

# Facilitating User-Centric Model-Based Systems Engineering using Generative AI

Elias Bader<sup>1</sup><sup>a</sup> Dominik Vereno<sup>1</sup><sup>b</sup> Christian Neureiter<sup>1</sup><sup>c</sup>

<sup>1</sup>*Josef Ressel Centre for Dependable System-of-Systems Engineering,  
Salzburg University of Applied Sciences,  
Urstein Süd 1, 5412 Puch/Salzburg, Austria  
{first name}.{last name}@fh-salzburg.ac.at*

Keywords: Large Language Model, GPT, Cyber-Physical systems, Artificial Intelligence, UML

Abstract: The increasing complexity of cyber-physical systems requires model-based systems engineering (MBSE) in an effort to sustain a comprehensive oversight. However, broader adaptation of these models requires specialized knowledge and training. In order to make this process more user-friendly, the concept of user-centric systems engineering emerged. Artificial intelligence (AI) could help users overcome beginner hurdles and leverage their contribution quality. This research investigates the feasibility of a large language model in the systems engineering context, with a particular emphasis on the identification of potential obstacles for similar tasks. Therefore, a GPT model is trained on a dataset consisting of UML component diagram elements. In conclusion, the promising results of this research justify utilizing AI in MBSE. Complex relationships between the UML elements were not only understood, they were also generated using natural-language text. Problems arise from the extensive nature of the XMI, the context limitation and the unique identifiers of the UML elements. The fine-tuning process enabled the LLM to gain valuable insights into UML modeling while transferring their base knowledge, which is a promising step toward reducing complexity in MBSE.

## 1 INTRODUCTION


Cyber-physical systems are becoming increasingly vital in various sectors. They merge physical processes with digital computation, resulting in their complexity (Lee, 2008). At the same time, the rapid evolution of artificial intelligence (AI) presents both challenges and opportunities. On one hand, data-driven decision-making makes systems even more complex. On the other hand, the emergence of generative AI, such as large language models (LLMs) like ChatGPT, shows promise in managing difficult engineering tasks.


Model-based systems engineering (MBSE) has established itself as a key tool in dealing with the complexity of developing cyber-physical systems (Neureiter et al., 2020). It focuses on a central digital model through various engineering stages. However, its broader adoption is limited by the need for specialized knowledge in object-oriented development, (semi-)formal modeling languages (e.g. UML or


SysML), and complex tools. The emerging concept of user-centric systems engineering (UCSE) aims to make MBSE more user-friendly. In making MBSE more accessible, generative AI can be tremendously helpful: LLMs can support users in constructing formally correct system models using natural-language prompts. By utilizing the code representation of system models, we can benefit from the current progress of LLMs in the area of code generation. Their increasing proficiency enables LLMs to generate, adapt, complete or refine code and a clear parallel can be seen in which AI-assisted programming is evolving in a more user-centric direction (Wong et al., 2023). These powerful language models can be fine-tuned to specific application scenarios, such as MBSE.

Our study aims at AI-enabled, user-friendly MBSE. We pursue two main objectives:

1. Establishing a proof of concept for generating UML models from natural-language inputs, utilizing a specialized and fine-tuned LLM.
2. Identifying and addressing the challenges involved in LLM-assisted model generation to develop a practical and efficient engineering methodology.

<sup>a</sup> <https://orcid.org/0009-0003-9025-8100>

<sup>b</sup> <https://orcid.org/0000-0002-7930-6744>

<sup>c</sup> <https://orcid.org/0000-0001-7509-7597>

Our approach involves fine-tuning an LLM, specifically GPT, using a dataset of natural-language description and corresponding XMI-encoded UML models. This research aims to showcase AI’s potential in making MBSE an accessible tool for dealing with the complexity of modern CPS.

## 2 BACKGROUND

This chapter provides an overview of the underlying concepts that are relevant to this research. First, the basics of the Unified Modeling Language (UML) and the associated XML Metadata Interchange (XMI) format are described. The focus is then placed on the general concepts of LLMs, with a particular emphasis on OpenAI’s GPT models. It also examines how context and fine-tuning processes influence the behavior and output of these advanced language models.

### 2.1 UML and XMI

The UML is a broadly used general-purpose language for modeling, specification, visualization and documentation of complex systems. UML is characterized by its use of graphical elements to represent system structures and behaviors. With the introduction of UML 2.0 in 2005, the language was further improved by the addition of the textual notation XMI (XML Metadata Interchange Format). XMI is a standard format for interchanging UML Models based on the Extensible Markup Language (Rupp et al., 2012). A structured representation of UML models is essential for interaction with LLMs, thus the use of the XMI representation is required.

#### 2.1.1 UML Component Diagrams

UML component diagrams fall under the category of structure diagrams, serving to represent various parts of a system as components in runtime. These diagrams are essential for visualizing the organization and interactions of different components within a system, particularly highlighting how components communicate through clearly defined interfaces (Rupp et al., 2012). Basic understanding of the following four components is important to follow this research:

- **Component:** Primary building block, representing a modular part of a system.
- **Port:** Points of interaction for a component.
- **Exposed Interface:** Reference to the interfaces that a component either provides or requires.

- **Interface:** Specification of the set of operations or services, that are either provided or required by a component.

Figure 1 illustrates a typical interaction of these four elements and serves as a reference implementation within this research. A component may have multiple ports with exposed interfaces. A required interface uses the provided interfaces, and both are realized by an interface with its definition.

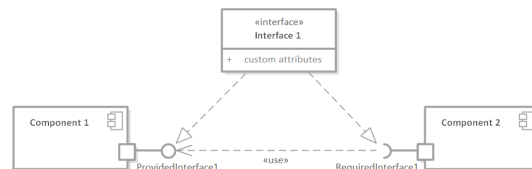


Figure 1: Exemplary interaction between components in UML component diagrams.

### 2.2 Large Language Models

Large Language Models are a combination of multiple AI disciplines that mainly inherit from deep learning and natural language processing (NLP) and are so-called generative AIs (Amaratunga, 2023). The fundamental basis of this method is grounded in stochastics and probabilities, however, there are some underlying concepts that require further clarification.

#### 2.2.1 Tokenization

The first challenge is to make a natural-language text machine-readable. This parsing process involves several different methods, for this research, only tokenization is relevant. Tokenization in NLP is a fundamental step in which a text is broken down into smaller units, known as tokens. These tokens are often words or sub-words and serve as base elements for various language processing tasks. This not only reduces the complexity of a sentence but also represents it in a structured format (Amaratunga, 2023).

#### 2.2.2 Transformer-based Models

Some of the most sophisticated LLMs nowadays are generative pre-trained transformers (GPTs). GPT models consist of multiple complex layers and are able to create an output from an input. The input is used as context for the generation of the output, but so is the output. This works because GPTs are so-called autoregressive models, they generate word after word. This makes it possible to use the previously generated text as the additional context for the next words (Amaratunga, 2023).

An important contribution to the success of transformer-based models was developed by Ashish Vaswani et al. with the paper "Attention Is All You Need" in 2017. This research presented a significant milestone in the field of natural language processing by introducing the attention mechanism into the transformer model, a new and promising approach that departs from traditional recurrent or convolutional neural networks. The attention mechanism allows the encoders and decoders to focus on different parts of the input sequence by assigning them more weight. Based on the task of translating a text from English to German, this transformer-based model was superior in quality and also required a significantly smaller amount of training time by being more parallelizable (Vaswani et al., 2017). Consequently, these advancements enabled the GPTs to train on huge datasets to create LLMs with more general knowledge and understanding of the concepts in the real world.

### 2.2.3 Training GPT Models

GPT models belong to the family of LLMs and have an incredible power of generating natural text due to their training performance that enabled them to gain knowledge through huge datasets. This ability can be further utilized to effectively train pre-trained models on a specific topic and transfer the already-known concepts and patterns to new domains (Amaratunga, 2023). While the process of fine-tuning LLMs requires an understanding of algorithms like back-propagation and gradient descent, vendors like OpenAI effectively simplify this complexity with an additional abstraction layer, making the technology more accessible. By tailoring such models through fine-tuning, we achieve a higher level of precision and reliability, by greatly improving their applicability and effectiveness on specific tasks.

## 2.3 OpenAI's GPT API

Since its introduction by OpenAI in 2018, the GPT series of models has evolved, with each iteration labeled as *GPT-n*, where *n* indicates the version number. The GPT-3.5 model, released in 2022, marked a significant advancement with its extensive API supporting fine-tuning, a key feature in this research. In fine-tuning, the model receives prompts specifying desired output characteristics, and each interaction contributes to a conversational context, influencing subsequent responses. A critical aspect of GPT models is their context size limitation, with the latest GPT-3.5-turbo-1106 model handling up to 16K tokens.

The interaction architecture between a human and the LLM is structured so that the model receives a

prompt detailing specific requirements that its output must meet. Each prompt and subsequent response generated by the LLM contributes to a conversational context that is continuously utilized to produce the next segment of the response. A notable distinction among GPT versions is their maximum comprehensible context size, managed internally by a context window that automatically drops the oldest context information if the tokens exceed the limit (OpenAI, 2023a). Currently, GPT-3.5-turbo-1106 is the most recent available model for fine-tuning jobs, and is capable of handling a context with the size of 16K tokens (OpenAI, 2023b).

GPT models are fine-tuned with a collection of JavaScript Object Notation (JSON) Objects in the JSON Lines format. Each line of this format maps a system description and a full conversation between the two personas "user" and "assistant". The dataset represents example conversations between a user and with the anticipated response of the LLM (OpenAI, 2023a). During the fine-tuning process, this data set is used to adjust the parameters of the model so that the expected responses are reproduced as closely to the training set as possible.

## 3 APPROACH

As outlined in Section 1 a user-centered approach to MBSE could be supported by the utilization of artificial intelligence. To achieve the results for this research, this section deals extensively with the approach of creating the training dataset, generating XMI with the trained GPT model and the approach for evaluating the results.

### 3.1 Dataset Generation

To conduct the research, UML component diagram elements are utilized. These are structured and straightforward elements describing significant components and their interconnections within a system. Representative for all UML elements, their XMI representation is used to generate the fine-tuning dataset for the GPT model. The dataset consists of 10 different conversations between a user and an assistant, where the user is requesting XMI code from the LLM. The requested code examples are UML component diagram elements with increasing complexity, organized into three complexity classes. These classes are designed to progressively demonstrate more intricate component interactions and system architectures based on the reference implementation in Section 2.1.1:

- 1. Standalone components and interfaces:** The simplest class involves standalone components, which may have a port with exposed interfaces and their realizing interfaces. These examples focus on individual elements in isolation, providing a fundamental understanding of components and their XMI structure.
- 2. Connected components with interfaces:** The second complexity class introduces connected components, where one component acts as a provider and/or as receiver of exposed interfaces. This class focuses on which interactions between components are permitted and desired, and therefore already uses all four elements.
- 3. Nested components with internal and external connections:** The most complex class features components within components that have internal connections and expose ports through multiple layers. The complex interaction and the numerous dependencies within the XMI are intended to challenge the training.

Each example is modeled in the systems modeling tool Enterprise Architect (Systems, 2023) and exported into the machine-readable XMI format. Given the extensive amount of information this format contains, it is possible to reduce some data segments in a pre-processing step without altering the UML elements. All data in XMI is subordinate to the XML tags. This makes it possible to delete tags like project information, styles, diagrams and diagram positions of the elements directly. Attributes such as the author, creation- and modification times can also be deleted without hesitation. This completes the XMI code for integration into a dataset

The JSON Lines format, which is the dataset format used for the fine-tuning process of GPT models, expects a conversation between a user and an assistant in each line as a JSON object. In the dataset used for this research, the user is requesting component diagram elements and the assistant responds with XMI code. Each request consists of multiple English sentences describing the desired components and their interactions. The XMI examples and the user prompts are then combined accordingly to create a dataset with 10 conversations. Figure 2 visually represents the data set generation steps applied in this study.



Figure 2: Fine-tuning dataset generation visualized steps.

## 3.2 XMI Generation and Evaluation

Once the GPT model is trained successfully with the specially prepared dataset, the model becomes accessible for practical use through the OpenAI API. To assess the effectiveness of the trained GPT model, we generate XMI code for the four types of UML elements covered in the training. This generated code is then imported into the Enterprise Architect software for further evaluation. The prompts used for generating XMI code adhere to specific criteria to ensure a robust evaluation:

- No prompts from the training dataset are reused, to avoid biasing the model's responses.
- Unique UML models are generated with different interaction scenarios for each complexity levels.

The XMI code generated by the GPT model often necessitates post-processing. This involves manual adjustments to refine and correct the output, ensuring the correctness of the XMI code. Figure 3 illustrates the steps to generate XMI code from GPTs API and import it into Enterprise Architect to visualize the result for an evaluation.



Figure 3: XMI code generation visualized steps

The evaluation is exclusively performed through a human assessment of the generated XMI code and focuses on two primary criteria: its syntactical correctness and the completeness of elements and relationships as specified in the request prompt. The analysis will concentrate then on identifying learned patterns and obstacles encountered during the generation process.

## 4 REALIZATION

Based on the determined research approach, this section describes the realization of the trained GPT model and the generation of the XMI code. The relevant steps are explained and the results are presented.

### 4.1 Complexity Classes

The basis for creating the necessary dataset for the training process is clear and structured data preparation. Therefore, 10 examples of increasing complexity are modeled, and respective user prompts are formulated according to Section 3.1. Figures 4,5 and 6 represent one complexity class each.



Figure 4: Representative model elements for complexity class 1.

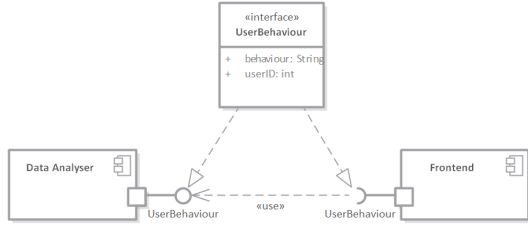


Figure 5: Representative model elements for complexity class 2.

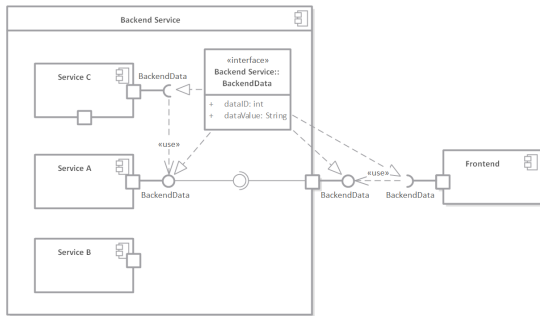


Figure 6: Representative model elements for complexity class 3.

## 4.2 XMI Pre-Processing

The pre-processing of the XMI includes the removal of irrelevant XML tags that do not alter the UML elements. This not only ensures that the GPT model only learns the essential relationships, but it also saves a non-negligible amount of tokens. Using the Python library "etree", it is possible to modify an XML file and deleting all tags and attributes described in Section 3.1.

With a slightly modified version of the python code provided by the OpenAI documentation, it is possible to count the tokens of a file for a specific GPT model version via the tiktoken python library (OpenAI, 2023a). Table 1 presents the token and the rough file sizes before (Raw) and after (Edited) the pre-processing step.

As can be seen, the pre-processing achieves a token

Table 1: Complexity level 2 token and file size comparison before and after pre-processing.

File State	File Size	Tokens
Raw	25KB	10.526
Edited	20KB	7.216

saving of about 30%. Nevertheless, even the reduced version of this example already occupies over 40% of the maximum context size of 16K Tokens.

## 4.3 Data Merging and Fine-Tuning

To create the data set for the fine-tuning process, the cleaned XMI needs to be merged with a natural text. The following 3 descriptions are the user prompts for the previously selected examples for the complexity classes 1–3, it should be noted that the example from complexity level 3 was created within 2 user prompts that build up on each other.

- **1 - Two Components** "Create Component X and Component Y"
- **2 - Two Connected Components** "Create a Component for my 'Data Analyser' service that exposes the interface 'UserBehaviour'. The 'UserBehaviour' interface has the attributes 'userID' as int and 'behaviour' as String, and is received by the 'Frontend' component."
- **3 - Two Prompts, Nested Components**
  1. "Create three components within the parent component 'Backend Service', named 'Service A', 'Service B', and 'Service C'."
  2. "Add an interface named 'BackendData' to 'Service A' within 'Backend Service'. This interface should be exposed to an external component named 'Frontend Service'. Include attributes 'dataID' as int and 'dataValue' as String in the 'BackendData' interface. 'Service C' is also requesting the 'Backend Service' from 'Service A'."

A similar description was created for all other examples and inserted manually into a JSON Lines file. The GPT model was then fine-tuned via OpenAI's online platform.

## 4.4 XMI Generation

Since the fine-tuned model is now easily accessible through an API call like in the OpenAI documentation, the model can be used to generate XMI code for a validation process. In this part, two exemplary XMI models are generated, those are intended to be representative of the results.

The first requesting user prompt: "Create a component named 'Weather Tracker' with a port element.

Additionally, create an interface named 'WeatherDataInterface', which includes the attributes 'temperature', 'humidity', and 'windSpeed'" addresses the complexity class 1.

In the training set is no case for individual components where a port exists without an exposed interface, which is realized by an interface. This test is intended to check that no false assumptions or relationships have arisen from the training. In Figure 7 the generated XMI code is imported and visualized in Enterprise Architect, no post-processing was needed and all requested elements and attributes are generated.



Figure 7: Generated XMI for complexity class 1.

The generation for complexity class 2 focuses on a mutual dependency between two components, since there were no similar examples defined in the training set. The following requesting user prompt generates the code for Figure 8: "Generate me two distinct components: 'Audio Player' and 'Media Library'. The 'Audio Player' should provide an interface called 'PlaybackControl' with functions such as 'Play', 'Pause' and 'Stop'. This interface is used by the 'Media Library'. Conversely, 'Media Library' provides an interface called 'LibraryAccess' with functions such as 'fetchTracks' and 'updateLibrary', which is used by the 'Audio Player'."

The generation process was interrupted because the context limit was reached. As a result, the unfinished code had to be corrected by closing all XML tags before importing into Enterprise Architect. Therefore, there was no change in the generated elements, however, the code is not complete and without human post-processing unusable. From an objective point of view, 2 ports and 2 exposed interfaces are missing, and both realizations of the interfaces are wrongly linked. However, the names of the generated elements and their features have been generated correctly.

As the context has already been exceeded in complexity class 2, it was not possible to generate more complex examples that add any value to this research.

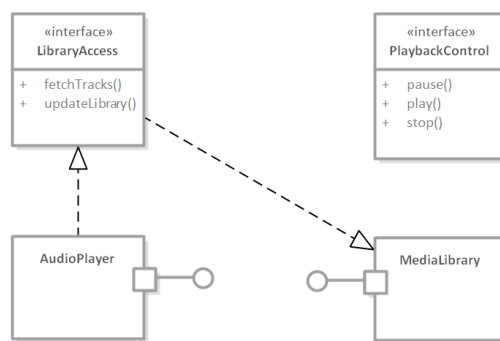


Figure 8: Generated XMI for complexity class 2.

## 5 FINDINGS

This study demonstrates that even with a limited dataset of 10 examples, the GPT model is capable of accurately creating UML elements and establishing correct links in XML code. Notably, the model shows proficiency in understanding the syntactical nuances of the XMI language. It reliably generates the names and features of components and interfaces, and correctly identifies that an exposed interface is always attached to a port. However, it consistently struggles with realization connections, indicating an area for further refinement.

The experiment revealed three significant obstacles:

1. **Extensive data representation:** The generality and extension of the XML code leads to a textual overhead for the representation of simple information. This requires the LLM to understand complex relationships between XML tags that are linked together throughout the entire context. We therefore believe that a much larger data set is required until the patterns are actually recognized. This complexity raises questions about the efficiency of using the XMI language for such tasks.
2. **Context limit:** The verbose nature of XML exacerbates the issue of context limitations in the GPT-3.5-turbo model. In scenarios of moderate complexity, the generation quickly approaches its context limit, leading to unreliable outputs or aborted generations. This limit is roughly exceeded by generating about 8 elements with small features. Furthermore, due to the moving context window, it is not possible to reference past elements in the course of a conversation. An adaptation of generated models within a conversation is with this approach not possible.



- Challenges with IDs:** A critical observation pertains to the handling of unique identifiers (IDs). Each XMI element maintains an ID, consisting of 37 characters and presents a unique challenge in the context of LLM processing. The concern is that while the LLM recognizes the pattern of an ID, we think it does not effectively verify its uniqueness and generate it using the appropriate algorithm. Furthermore, false associations during the training process could arise. For instance, the model might incorrectly learn that a port element always possesses a certain ID pattern, or inversely, that a particular ID pattern dictates an element's behavior. Additionally, the training process can inadvertently identify non-existent errors and thereby impair the model's learning process. The LLM's approach of constantly generating responses and comparing them against our training dataset for weight adjustment becomes inefficient due to the inherently random nature of IDs.

## 6 CONCLUSION AND OUTLOOK

The application of AI is becoming increasingly relevant in the design of user-centric MBSE. Not only could it reduce the inherent workload, it could also help to overcome beginner hurdles and assist in the standardized use of modeling languages.

Within this research, a dataset of 10 XMI example models were created and used to fine-tune a GPT-3.5 model. This trained model was then utilized to generate UML component elements using a description in a natural language text. While the model shows promise in recognizing patterns and relationships in XML code, significant challenges arise with extensive data representation, context limits, and the handling of unique IDs. Nevertheless, the results justify using AI in MBSE.

Looking forward, the release of GPT-4 for fine-tuning with an expanded context limit of 128K tokens and additional training data may still find XMI's representation of UML elements challenging. To address both the issues of data bloat and IDs, transitioning to alternative modeling languages, such as SysML v2, could be advantageous. SysML v2 not only offers a more efficient textual representation related to programming languages but also incorporates an extensive and expandable graphical notation.

This shift to a simpler, more intuitive language structure could significantly enhance AI's ability to model complex systems, opening new approaches for research and application in user-centric AI-driven systems engineering.

## ACKNOWLEDGEMENTS

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development and the Christian Doppler Research Association as well as the Federal State of Salzburg is gratefully acknowledged.

## REFERENCES

- Amaratunga, T. (2023). *Understanding Large Language Models*. Apress.
- Lee, E. A. (2008). Cyber physical systems: Design challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369.
- Neureiter, C., Binder, C., and Lastro, G. (2020). Review on domain specific systems engineering. In *2020 IEEE International Symposium on Systems Engineering (ISSE)*, pages 1–8. IEEE.
- OpenAI (2023a). Openai api documentation.
- OpenAI (2023b). Openai pricing.
- Rupp, C., Queins, S., and die SOPHISTen (2012). *UML 2 glasklar*. Carl Hanser Verlag GmbH Co KG.
- Systems, S. (2023). Sparxsystems enterprise architect.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.
- Wong, M.-F., Guo, S., Hang, C.-N., Ho, S.-W., and Tan, C.-W. (2023). Natural language generation and understanding of big code for ai-assisted programming: A review. *Entropy*, 25(6).