, and requirements. The language is structured to be more declarative, allowing users to describe what the system in question must do, rather than how it should do it.

This shift towards a more user-friendly approach may reduce the learning curve for new users and may enhance efficiency for experienced modelers.

Given its closer alignment with natural language, SysML v2 is well-positioned to benefit from the integration with LLMs such as GPT. The ability of LLMs to understand and generate text-based inputs opens the door for directly interacting with SysML v2 models through written or typed commands and descriptions. Engineers can describe aspects of the system in plain language, and the LLM, leveraging its natural language processing capabilities, can interpret these descriptions to create or modify SysML v2 model elements accordingly.

The potential for LLMs to directly interact with SysML v2 represented systems circumvents the need for specialized API knowledge or the development of custom interfaces. This is particularly advantageous for advanced engineering teams looking for cost-effective and efficient solutions. Ingrained API development and managing vendor-specific tooling can introduce extensive resource overheads. With LLMs, engineers can focus on system design and validation, as the LLM can serve as an intermediary, translating human-readable input into accurate SysML v2 constructs and facilitating interactions with simulation models within a wider engineering toolchain.

This approach does not only promise to streamline the modeling process but also offers a scalable solution adaptable to the various complexities of systems engineering projects. It allows for a flexible and iterative design process, where engineers can rapidly prototype and evaluate changes in the system model without the iterative overhead of formal programming or script development.

Moreover, the ability to use natural language as a means to interact with models can play a significant role in collaborative environments (Nikhil Mehta, 2023). It can enhance communication between interdisciplinary teams, where the barrier of technical jargon is often a bottleneck for progress and understanding. By consolidating the interaction layer into a language that is inherently understood by all, the potential for misinterpretation is diminished, paving the way for more effective teamwork and an optimized systems engineering lifecycle.

As we venture further into the analysis of SysML v2 and LLMs, it's important to remain vigilant of the challenges this new interaction paradigm may present. Intricacies within the Human language and the precision required for modeling complex systems are factors that necessitate a careful balance to achieve the true utility and efficacy of this integration.

Challenges with SysML v2 API

The SysML v2 API represents the programmable interface through which system models can be managed, manipulated, and queried. While APIs are powerful, they often introduce a layer of complexity, in terms of difficulty involved in performing API interactions and integrating models into larger workflows, that can be a significant barrier to productivity and model traceability. Particularly with SysML v2, the challenges of working with the API stem from intricate issues such as the transition between the model's concrete syntax and the API's underlying interchange format, JSON.

A primary concern regarding this API interaction is the potential asymmetry and loss of traceability when converting from the SysML v2 model to the API representation. Once the model is published to the API any changes from the API cause the concrete and API models to become disjointed.

Additionally, the requirement for engineers to understand and navigate the API's schema to perform Create, Read, Update, and Delete (CRUD) operations imposes an additional cognitive load that detracts from systems engineering pursuits. Engineered modeling tools are often required to interact with the JSON-based representation of the models within the API directly, potentially bypassing the concrete SysML v2 language. As a result, teams may find themselves focusing more on manipulating the JSON data rather than engaging with the expressive capabilities that SysML v2 offers.

In response to these issues, we present the proposal of utilizing LLMs to conduct CRUD operations directly on SysML v2 models. LLMs, with their advanced language understanding capacities, could potentially mitigate the challenges associated with API-model asymmetry. By processing the SysML v2 language constructs directly, LLMs can maintain the lineage and integrity of the system model by avoiding lossy translations into intermediary formats.

An LLM could serve as a conduit between the engineer and the model, interfacing in a manner that preserves the readability and traceability of SysML v2. Such a system could allow for direct interaction with the model through the SysML v2 language, managing the complexity of JSON or other API-dependent structures behind the scenes.

Furthermore, exploring database integration options alongside LLMs may provide a pathway for managing large-scale SysML models. The combination of a database's structured data management capabilities with the LLM's language processing can deliver more cohesive model access and maintenance methods. Rather than manipulating JSON constructs, engineers would interact with the database through the SysML v2 language via the LLM, streamlining model evolution while ensuring accurate model representation and integrity.

This approach stresses the importance of maintaining the beauty and traceability of the SysML v2 language in system modeling. It confronts the real dilemma that any Domain Specific Language may face, where the concrete syntax risks being overshadowed—or even rendered obsolete—by the API interface. By advocating for direct interaction through LLMs, we aim to preserve SysML v2 as the bridge language of systems modeling, ensuring that the language remains central to the systems engineering process.

LLMs as a Bridge to Technological Integration

Integrating modeling languages with other technologies and programming tools is essential for full-spectrum system analysis and simulation. In the context of SysML v1.x and earlier iterations, there has been a distinct gap between system modeling and subsequent analysis and simulation. Bridging this gap effectively has been a persistent challenge, often resulting in poor integration limiting the potential for a coherent system engineering workflow (Micouin, 2019).

SysML v2's design aims to improve this integration by fostering an environment conducive to direct interaction with various technologies. LLMs have the potential to be the linchpin in this integration effort by serving as an adaptive interface that can interpret SysML v2 constructs and seamlessly generate the required "bridge wires" to other systems and programming environments. This automatic binding is particularly crucial as it reduces the time and complexity ordinarily associated with setting up integrations manually.

LLMs can function as comprehensive middleware, translating SysML v2 models directly into a form that other technologies can understand and process. For instance, a SysML v2 model describing a state machine could be interpreted by an LLM to produce executable code in programming languages such as Python or C. Similarly, the LLM could convert SysML v2 representations into formats suitable for Computer-Aided Design (CAD), expert systems, databases, and Product Lifecycle Management (PLM) systems.

An example of this potential is seen in the SysML v2 JupyterLab kernel. This tool already brings system models into direct contact with powerful simulation and querying tools. The integration of LLMs here could further enhance this synergy, enabling the automatic generation of scripts or commands needed to bridge system models with associated simulation tools. The LLM would act as a dynamic interpreter and generator, building the necessary code or queries based on the high-level system descriptions provided in SysML v2.

This level of technological integration directed by LLMs would dramatically improve the MS&A creation and integration capabilities within a system engineering environment. Where currently, the modeler must often manually establish ties to analysis and simulation software—a time-consuming and error-prone process—LLMs could automate this task, ensuring a consistent and accurate translation from model to analysis/simulation. Such a workflow allows engineers to focus more on their system's design and behavior, with LLMs handling the intricacies of interfacing with other technologies.

Methodology

The methodology for testing the integration of LLMs with SysML v2 models involves an experimental setup where the LLM interacts with system models via natural language queries and provides Python code snippets to execute data manipulation tasks. This process ensures that the systems engineering team interfaces with the model directly through natural language, aided by automation scripts generated by the LLM.

Experimental Setup

The tools involved in our experimental setup include:

- The SysML v2 language as the foundation for constructing text-based system models.
- OpenAI's Assistants (OpenAI, (n.d.), 2023) *(essentially a trainable LLM)* - Recently released for public consumption, Assistants can call OpenAI's models via an API utilizing specific instructions to tune their personalities or utilize provided files for knowledge retrieval. These Assistances are capable of processing and generating text, as well as interpreting and executing code.
- A file system, to store SysML v2 models.
- A scripting environment or a JupyterLab notebook, to interface with the LLM and execute generated code.

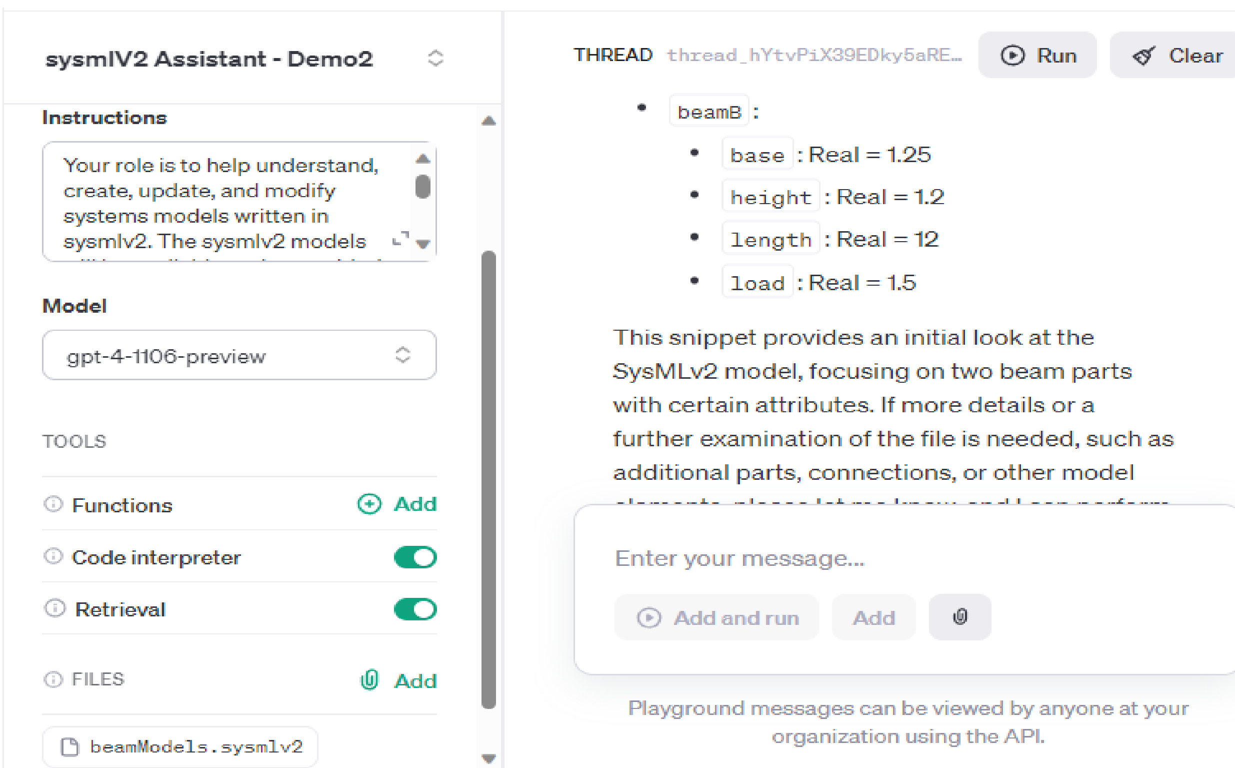

Figure 1- An example of an Assistant developed by OpenAI and used here to interrogate and modify a SysMLv2 model.

In the experiment, we utilize a pre-trained instance of an Assistant, and provide it with a SysML v2 model stored as a textual file. Through an API the Assistant is instructed to perform specific CRUD operations using the provided file as a template. These instructions are framed in natural language and correspond directly to changes that need to be applied to the model (e.g., "change the beam A length to X units"). To illustrate the direct interaction with SysML v2 models using an Assistant, we create an API client instance and upload the SysML v2 reference model file. An assistant is then set up with instructions detailing its role, and with access to the required tools for retrieval and code interpretation.

Workflow

Here is how the workflow operates:

1. An Assistant is manually created by the user in the OpenAI Playground or is automatically created by the API interaction workflow.
2. The user manually pushes a reference SysMLv2 model (or automatically by the workflow) to the Assistant. This model is immutable and acts as a training object for the Assistant. *(Note: The working model remains on the local server and may be edited by returned and executed code.)*
3. The user or another computer submits natural language requests to the Assistant to query or modify a SysML v2 model, creating and running a thread on the LLM model server.
4. Using the reference model, the Assistant processes this request thread, understands the SysML v2 syntax in the context of the reference model, and generates a textual response to a query or Python code that reflects the requested information or changes.
5. If code is returned to the local API endpoint, it recognizes the code, extracts the code from the response, and then executes this code in the user's local environment, where the working SysML v2 model files are stored.
6. If an error occurs when locally executing the returned code the error is captured and a new thread is started with the same initial request but this time including the error message. This can be repeatedly used by the Assistant to correctly modify the response until the code executes properly.
7. Any requested modifications are performed on the local SysML v2 model and any post processing may occur such as model publishing, diagraming, etc.
8. For continued work the updated local model now becomes the reference model and this reference model is now pushed to the Assistant as knowledge for any further queries or modification requests.

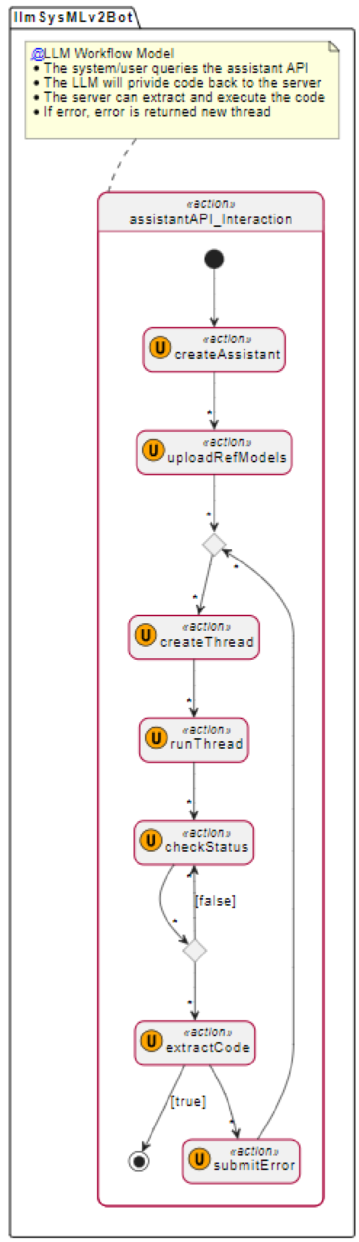

Figure 2. Workflow for executing code provided by the Assistant API.

Expanding the Workflow

This workflow can be extended by integrating the LLM's capabilities with simulation models, analytics tools, and other systems engineering applications. For instance, the LLM could provide Python code to interact with a simulation tool API, executing a simulation with parameters defined in the SysML v2 model and then analyzing the results.

Challenges and Issues

Several challenges could arise when developing such a tool:

- **Ambiguity in Natural Language Requests:** The LLM might misinterpret the engineer's intent if the natural language request is ambiguous or lacks the necessary technical specificity.
- **Syntax Preservation:** The LLM-generated code must preserve the exact syntax and semantics of SysML v2 to maintain model integrity.
- **Error Handling:** Any errors in the code generation or execution process need to be handled gracefully, with mechanisms for rollback or corrections to avoid corrupting the model.
- **Security and Privacy:** As models may contain sensitive data, ensuring security during the manipulation process is essential.

The example code is not only a demonstration of how an API-client interaction might work but also brings to light the complexity of implementing a reliable workflow for CRUD operations on SysML v2 models, utilizing the natural language processing strengths of Assistant API. In a production environment, we must pay close attention to maintain model integrity, system security, and the chase of ensuring an efficient and user-friendly interaction with the model.

It is imperative for the reader to understand that this paper's proposed workflow is instrumental in adding a new dimension to how systems engineers might interact with SysML v2 models, not through traditional code but via an intuitive natural language interface, potentially reshaping the future of systems modeling and analysis.

Error Handling and Iterative Solution Improvement

In this system, error handling becomes a pivotal component. When the LLM-generated code encounters an error during execution, the failure is not an endpoint. Instead, it represents a feedback loop (as shown in figure 2) that is essential for refining and perfecting the solution. The result is an iterative process with the following steps:

1. Attempt to execute the generated Python code within the modeling environment.
2. If an error occurs, log the code, the error message, and the context in which it was run.
3. The primary assistant evaluates the error, possibly by consulting a secondary assistant specializing in debugging or error analysis.
4. A revised piece of code is generated and sent back to the user or executed directly, depending on the setup.

This process repeats, with the system learning from each error, until a working solution is converged upon, and the thread of interactions leads to a successful operation.

Case Studies and Applications

This section delves into practical implementations to demonstrate the capabilities of Assistants and LLMs in interacting with SysML v2 and other related technologies. By examining applied case studies, we can appreciate how Assistants and LLMs translate theoretical advantages into real-world benefits.

Case Study 1: Modifying a Beam Model

The first case study illustrates the use of an Assistant to modify a SysML v2 model defining the physical characteristics of a beam. The objective is to change the length attribute of 'beam A' within the model stored in a local directory. The example model is shown to the left in figure 3.

```
package beamModels {

    import ScalarValues::Real;

    part beamA {
        attribute base: Real = 1;
        attribute height: Real = 1;
        attribute length: Real = 10;
        attribute load: Real = 1;
    }
}
```

Figure 3. Example SysMLv2 model where the objective is to change the length attrbute value.

Here, the assistant is provided with the necessary details to understand and interact with the model, including its file path and specific instructions. Once the OpenAI client is set up and the reference model file is uploaded, natural language requests are made. The LLM interprets these requests and generates corresponding Python code snippets to modify the model.

An example prompt is shown in figure 4 below. This prompt is sent to the Assistant as a part of the thread.

```
content = f"Provide python code to modify my file {file_name} in my local directry {file_path} to "
content = content + f"change the beam A length to {value} units "
content = content + f"use the file name {assistant.file_ids[0]} as the file to modify,"
content = content + f", you must return pyton to the user, always update the {file_name}."
```

Figure 4. Example prompt integrating python variables to define the file name and path, the model value to update, and the reference model to use for knowledge.

The proof-of-concept code snippet returned by the Assistant uses Python's standard library, such as the re (regex) module (see figure 5), to update the required attribute within the SysML v2 model's text file.

```
import re

# Define the path to the local sysmlv2 model file
beam_model_path = '/workspaces/ai_assistant/sysmlv2/models/beamModels.sysmlv2'

# Read the content of the existing sysmlv2 model file
with open(beam_model_path, 'r') as file:
    model_content = file.read()

# Define a regular expression pattern to find and replace the length value for beamA
pattern = re.compile(r'(part beamA {[^}]*attribute length: Real = )[0-9.]+;')
new_length_value = '12.666'

# Replace the length value using the regular expression pattern
updated_model_content = re.sub(pattern, r'\g<1>' + new_length_value + ';', model_content)

# Write the updated content back to the sysmlv2 model file
with open(beam_model_path, 'w') as file:
    file.write(updated_model_content)
```

Figure 5. Example of code returned by the Assistant API. This code is extracted from the full response and executed locally.

The Assistant API can quickly provide this specialized code in response to the user's command without a cumbersome back-and-forth or complex API calls.

Case Study 2: API Endpoint and Testing Front End for System Model Queries

The second case study examines an API built upon a Flask web application. The API interfaces with OpenAI's Assistants API, allowing developers to create applications to directly interact with SysMLv2 based system models. The tool provides access to the model programmatically via an `/ask` endpoint and through a front-end UI for testing purposes.

Figure 6 below represents a typical programmatic interaction with the Assistant. Here we interact with the SysML v2 reference model from Case Study #1 (where the length of the beam was changed) using the flask applications `/ask` endpoint. With the changes made in the previous case study the updated SysML model file is pushed to the Assistant as a new reference. Now we can perform queries or request changes to the updated model. Here we can see that a quite complex question is asked of the Assistant to evaluate the updated model for the maximum von mises stress. No equations were provided in the prompt, the Assistant only received boundary conditions and a load, both provided in a conversational manor. The Assistant then performs the calculation and returns the correct maximum von mises stress of 759.96psi (Note: `beamA` updated attributes are shown in figure 7 below).

```
curl -X POST https://shiny-potato-69gj97xg57r4f5rvp-5000.app.github.dev/ask \
-H "Content-Type: application/json" \
-d '{"user_id": "0001", "assistant_id": "asst_ntrbx78DAb8LziNedL5LjHiH", "query": "If beamA is made
of steel and subject to a 10lb load at one end and is fixed at the other end calculate the maximum
von mises stress in beamA?"}'
{
  "response": "The maximum von Mises stress in beam A, which is made of steel and subject to a 10lb
  load at one end and fixed at the other, is calculated to be approximately 759.96 psi (pounds per square
  inch).\n"
}
```

Figure 6. Example curl command for a complex conversation with the Assistant Ask endpoint.

In figure 7 we see the flask applications front end. This front end allows users to test interactions with the Assistant and even train it if necessary. Users can review historical conversations and results as well as select various Assistants to work with by modifying the `assistant_id` prior to sending the prompt.

Although the front end is useful for testing purposes the primary purpose of the flask application is to computationally interact with the Assistant via the `/ask` endpoint.

This setup exemplifies how Assistants and LLMs can be utilized to facilitate model understanding — answering complex questions, interacting with engineering tools, or even directly narrating characteristics captured in the SysML v2 model.

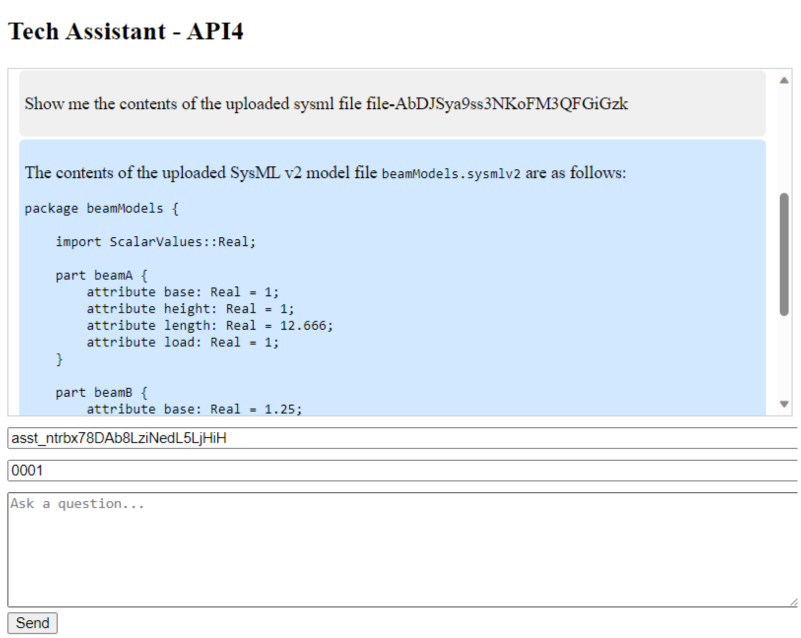

Figure 7. Example interactive front end to test the API '/ask' endpoint.

Case Study 3: Advanced SysML v2 Model Validation with AI Assistance

Background: In this case study, we explore the utilization of an AI assistant to validate and interpret a complex SysML v2 model. The model in question includes a specialized calculation for volume within a block definition, alongside two specific requirements that need validation. This scenario tests the AI assistant's capability to accurately interpret and evaluate SysML v2 models, especially when presented with unconventional calculations designed to challenge its understanding.

SysML v2 Model Overview: The SysML v2 model defines a block (`Block`) with dimensions (length, width, and height) and a volume attribute. A unique twist in the model is the volume calculation method, which intentionally includes an unconventional addition to the height parameter to test the AI assistant's interpretative skills. Additionally, the model specifies two requirements: a maximum volume requirement and a maximum length requirement, both of which are to be validated against the calculated dimensions of the block.

```
[ ]: // Definition of a block
    part def Block {
        attribute length: Length:Real;
        attribute width: Width:Real;
        attribute height: Height:Real;
        attribute volume: Volume:Real;

        // Calulcation definition is containted by the block definition
        calc def CalculatedBlockVolume {
            in l: Length;
            in w: Width;
            in h: Height;
            return : Volume = (l * w * (h+5)); // little trick here :)
            }
        }

    // Usage of a block
    part block_1: Block {
        i_length::> length = 2[m];
        i_width::> width = 2[m];
        i_height::> height = 2[m];
        c_volume =  0; // Returned calculated volume from analysis template
        m_volume::> volume = calculatedBlock_1Volume.volume; // Modeled Volume

        calc calculatedBlock_1Volume: CalculatedBlockVolume {
                in l = i_length;
                in w = i_width;
                in h = i_height;
                return volume;
            }

        }
```

Figure 8. Example SysMLv2 model with a complex calculation definition and usage.

Jupyter Notebook Implementation: The Jupyter Notebook (John K. DeHart, 2024) provided serves as the execution environment for this case study. It includes:

- Initialization of the OpenAI API and setup of necessary variables and directories.
- Reading of the SysML v2 model file and extraction of its content.
- Sending the SysMLv2 Model text to the AI Assistant as context input (training).
- Creation of an AI assistant with instructions to evaluate the model requirements and summarize the results in a specified table format based on the training data.

```
[ ]: CONTENT_TEXT = """Please evaluate the model requirments and summarize the requirments
evaluation results in a table with the colums ID, Subject, Description,
Actual Value, Calculated Value, Result (pass.fail)"""
```

Figure 9. Example AI assistant prompt to perform the evaluation. Notice that the prompt is quite vague and does not ask the assistant to directly evaluate the volume, only the requirements.

Evaluation Process and Results: The AI assistant was tasked with evaluating the model's requirements based on the provided SysML v2 model. The assistant successfully interpreted the model, including the specialized volume calculation, and proceeded to evaluate the defined requirements.

```
requirement def MaximumVolumnRequirement {
    doc /* The maximum volume allowable */

    attribute volumeCalculated: Volume;
    attribute volumeRequired: Volume;
    require constraint {volumeCalculated <= volumeRequired}
}

requirement def MaximumLengthRequirement {
    doc /* The maximum length allowable */

    attribute lengthCalculated: Length;
    attribute lengthRequired: Length;
    require constraint {lengthCalculated <= lengthRequired}
}

requirement <'1'> maxBlockVolume : MaximumVolumnRequirement {
    subject block_1: Block;
    attribute :>> volumeRequired = 5.5[m^3];
    require constraint requiredMaximumVolume { volumeCalculated <= volumeRequired }
    }

requirement <'2'> maxBlockLength : MaximumLengthRequirement {
    subject block_1: Block;
    attribute :>> lengthRequired = 2.08[m];
    require constraint requiredMaximumLength { block_1.length <= lengthRequired }
    }
```

Figure 10. Requirements definitions and usages.

The evaluation results were summarized in a table format, highlighting the ID, Subject, Description, Actual Value, Calculated Value, and Result (pass/fail) for each requirement.

```
Here is the summary table:

| ID  | Subject  | Description                 | Actual Value | Calculated Value | Result  |
|-----|----------|-----------------------------|--------------|------------------|---------|
| '1' | block_1  | Maximum Volume Requirement | 5.5[m^3]     | 28[m^3]          | Fail    |
| '2' | block_1  | Maximum Length Requirement | 2.08[m]      | 2[m]             | Pass    |

The model requirements evaluation reveals that while the system meets the maximum length requir
ement, it fails to meet the maximum volume requirement as calculated based on the model definit
ions.
```

Figure 11. AI Assistance requirements evaluation response. Notice that the AI Assistant correctly calculates the block volume, correctly evaluates the requirements, and returns the results in the requested tabular format.

Challenges and Observations:

- The specialized volume calculation was correctly interpreted by the AI assistant, demonstrating its ability to understand and process non-standard SysML v2 model definitions.
- The AI assistant accurately validated the requirements against the calculated values, identifying a failure in the maximum volume requirement.
- Note that the AI assistant can traverse the model depth of 'Definition (Block:Volume)' → 'Usage (block_1:volume)' and maintain and understanding of this relationship relative to the requirement evaluation.
- This case study showcased the potential of AI assistants in automating the validation of SysML v2 models, especially for complex calculations and requirements evaluation.

Conclusion: The successful interpretation and validation of the SysML v2 model by the AI assistant in this case study underscore the potential of integrating AI technologies with SysML v2 for enhancing systems engineering processes. This approach not only streamlines the validation of complex models but also introduces a layer of automation that can significantly reduce manual effort and increase accuracy in systems modeling and analysis.

Concluding Analysis

Across these case studies, we witness a consistent theme: LLMs can significantly enhance the interaction with system models, accelerate workflow, and reduce complexity. These implementations serve as proof of concept for the methodology discussed in earlier sections and underline the potential of integrating LLMs within systems engineering processes. This paves the way for future development, where direct manipulation of SysML v2 models and integration with various technological tools could become more streamlined, intuitive, and efficient.

Analysis and Results

The use of LLMs for direct interaction with SysML v2 models has been assessed against typical API calls based on the examples provided by the SysML-v2-API-Cookbook (SysML-v2-API-Cookbook. GitHub, n.d., 2023). Our findings underscore the advantages of LLMs in terms of improved efficiency, usability, and integration.

Baseline: Traditional API Interaction Complexity

Traditional API interactions entail:

- API Call Setup: Engineers need to understand HTTP methods, headers, and request/response formats.
- Data Schema Comprehension: Intimate knowledge of the data schema is required.
- Error Handling: Errors are manually parsed and resolved.
- API calls are typically extremely nested where complex recursions are necessary.

Note: *Due to space limitations within this paper an example API call is not shown here but the writer has several examples of very basic interactions obtaining 2 attribute values from a similar sysmlv2 model published to the SysMLv2 API. This API call, created using a Jupyter Notebook, required ≈150 lines of complex code. In contrast the entire code to create the Assistant and its necessary components to modify the local sysmlv2 file is ≈40 lines of very low complexity.*

LLM-Facilitated Interaction

Conversely, LLMs provide a streamlined experience:

- Efficient Command Interpretation: Natural language commands are readily interpreted into executable actions.
- Error Parsing and Code Generation: Errors trigger an iterative refinement process for code generation.
- Reduced Technical Barrier: Specialized API knowledge is less critical due to the conversational interaction format.

Comparative Analysis

Efficiency and Time Economy:

Conventional Methods	LLM Approach
Manual construction of requests	Natural language commands
Decoding API responses	Immediate response interpretation
Tedious error-handling process	Automated iterative refinement

Usability Improvement Metrics:

Metric	Conventional	LLM
Learning Curve	Steep	Shallow
User Engagement	Code-centric	Conversation-centric
Accessibility	Experts only	Inclusive for non-experts
Time to Completion	Lengthy	Reduced
Lines of Code	150 Lines	40 Lines
Coding time	≈ 2.0 Hours	≈ 0.5 Hours

Model Integration Capability:

API	LLM
Standalone scripts/API calls	Seamless tool and workflow integration
Manual model updates	Direct conversational model manipulation
Fragmented interaction	Unified, context-aware interaction

Outcome Evaluation

The test cases demonstrate:

- A significant reduction in the time and complexity associated with API interactions.
- LLM Assistants can minimize the manual code engineers must wrangle, directly enhancing productivity.
- Model integration into larger workflows becomes more straightforward with conversational interactions.
- The robust error handling employs LLMs' inherent iterative processing capacity, promoting minimal user intervention for path correction.

Adopting LLMs facilitates a distinct improvement in how engineers can interact with SysML v2 models. The analysis casts LLMs not just as tools, but as collaborative agents in systems engineering, marking a critical leap toward intuitive, efficient, and inclusive design methodologies.

Discussion

The utilization of LLMs for direct interaction with SysML v2 models heralds a significant paradigm shift in systems engineering. Our findings suggest that LLMs can improve the efficiency and accessibility of Model-Based Systems Engineering (MBSE), allowing for more agile interactions and potentially influencing the entire lifecycle of system development.

Implications for Systems Engineering

LLMs may enable a knowledge-driven approach to systems engineering. By understanding and generating SysML v2 constructs based on the natural language input, LLMs can help engineers leverage existing knowledge bases for analysis and design. This represents a move away from purely heuristic or data-driven methods that traditionally lead to worst-case design scenarios (Hamed Haddad Khodaparast, 2012), especially in the context of flight-critical hardware.

In such high-stakes environments, it's common for design decisions to be driven by the need to accommodate for worst-case scenarios as opposed to making probabilistically informed decisions.

By contrast, a knowledge-based design facilitated by LLMs could help designs remain firmly rooted in established expertise, grounded in a comprehensive understanding of system behavior. This shift has the potential to curb the tendency towards over-conservatism while still maintaining safety through an informed and knowledge-rich approach.

Limitations and Considerations

However, several limitations and considerations are recognized:

- **Model Reliability:** While LLMs are adept at language processing, their outputs need to be validated for technical reliability and accuracy, especially for flight-critical hardware where failures can have severe consequences.
- **Knowledge Quality:** The quality of the LLM's output is heavily dependent on the quality of the information provided. Ensuring the knowledge bases are current, comprehensive, and technically sound is vital.
- **Design Evolution:** Systems engineers must be cautious about becoming too reliant on knowledge-based designs that may not adapt quickly to novel situations or incorporate the latest empirical data as it emerges.

- **Integration into Current Workflows:** The adoption of LLMs requires adaptation of existing workflows and possibly retraining personnel, which could be met with resistance or require significant resource investment. (not unlike the transition to MBSE…)

Prospects for Knowledge-Driven Design

The embracing of LLMs for a knowledge-driven approach to systems design aligns with the growing trend of MBSE. It offers an alternative to the "safety by default" worst-case design philosophy, which could facilitate more efficient use of resources without compromising safety or performance. By basing design decisions on a deep understanding of system requirements, behavior models, and performance under various scenarios, systems engineering can become more grounded in realistic conditions, possibly leading to innovative, optimized solutions.

The exploration of LLMs in conjunction with SysML v2 within systems engineering introduces a transformative methodology that integrates human expertise and statistical data in the design process. This method holds the potential for a more formed approach to creating flight-critical hardware, shifting from heavy reliance on worst-case models towards a knowledge-based framework supported by statistics. However, the advent of LLMs in this domain also necessitates a mindful approach to avoid the pitfalls of automation bias (Parasuraman, 1997) and the uncanny valley effect (Mori, MacDorman, & Kageki, 2012-06). Preventing knee-jerk shifts to extreme conservatism contributing to systems that are excessively massive and costly, often exceeding safety requirements under the guise of caution once a failure, even when within the stated probability occurs, is crucial to the successful implementation.

Conclusion and Future Work

This paper presented an innovative approach to interfacing with SysML v2 models by harnessing the capabilities of LLMs. We proposed leveraging LLMs to interpret, manipulate, and generate SysML v2 constructs through natural language interaction, reducing the need for complex API calls. This approach has the potential to facilitate a more intuitive and efficient modeling process for systems engineers and stakeholders alike.

Throughout this investigation, we've highlighted some potential benefits of incorporating human-like language processing in the realm of systems modeling. By doing so, we anticipate the simplification of model interactions, improved accessibility of MBSE for non-specialists, and enhanced integration of models with other tools and workflows. Our case studies and applied methodologies underscored the practicality and versatility of LLMs in a system engineering context.

However, the concurrence of SysML v2's human-readable syntax and the advent of sophisticated LLMs does not guarantee a seamless merger. While SysML v2 is designed with readability in mind, creating a framework where LLMs can effectively 'understand' and interact with these models poses challenges.

Yet, the timing of the emergence of both SysML v2 and advanced LLMs presents a unique opportunity. Their synergy could catalyze the evolution of systems engineering, transitioning from static designs to dynamic models that grow and adapt through the iterative insights generated by LLM interactions. The combination of human-centric design language and AI's analytical prowess may open new frontiers in the optimization and management of complex systems.

Looking ahead, the continuous refinement of LLMs presents an exciting frontier for MBSE. As our understanding of their capabilities deepens, and as LLMs become more sophisticated, future research can explore extending this approach to more complex integrations, including the development of Domain Expert Systems (DES) that leverage the synergy between SysML v2 and LLMs.

Development and Integration of Expert Systems with LLMs and SysML v2:

For years, attempts to build knowledge-based expert systems using tools such as CLIPS (Riley, 1991) have faced challenges, including poorly performing models and the difficulty of defining and curating domain-level knowledge. However, the emergence of LLMs, coupled with the potential integration of SysML v2, may finally offer a viable solution to these longstanding issues. The concept of Domain Expert Systems (DES) (Giarratano, 2005) is particularly promising in the context of Retrieval Augmentation Generation (RAG) tools and methods (Penghao Zhao, 2024). These advancements enable specialized domains to compile and manage their own collections of text-based knowledge, allowing for the training of LLMs tailored to their specific requirements. Additionally, they provide unprecedented clarity in understanding domain-specific knowledge across different fields. This approach could revolutionize the typically siloed business of engineering by providing a more cohesive and intelligent framework for knowledge integration and application.

Future Directions in Research and Application:

- Domain-Specific LLM Training: Investigating methodologies for efficiently training LLMs with domain-specific knowledge sets, enabling the creation of highly specialized expert systems that can support complex engineering tasks.
- SysML v2 and LLM Integration: Developing frameworks and tools that seamlessly integrate SysML v2 models with LLM-powered expert systems, enhancing the ability to interpret, validate, and generate complex systems models.
- Ethical and Practical Considerations: As these technologies evolve, it is imperative to address the ethical implications of AI in engineering, ensuring that the integration of LLMs and expert systems into MBSE practices enhances human expertise without replacing it. This includes establishing guidelines for transparency, accountability, and decision-making support in AI-assisted engineering processes.

Impact on the Systems Engineering Lifecycle:

The convergence of SysML v2, with its conciseness and readability, with the cognitive processing power of LLMs, stands to significantly impact the systems engineering lifecycle. This impact includes the potential for more efficient design processes, improved accuracy in systems modeling, and enhanced collaborative capabilities across engineering teams. The work ahead involves rigorous experimentation, validation, and a disciplined approach to ensuring these technologies augment rather than overshadow human expertise. By embracing these advancements, the field of systems engineering can look forward to a future where expert systems and AI play a pivotal role in shaping the development of complex systems.

References

Estefan, J. (2008). Survey of Candidate Model-Based Systems Engineering (MBSE) Methodologies. Seattle, WA, USA: International Council on Systems Engineering (INCOSE).

Giarratano, J. C. (2005). *Expert systems : principles and programming.* Australia ; Boston, Mass.: Thomson Course Technology.

Hamed Haddad Khodaparast, G. G. (2012). EFFICIENT WORST CASE ”1-COSINE” GUST LOADS PREDICTION. *Journal of Aeroelasticity and Structural Dynamics, Vol 2, No 3*, 33-54.

John K. DeHart. (2024). *volume_calc/ai_volume_analysis.ipynb.* Retrieved from Avian Inc. GitHub repository: https://github.com/avianinc/incose_2024/blob/main/notebooks/SysML_Testing/volume_calc/ai_volume_analysis.ipynb

Micouin, P. (2019). MBSE, What is Wrong with SysML -First Issue. ffhal-02070455v2. *HAL Open Science*.

Mori, M., MacDorman, K. F., & Kageki, N. (2012-06). The Uncanny Valley [From the Field]. *IEEE robotics & automation magazine, Vol.19 (2)*, 98-100.

Nikhil Mehta, M. T. (2023, 4 21). *Improving Grounded Language Understanding in a Collaborative Environment by Interacting with Agents Through Help Feedback. arXiv. arXiv:2304.10750v2 [cs.CL].* Retrieved from https://doi.org/10.48550/arXiv.2304.10750

Object Management Group. (2024, April). *ABOUT THE OMG SYSTEM MODELING LANGUAGE SPECIFICATION VERSION 2.0 BETA 2.* Retrieved from OMG | Object Management Group: https://www.omg.org/spec/SysML

Object Management Group., (n.d.). (2023, 9 6). *SysML v2 Overview.* Retrieved from https://www.omg.org/pdf/SysML-v2-Overview.pdf

OpenAI, (n.d.). (2023, 08 04). *Assistants overview.* Retrieved from Assistance Overview - OpenAI API: https://platform.openai.com/docs/assistants/overview?context=with-streaming

Parasuraman, R. &. (1997). Humans and Automation: Use, Misuse, Disuse, Abuse. . *Human Factors, 39(2)*, pp. 230-253.

Penghao Zhao, H. Z. (2024). Retrieval-Augmented Generation for AI-Generated Content: A Survey. *arXiv:2402.19473 [cs.CV]*.

Riley, G. (1991). *CLIPS: An expert system building tool.*

Rouse, W. (2020). AI as Systems Engineering Augmented Intelligence for Systems Engineers. *INSIGHT 23*, 52-54. Retrieved from https://doi.org/10.1002/inst.12286

SysML-v2-API-Cookbook. GitHub, n.d. (2023, 9 6). Retrieved from Systems Modeling: https://github.com/Systems-Modeling/SysML-v2-API-Cookbook. Accessed

Tom B. Brown, B. M.-V. (2020). Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs.CL]*.

Biography

John K. DeHart is a Sr. Principal Systems Engineer and Digital Engineer at AVIAN, holding an Masters in Applied Systems Engineering from Georgia Institute of Technology. His expertise in digital workflows, SysML v2, and LLM integration drives pioneering digital engineering solutions in defense, emphasizing model-based engineering and AI applications.