On the use of LLMs to support the development of domain-specific modeling languages

Claudio Di Sipio
University of L'Aquila, Italy
claudio.disipio@univaq.it

Riccardo Rubei
University of L'Aquila, Italy
riccardo.rubei@univaq.it

Juri Di Rocco
University of L'Aquila, Italy
juri.dirocco@univaq.it

Davide Di Ruscio
University of L'Aquila, Italy
davide.diruscio@univaq.it

Ludovico Iovino
GSSI, L'Aquila, Italy
ludovico.iovino@gssi.it

Abstract

In Model-Driven Engineering (MDE), domain-specific modeling languages (DSMLs) play a key role to model systems within specific application domains. Creating DSMLs is a complex, iterative process requiring input from both domain specialists and technical experts. This process often involves developing language artifacts, including syntax, semantics, and supporting environments, to gather feedback and achieve consensus among stakeholders.

To facilitate the interaction between technical experts and domain specialists in the creation of new DSMLs, we propose using large language models for the specific task of supporting the requirement elicitation of language semantics. Our approach aims to reduce the time needed to develop proof-of-concept implementations, facilitating quicker agreement on the language's intended functions. Once consensus is reached, traditional technologies can be employed to develop the semantics of the agreed language. This method aims to mitigate potential misunderstandings that can arise during interactions between technical specialists and domain experts. As an initial investigation of this idea, we explore the model mutation problem as a case study. We developed a custom GPT model, named MuTaGENe, designed to support the definition of model mutations and tested it against the existing Wodel language.

CCS Concepts

• **Software and its engineering**; • **Computing methodologies** → *Artificial intelligence*;

Keywords

Large Language Models, Model Driven Engineering, Domain-specific languages

ACM Reference Format:
Claudio Di Sipio, Riccardo Rubei, Juri Di Rocco, Davide Di Ruscio, and Ludovico Iovino. 2024. On the use of LLMs to support the development of domain-specific modeling languages. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3652620.3687808

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3687808

1 Introduction

In Model-Driven Engineering (MDE) [5] software models are considered as first-class entities throughout the entire life-cycle of complex systems. The specification of models is supported by domain-specific modeling languages (DSML) that allow modelers to represent real-world concepts in a given application domain [13]. Modeling languages are built on top of three main components [6], i.e., abstract syntax, concrete syntax, and semantics. The abstract syntax is typically defined as metamodels encoding concepts, relationships, and rules related to the application domain of interest. Metamodels underpin the development of textual or graphical concrete syntaxes that target users exploit to define models. Semantics refers to the meaning and behavior of the language elements.

Developing DSMLs can be a complex activity, encompassing an iterative process requiring input from domain specialists. In contrast, technical experts develop the syntaxes and the semantics of the language of interest and refine them to accommodate gathered feedback until a consensus among the stakeholders is achieved. Even though advanced techniques and tools are nowadays available for developing language syntaxes and corresponding facilities (like graphical or textual editors endowed with syntax highlighting, completion, etc.), developing language semantics still requires advanced expertise.

Over the last decades, MDE and Software Engineering in general have been revolutionized by the increasing adoption of Machine Learning (ML) and Deep Learning (DL) approaches to support different SE activities including intelligent modeling assistants [19] that have been adopted to automatize several MDE tasks [11, 20, 23]. To facilitate the interaction between technical experts and domain specialists in creating new DSMLs, in this paper, we propose using Large Language Models (LLMs) to support the requirement elicitation of language semantics. Our approach aims to reduce the effort needed to develop proof-of-concept language implementations, facilitating quicker agreement on the language's intended functions. Once consensus is reached, technical experts can employ traditional technologies to develop the semantics of the agreed language. This method mitigates potential misunderstandings between technical and domain experts during interactions.

As an initial investigation of this idea, we explore the model mutation problem as a case study. We developed a custom GPT-4 model, named MuTaGENe, to support the definition of a DSML for specifying and applying model mutations [3], resembling the syntax and the semantics of the existing Wodel language [15]. In particular, we use GPTBuilder utilities provided by OpenAI to customize the general-purpose GPT model, motivated by the need to have AI agents specialized in a set of tasks [16]. To build MuTaGENe, we first create a knowledge base composed of Wodel artifacts, including the language's grammar and the GitHub supporting pages containing the description of publicly available mutation programs. In addition, we conceived a set of helper functions to assist MuTaGENe in understanding the crafted knowledge. Afterward, we perform the mutation phase using the Chain-of-Thoughts (CoT) strategy to mimic human reasoning to support model mutations using the specified DSML. By running initial experiments, we show that MuTaGENe is capable of understanding and implementing model mutations. Even though the proposed methodology needs an in-depth evaluation to be generalized, it represents a stepping stone to foster further research in the domain.

2 Motivation and Background

2.1 Development of DSMLs

DSMLs are specialized modeling languages tailored to a specific application domain, designed to express the concepts and rules inherent to that domain more precisely than general-purpose languages [14]. While the syntax of a DSML defines the set of rules and structures used to create models within a specific domain, the semantics of a DSML define the meaning of its constructs and how they interact.

DSML development is based on the close collaboration between technical and domain experts. This collaboration is essential for ensuring that the DSML accurately captures domain concepts and meets the needs of its users [7]. In particular, domain experts provide insights into the domain concepts, workflows, and constraints. Technical experts translate these insights into language constructs, ensuring that the language can express the necessary domain-specific operations and rules.

A key challenge in this process is developing the semantics of the language, which is inherently complex and requires a precise definition of the meaning and behavior of language constructs [12]. Thus, an agile and lightweight approach, characterized by rapid prototyping, continuous feedback, and lightweight tools, is essential for effectively managing the complexity of semantics development in the early stages of the process. In this respect, we propose the adoption of LLMs to support the development of proof-of-concept implementations of DSMLs with the aim of providing early feedback to technical and domain experts during the early stages of the language design.

2.2 Usage of LLMs in MDE

The MDE community recently investigated the LLM usage, revealing limitations in understanding domain models [8], automating the model completion [9], and specifying OCL constraints [2]. Overall, there is a need to devise specialized knowledge to support specific tasks. In this respect, OpenAI recently released the *GPT Builder* [21] feature, a dedicated service that offers a specific interface to create custom GPT models. First, the developers can specify the role, the application domain, and the main objectives to specialize the agent. In addition, the system allows the creation of an internal *knowledge base* that can include different files, e.g., project samples or code snippets. The custom agent can use such knowledge during the generation phase, thus equipping the general-purpose GPT model with the knowledge of the application domain of interest. The overall GPT accuracy can be enhanced by employing three leading prompt engineering (PE) strategies [22], i.e., *zero-shot*, *few-shots*, and *Chain-of-Thoughts*. In the *zero-shot* setting [22], the underpinning LLM operates without examples of the expected outcomes. Contrariwise, *few-shot prompting* involves using a limited number of labeled examples, allowing it to quickly generalize and learn the task with minimal supervision [18].

Unlike traditional question-answering tasks, where each question is independent, Chain-of-Thoughts (CoT) [24] requires the model to understand and maintain context from previous interactions throughout the entire conversation. The goal is to evaluate the model's ability to engage in coherent, multi-turn dialogue and infer relationships and dependencies by introducing reasoning steps across the conversation. In this paper, we adopted the CoT prompt strategy, as it outperforms other Prompt Engineering techniques in MDE tasks [10]. In addition, introducing semantic reasoning can improve the LLM capabilities in providing early feedback during DSML specification.

3 Proof of concept

This section describes the essential components of the envisioned approach to support the development of early implementations of DSMLs. To this end, the Wodel mutation language is considered as playground giving us the opportunity to concretely discuss issues and how MuTaGENe can address them.

Wodel and its capabilities are described in Sec. 3.1. Afterward, we describe MuTaGENe and its two constituting phases (see Fig. 1), i.e., *i) knowledge creation*, which serves to set up MuTaGENe and establish its foundational knowledge base enabling the usage of GPT (see Sec. 3.2), and *ii) mutation*, which is achieved through the utilization of CoT prompt engineering, leveraging Wodel rules (see Sec. 3.3). Finally, we present an explanatory output of the approach and its main limitations in Section 3.4.

3.1 Wodel DSML

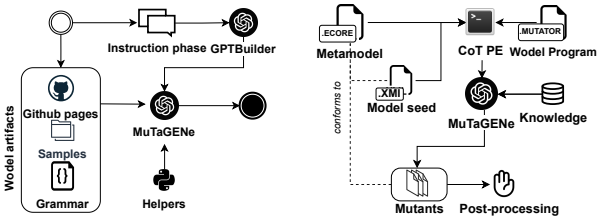
A model mutation is a kind of model manipulation that creates a set of variants (or mutants) of an input model, namely the *seed* model, by the application of one or more mutation operators [4]. In the scope of the paper, we selected Wodel [15] because *i)* it provides a dedicated DSML to express model mutations and *ii)* it offers publicly available materials for fostering mining and reproducibility.

Listing 1: Explanatory Wodel program

```
generate 2 mutants
in "data/out/"
from "data/model/"
metamodel "Families.ecore"

create Member with { firstName = random-string (1,10)}[1..5]

f = select one Family
```



(a) Knowledge creation phase **(b) Mutation phase**

Figure 1: MuTaGENe overview

```
9  daugther = select one Member where {familyDaughter<> null}
10 modify f with {daughters= daugther}
11
12 remove all Member where {familySon<> null}
```

Listing 1 shows an example of Wodel *program* consisting of different mutators. Specifically, the *mutator* defined at line 6 with the `create` *command* specifies the creation of new model elements conforming to the `Member` metaclass of the input metamodel. The mutators specify also that the attribute `firstName` should be assigned with a random string whose length falls within the range of 1 to 10 characters. Creation commands may be annotated with an integer interval indicating the minimum and maximum number of new elements to be generated (as seen in the interval [1..5] at line 6). The execution engine determines the specific number by selecting a value within the specified interval.

The `modify` command is employed to specify mutators altering the content of selected elements. For example, in lines 8-10, the first step is to select one instance of the `Family` metaclass. Subsequently, a `Member` instance with the reference `familyDaughter` not null is randomly chosen. Finally, this selected `Member` instance is assigned to the reference `daughters` of the previously selected `Family` instance.

The `remove` command is employed to define deletions of instances that meet specific criteria. For example, the mutator at line 12 is intended to remove all instances of the `Member` class that have at least one reference to a *son* instance.

3.2 Knowledge creation phase

As previously discussed, *GPT Builder* permits users to create custom GPT models by leveraging a knowledge base, which must be appropriately curated for the particular tasks the model should support. Being MuTaGENe a tool for supporting the creation of Wodel specifications and their execution, we created a knowledge base by ingesting *Wodel artifacts*, Python *Helpers*, and following an *Instruction phase* to configure the role of MuTaGENe, by relying on mechanisms provided by *GPT Builder*. The main ingredients of the knowledge creation phase are shown in Fig. 1a and detailed below.

Wodel artifacts. Three different kinds of *Wodel artifacts* have been mined to create the MuTaGENe knowledge, i.e., the Xtext grammar of the Wodel DSL,[1] information retrieved from the GitHub pages of the approach,[2] with special treatment for sample projects featured on the official website [1].

[1]https://github.com/gomezabajo/Wodel/blob/master/wodel.dsls.wodel/src/wodel/dsls/Wodel.xtext
[2]https://gomezabajo.github.io/Wodel/

Table 1: Overview of the projects in the knowledge

Domain	# of projects	# of mutator files	# of seeds
BPEL	22	22	18
UMLDiagrams	62	62	62
DFASamples	14	14	56

Concerning the Wodel grammar, we loaded the original Xtext file without any modification to avoid any bias. Regarding Wodel GitHub pages, they contain several explanatory Wodel mutators that we opted to encode and include in the knowledge base. Figure 2 displays a fragment of a Wodel program[3] and the corresponding text encoding that we devised to add to the MuTaGENe knowledge the mutation rule for adding a redundant constraint to a given class diagram. For this purpose, we utilized the Beautiful Soup Python library[4] for data extraction. Subsequently, we enriched the context obtained from GitHub by annotating the aforementioned set of rules using the special tag `#USER`, as illustrated in the lower section of Figure 2. In addition, we appended the corresponding `#ECORE` file, a `#SEED` model, the mutations `#RULES`, and the `#GENERATED MUTANT(s)` to the text file (not shown in Fig. 2 for the sake of space, but available online).[5]

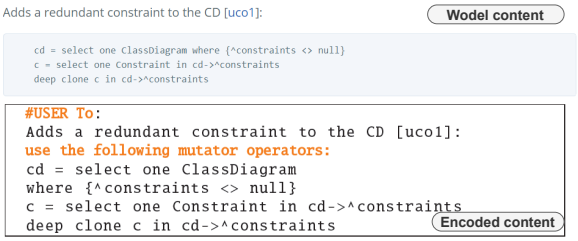


Figure 2: Fragment of an encoded Wodel GitHub page content

Concerning the sample projects, we mined 98 Wodel projects, consisting of 98 program files, and 136 seed models as shown in Tab. 1. For each project, we performed a pre-processing step on the original folder by extracting the metamodel, the seed models, and the Wodel programs. The remaining files, e.g., Eclipse configuration files or the outputs of Wodel commands, have been filtered out to avoid noisy data in the knowledge.

Helpers. After loading the above-mentioned static files related to Wodel, we enhanced the knowledge base with Python functions devoted to improving the reasoning of MuTaGENe as detailed in the *Instructions* phase discussed below. In particular, we defined the following helpers:

➢ *grammar_utils.py*: This file contains a set of tailored functions employed to parse the Wodel grammar syntax. In particular, we devise a function that reads a Wodel program file content and extracts the DSL keywords to understand the mutation logic. Notably, the introduction of those helper functions enables the understanding of Wodel syntax. Furthermore, this enrichment of the GPT local knowledge reduces the failures in generating mutants;

[3]https://gomezabajo.github.io/Wodel/umlcd.html
[4]https://pypi.org/project/beautifulsoup4/
[5]https://github.com/MDEGroup/ModelMutator-Replication-Package/blob/main/GPT_KB/umlcd.htmltxt

➢ *parse_documentation.py:* This function is used to analyze the parsed GitHub documentation depicted in Figure 2, thus helping MuTaGENe during the reasoning phase.
➢ *mapping_models.py:* This file contains a dedicated set of functions that exploits the PyEcore library[6] to parse the modeling artifacts, i.e., the metamodel and the seed model expressed in .ecore and .xmi format, respectively.

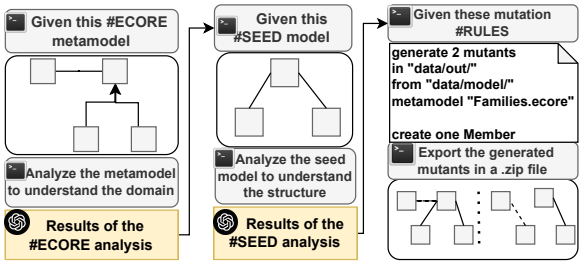


Figure 3: CoT prompt

Instructions. Employing the "Instructions" mechanism of GPT Builder, it is possible to configure the model by offering structured guidelines and examples so that the AI understands the task to be supported. Thus, we configured MuTaGENe by following an iterative process, where we used the instruction prompting technique [25] to enable the AI to understand Wodel commands and apply them to input seed models. During the first iteration of the process, the AI model could not parse the input Wodel commands or correctly execute them on the given seed model. Then we decided to define the helpers previously presented, add them to the knowledge base, and refer them to the specified instructions, including the following one: *"The GPT references local files for producing models tailored to software engineering and MDE needs, utilizing the "grammar_utils.py" file for understanding the Wodel grammar"*, thus ensuring that MuTaGENe has access to the defined helpers.[7]

It is worth mentioning that all the steps outlined in this section, such as loading Wodel artifacts and helpers, followed by the instruction phase, are preparatory and executed only once. However, we anticipate a maintenance phase where we consistently update the defined knowledge to enhance the quality of the models generated by MuTaGENe.

3.3 Mutation phase

After the configuration phase detailed in the previous section, MuTaGENe can execute a Wodel program for mutating input models according to the process depicted in Fig. 1b. The user's request comprises the seed model, the corresponding metamodel, and the mutation programs specified in Wodel. Regarding the prompt engineering strategy, we have chosen the Chain-of-Thoughts (CoT) strategy outlined in Section 2. This decision stems from the impracticality of employing zero-shot techniques for larger modeling artifacts, given the limitation on input tokens for each query. Additionally, prior research demonstrates that CoT can outperform both zero-shot and few-shot strategies for MDE tasks [10].

[6]https://pyecore.readthedocs.io/en/latest/
[7]https://chatgpt.com/g/g-7GQpuc3wx-mutagene

The employed CoT strategy is illustrated in Fig. 3. Initially, we present the metamodel and the seed model in two separate prompts to address token limitation issues. Afterward, the Wodel program to be executed is given. Following each artifact, a dedicated prompt asks the model to analyze it.

Artifacts are annotated with the tags `#ECORE`, `#SEED`, and `#RULES` to enhance the reasoning phase of the model. The final prompt requests the model to generate mutants, which are then returned in a package file that can be downloaded. We acknowledge that not all the mutants are syntactically valid, e.g., we report minor issues in the model structure or incorrect model URIs. Therefore, a manual *post-processing* phase is needed to fix and adapt the produced mutants to the selected model environment. For instance, Chen et al. [10] used a similar strategy to validate the generation of domain models.

3.4 Explanatory example

As a motivating example, we rely on an existing Wodel project that applies mutations Probabilistic Finite State Machine (PFSM).[8] It is worth noting that this project is not part of the proposed knowledge base described in Section 3.2. Figure 4a and Figure 4b show the XMI representation of the mutants obtained by Wodel and MuTaGENe, respectively.[9] Created elements are highlighted in **green**, and in **blue** are the modified ones. The applied Wodel program comprises mutators designed to select two *transitions* from the seed model and decrement their probabilities by a random integer value. It is essential to note that custom GPT models are integrated with Bing browsing functionality, enabling them to retrieve artifacts from the Web to respond to user queries without executing the mutators. To prevent this scenario and avoid bias in the evaluation, we adjusted the original Wodel programs found in GitHub repositories, i.e., introducing the element `states name="class-control-alp"`. Concerning the limitations, we acknowledge that the selection of files to be included in the KB could introduce bias in the generation phase. To address this, we pre-processed the Wodel sample projects by removing unnecessary data for the mutation process. As for the mutation phase, the created mutants may include irrelevant information due to incorrect program encodings. We tackled this by adopting the CoT strategy combined with self-consistency evaluation, which involves repeating the mutation process ten times for each testing project to reduce imprecise content. Another limitation is the generalizability to support other DSMLs apart from the Wodel one. We anticipate that additional strategies can be more effective in employing LLMs in practice. For instance, retrieval augmented generation (RAG) [17] can be seen as an alternative method to increase the effectiveness of LLMs to support the development of DSMLs.

4 Conclusion and future works

During the definition of a domain-specific modeling language (DSML), developers need to interact with domain experts to specify the necessary modeling artifacts, including the development of the language semantics. This paper envisions using large language

[8]https://gomezabajo.github.io/Wodel/pfsm.html
[9]Due to the page limitation, we omitted the elements of the model that are not affected by the rules

```
<?xml version="1.0" encoding="UTF-8"?>
<PFSM:PFSM>
...
<states name="class-control-alp"/>
<transitions ... probability="-0.07"/>
<transitions ... probability=0.25"/>
<transitions ... probability="1.07"/>
<transitions ... probability="1.0"/>
<transitions ... probability="1.0"/>
<transitions ... probability="0.75"/>
...
</PFSM:PFSM>
```

(a) Wodel output

```
<?xml version="1.0" encoding="UTF-8"?>
<PFSM:PFSM>
...
<states name="class-control-alp"/>
<transitions ... probability="0.78"/>
<transitions ... probability="0.25"/>
<transitions ... probability="1.0"/>
<transitions ... probability="1.0"/>
<transitions ... probability="1.0"/>
<transitions ... probability="0.96"/>
...
</PFSM:PFSM>
```

(b) MuTaGENe output

Figure 4: Modify and Create mutators applied to the seed model of the PFSM project

models (LLMs) to provide early feedback during the DSML specification phase. Built on top of a specific DSML for model mutation, we create a custom GPT model called MuTaGENe that can resemble the expected outputs, understanding the concepts and rules expressed with the Wodel language. Although an in-depth evaluation is needed to confirm our intuition, LLMs can help articulate language semantics in natural language, enabling stakeholders to validate language alignment with user requirements. In future work, we plan to extend the application of the custom GPT to additional DSMLs and experiment with open-source LLMs and techniques such as fine-tuning or RAG to improve support during the entire DSML specification process.

Acknowledgments

This work has been partially supported by the EMELIOT national research project, which has been funded by the MUR under the PRIN 2020 program grant n. 2020W3A5FY. The work has been also partially supported by the European Union–NextGenerationEU through the Italian Ministry of University and Research, Projects PRIN 2022 PNRR *"FRINGE: context-aware FaiRness engineerING in complex software systEms"* grant n. P2022553SL. We acknowledge the Italian "PRIN 2022" project *"TRex-SE: Trustworthy Recommenders for Software Engineers,"* grant n. 2022LKJWHC, and the research project *"RASTA: Realtà Aumentata e Story-Telling Automatizzato per la valorizzazione di Beni Culturali ed Itinerari"*; Italian MUR PON Proj. ARS01 00540.

References

[1] Gomez Abajo. 2024. Wodel Samples. https://gomezabajo.github.io/Wodel/samples.html. Last Accessed:22-03-2024.

[2] Seif Abukhalaf, Mohammad Hamdaqa, and Foutse Khomh. 2023. On Codex Prompt Engineering for OCL Generation: An Empirical Study. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. 148–157. https://doi.org/10.1109/MSR59073.2023.00033

[3] Bakr Al-Batran, Bernhard Schätz, and Benjamin Hummel. 2011. Semantic Clone Detection for Model-Based Development of Embedded Systems. In *Model Driven Engineering Languages and Systems (Lecture Notes in Computer Science)*, Jon Whittle, Tony Clark, and Thomas Kühne (Eds.). Springer, Berlin, Heidelberg, 258–272. https://doi.org/10.1007/978-3-642-24485-8_19

[4] Manar H. Alalfi, James R. Cordy, Thomas R. Dean, Matthew Stephan, and Andrew Stevenson. 2012. Models are code too: Near-miss clone detection for Simulink models. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, Trento, Italy, 295–304. https://doi.org/10.1109/ICSM.2012.6405285

[5] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-Driven Software Engineering in Practice*. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-031-02549-5

[6] Antonio Bucchiarone, Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio (Eds.). 2021. *Domain-Specific Languages in Practice: with JetBrains MPS*. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-030-73758-0

[7] Loli Burgueño, Federico Ciccozzi, Michalis Famelis, Gerti Kappel, Leen Lambers, Sebastien Mosser, Richard F. Paige, Alfonso Pierantonio, Arend Rensink, Rick Salay, Gabriele Taentzer, Antonio Vallecillo, and Manuel Wimmer. 2019. Contents for a Model-Based Software Engineering Body of Knowledge. *Software and Systems Modeling* 18, 6 (Dec. 2019), 3193–3205. https://doi.org/10.1007/s10270-019-00746-9

[8] Javier Cámara, Javier Troya, Lola Burgueño, and Antonio Vallecillo. 2023. On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. *Software and Systems Modeling* 22, 3 (June 2023), 781–793. https://doi.org/10.1007/s10270-023-01105-5

[9] Meriem Ben Chaaben, Lola Burgueño, and Houari Sahraoui. 2023. Towards Using Few-Shot Prompt Learning for Automating Model Completion. In *Proceedings of the 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER '23)*. IEEE, IEEE Press, Melbourne, Australia, 7–12. https://doi.org/10.1109/ICSE-NIER58687.2023.00008

[10] Kua Chen, Yujing Yang, Boqi Chen, José Antonio Hernández López, Gunter Mussbacher, et al. 2023. Automated Domain Modeling with Large Language Models: A Comparative Study. In *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 162–172. https://doi.org/10.1109/MODELS58315.2023.00037

[11] Claudio Di Sipio, Juri Di Rocco, Davide Di Ruscio, and Phuong T Nguyen. 2023. MORGAN: a modeling recommender system based on graph kernel. *Software and Systems Modeling* (April 2023), 1–23. https://doi.org/10.1007/s10270-023-01102-8

[12] Sebastian Erdweg, Tijs van der Storm, Markus Völter, Meinte Boersma, Remi Bosman, William R. Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh, Gabriël D. P. Konat, Pedro J. Molina, Martin Palatnik, Risto Pohjonen, Eugen Schindler, Klemens Schindler, Riccardo Solmi, Vlad A. Vergu, Eelco Visser, Kevin van der Vlist, Guido H. Wachsmuth, and Jimi van der Woning. 2013. The state of the art in language workbenches. In *Software language engineering*, Martin Erwig, Richard F. Paige, and Eric Van Wyk (Eds.). Springer International Publishing, 197–217.

[13] Martin Fowler. 2010. *Domain-specific languages*. Pearson Education.

[14] Ulrich Frank. 2013. *Domain-specific modeling languages: Requirements analysis and design guidelines*. Springer Berlin Heidelberg, Berlin, Heidelberg, 133–157. https://doi.org/10.1007/978-3-642-36654-3_6

[15] Pablo Gómez-Abajo, Esther Guerra, and Juan De Lara. 2016. Wodel: a domain-specific language for model mutation. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM, Pisa Italy, 1968–1973. https://doi.org/10.1145/2851613.2851751

[16] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, et al. 2023. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv:2308.00352 [cs.AI]

[17] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[18] Xuan Li, Shuai Yuan, Xiaodong Gu, Yuting Chen, and Beijun Shen. 2024. Few-shot code translation via task-adapted prompt learning. *Journal of Systems and Software* 212 (2024), 112002. https://doi.org/10.1016/j.jss.2024.112002

[19] Gunter Mussbacher, Benoît Combemale, Jörg Kienzle, Silvia Mara Abrahão, Hyacinth Ali, Nelly Bencomo, Márton Búr, Loli Burgueño, Gregor Engels, Pierre Jeanjean, Jean-Marc Jézéquel, Thomas Kühn, Sébastien Mosser, Houari A. Sahraoui, Eugene Syriani, Dániel Varró, and Martin Weyssow. 2020. Opportunities in intelligent modeling assistance. *Software and Systems Modeling* 19 (2020), 1045–1053.

[20] Phuong T Nguyen, Juri Di Rocco, Ludovico Iovino, Davide Di Ruscio, and Alfonso Pierantonio. 2021. Evaluation of a machine learning classifier for metamodels. *Software and Systems Modeling* 20, 6 (2021), 1797–1821.

[21] openAI. 2024. Creating a GPT | OpenAI Help Center. https://help.openai.com/en/articles/8554397-creating-a-gpt

[22] Bernardino Romera-Paredes and Philip H. S. Torr. 2015. An embarrassingly simple approach to zero-shot learning. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37* (Lille, France) *(ICML'15)*. JMLR.org, 2152–2161.

[23] Rijul Saini, Gunter Mussbacher, Jin L. C. Guo, and Jörg Kienzle. 2020. DoMoBOT: A Bot for Automated and Interactive Domain Modelling. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (Virtual Event, Canada) *(MODELS '20)*. Association for Computing Machinery, New York, NY, USA, Article 45, 10 pages. https://doi.org/10.1145/3417990.3421385

[24] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html

[25] Xuansheng Wu, Wenlin Yao, Jianshu Chen, Xiaoman Pan, Xiaoyang Wang, Ninghao Liu, and Dong Yu. 2024. From Language Modeling to Instruction Following: Understanding the Behavior Shift in LLMs after Instruction Tuning. arXiv:2310.00492 [cs.CL]