An Automated Diagram Generator of Reference Solutions for Modeling Educators

An Automated Diagram Generator of Reference Solutions for Modeling Educators

Giacomo Garaccione[1 [0000−0001−7254−9578]], Riccardo Coppola[1 [0000−0003−4601−7425]], and Luca Ardito[1 [0000−0002−0501−7886]]

Politecnico di Torino, Torino, Italy
first.last@polito.it

Abstract. UML class diagrams are a relevant modeling language in Software Engineering education since they can be used to teach students how to visualize and display the different entities that compose a system, with their functionalities and relationships. The definition of modeling exercises and their evaluation can be time-consuming for educators due to the need to consider possible semantic variations and alternative representations of the same system requirements. To facilitate teachers in this process, we present TIGRE (auTomated dIagram Generator of REference solutions), an online editor for the definition of UML modeling exercises where teachers can define reference solutions in the form of both diagrams and detailed structures to be used for automated evaluation. The tool is enhanced by the interaction with recent Large Language Models for the automated generation of reference solutions starting from text, facilitating the creation of early drafts. A proof-of-concept case study has been performed by having TIGRE generate reference solutions for two exercises: most of the relevant concepts have been represented correctly, but issues emerged in the form of unnecessary classes being included and incorrect understanding of associations.

Keywords: UML Class Diagrams · Software Modeling · Large Language Models · Education

1 Introduction

Software modeling plays a pivotal role in Software Engineering (SE) education, as it allows students to develop critical thinking and the ability to represent real-world problems in an easily understandable visual form.

Among the various modeling types, conceptual modeling is particularly relevant, as it lets students learn how to visualize the different entities involved in a system, their functionalities, and the relationships among them.

One of the most commonly used notations in SE education is the Unified Modeling Language (UML), which provides several diagram types (e.g. class, use case, activity, deployment). UML class diagrams are considered to be one of the most suitable notations for conceptual modeling activities thanks to their ability to represent the different concepts involved in a system, their characterizing attributes and methods, and different relationship types.

Learning UML usually requires students to solve exercises in the form of textual assignments, where a given system is described and the goal is to produce a satisfying model [1]. It is possible for students to produce solutions that differ from the original one envisioned by the teacher, either due to different element names or representing concepts in an equivalent (but still correct) way. Several tools and methodologies exist to perform UML modeling exercises [2], although some issues are presented such as tools not being available for use, creating and evaluating exercises being a time-consuming process that is rarely assisted by dedicated tools, and the inability to provide individual feedback to students.

To facilitate the evaluation process of UML class diagrams, and assist teachers in the definition of modeling exercises, we present TIGRE (auTomated dIagram Generator of REference solutions), an intelligent web-based platform that enhances the creation of diagrams by providing a modeling canvas where teachers can draw references, and the ability to interact with recent Large Language Models (LLMs) for the creation of preliminary sketches. The tool supports the definition of reference data structures for evaluating the students' diagrams, as one such structure contains the list of required elements, together with synonyms to ensure enough variability in the solution space is covered.

In this paper, TIGRE is presented as a prototype and its capabilities are assessed with a preliminary proof-of-concept example: by using the tool, we generated the solutions for two modeling exercises and then compared them with the original reference solutions; the evaluation showed that the most relevant concepts could be represented, although some issues were encountered in the form of unnecessary elements being added, incorrect representation of associations, and a tendency to represent associations as separate classes.

The remainder of the paper is structured as follows: Section 2 presents the background information with regards to the current state of the art, Section 3 describes the tool architecture, its functionalities, and the intended usage scenarios of the tool. In Section 4 we then describe the proof-of-concept example and discuss our findings and, lastly, we draw our conclusions and describe our future plans for TIGRE in Section 5.

2 Background and Related Work

There are various examples of tools for the automatic generation of class diagram exercises in literature. One such example is Wodel-EDU [3, 6], a modeling tool built on the domain-specific language (DSL) Wodel [5]; through Wodel, teachers can define diagram constraints for an exercise and then have a student-produced solution be automatically evaluated to see how many constraints are respected. Wodel-EDU exploits the DSL by supporting several diagram types and, through a meta-model of a diagram and the corresponding graphical representation, allows for the generation of mutants that can be considered variations of a base diagram.

AutoER, proposed by Foss et al. [4], is a question-generation system for UML database design exercises that provides all requirements of a problem whose parts

students can select to add new constructs to a diagram. This diagram is then compared with a teacher-defined diagram to identify matching names for entities, attributes, keys, and associations. Visual Narrator [7] is a tool built on a custom algorithm that exploits natural language processing techniques to convert user stories into conceptual models. The algorithm identifies concepts, relationships, attributes, and cardinalities.

Saini et al. [8] describe an approach built on machine learning and NLP techniques through a bot that receives textual input and continuous user feedback to generate and iteratively improve a class diagram: this is done by selecting decision points in the text and producing variations based on them, which the user can then choose as starting points for continuous improvement.

Yang et al. [9] describe a similar approach, where a textual description is processed by classifying sentences to identify classes, attributes, and associations, with grammar-based rules to then compose the various elements that make up a diagram. An evaluation performed on the tool yielded low precision and recall scores, painting the solution as inaccurate and unable to handle synonyms, ambiguities, and complex phrases.

The proposed approaches appear to be effective in some scenarios (with some being more effective than others) but are not completely generalizable. Moreover, none of the existing solutions appear to offer full support for editing diagrams directly, but instead rely on textual specifications, thus reducing the ability for users to define models with ease.

The use of LLMs for the creation of UML class diagrams has been explored in recent years by some preliminary studies [10–12] that highlight the potential of using such an approach for assisting teachers in the definition of class diagram exercises and references. To the best of our knowledge, no tools exist yet that directly integrate the LLM functionality in a modeling tool, as the aforementioned studies all make use of OpenAI's ChatGPT to produce the diagrams. Our aim is to provide an editor where the diagrams produced by an LLM can be edited with ease, rather than adopting strategies such as PlantUML that rely on text editing to make changes to a diagram. The same editor is intended to be used by students, providing them an easy interface for solving exercises. Additionally, with the definition of a reference structure, we intend to provide a replicable and reliable approach for the evaluation of class diagrams, without relying on one specific strategy to represent a diagram.

3 Tool Features

TIGRE is currently under development as a React-based web application that uses Apollon [13], an open-source online modeling editor that supports UML class diagrams. The application exploits HuggingFace's API services, allowing users to select the model they prefer to use. For this preliminary proof of concept, we selected DeepSeek's R1-Distill-Qwen-32B [14] as the model to use; we chose this model due to it showing good performance in reasoning while being, at

the time of writing the article (April 2025), one of the most recent open-source LLMs.

In its current implementation, TIGRE allows SE teachers to define modeling exercises by providing a unique title and the textual description that accompanies the assignment. Reference solutions can then be defined for each existing exercise, and are identified by two objects, a *model* and a *structure*.

A *model* object contains all the information needed to draw a diagram and make it compatible with the Apollon modeler: it lists all classes present in the diagram, with their attributes and eventual methods, together with the graphical details needed to draw each class (e.g. width, height, position in the canvas); relationships are also included and are characterized by their type, the names of the two connected classes, the cardinality, and graphical details such as the starting points on each class and the relative position of the target class to the source.

A *structure* object, instead, is detached from the graphical representation of a class diagram and is intended to be used as a list of necessary elements that a student-produced diagram is expected to include when solving an exercise. Each structure object has the following attributes:

- A list of necessary classes. Each class has a unique name, a list of synonyms, a weight that expresses the importance of the class, a list of attribute names that the class cannot have, and a list of expected attributes. For each attribute, the following details are known: its name, its list of synonyms, its weight, and a list of accepted types.
- A list of necessary associations. Each association specifies the source and target class, a name, a list of synonyms, its type, and its weight. For both the source and target class, the class name, the eventual role that the class plays in the association, and a list of allowed multiplicities for the class are known.
- A list of necessary enumerations. Enumerations are similar to classes in terms of known attributes since they also have a unique name, a list of synonyms, a weight, and a list of literals. Literals are a special case of attributes, with the only difference being the list of allowed types, which is hard-coded to be an empty and unmodifiable array.
- A list of associations between an enumeration and a class. These associations specify the enumeration and the class, without adding the additional information specified by regular associations.
- A list of names that cannot be used to represent a class.
- A list of pairs of class names, with each item representing two classes that cannot be connected by an association.

Teachers can choose to create a new reference solution for an exercise by either drawing it on the modeling canvas, specifying directly the classes and their associations in an easily displayed and editable manner or through a dedicated form where, for each list of the structure object, it is possible to add a new item of the corresponding type or edit one of the existing items.

Additionally, both functionalities can also be enhanced through the interaction with the LLM: instead of drawing an entire diagram, it is possible to send the textual description of the exercise to DeepSeek and receive as an answer the list of expected elements that should be included. The answer is parsed and converted into a *model* object that can be displayed directly on the canvas, thus creating a preliminary sketch that teachers can use as a starting point for the full solution. The same can also be done for the creation of a starting *structure*: the answer given by DeepSeek is converted into a structure object that is set as the starting data for the form. Any time a reference is saved, no matter if it is a model or structure, a new object of the other type is also created with the same exact data, ensuring full compatibility between the two representations. Additionally, it is possible to enhance a structure object by passing it to DeepSeek and generate a list of synonyms for each named element, which teachers can then edit and validate; these synonyms can be easily revised through a dedicated form and removed from a structure in case they are not suitable. This aims to produce variation in the possible solution space, a goal that is further strengthened by the tool letting teachers define multiple alternative solutions (either by themselves or with the LLM's assistance) for a single exercise.

4 Proof-of-Concept Case

We present a proof-of-concept case study to test TIGRE's initial capabilities, identify potential issues and improvement areas, and define future development goals. Our objective is to evaluate TIGRE's integration with LLMs for editing UML class diagrams, focusing on the effectiveness of its automated diagram generation. We tested TIGRE by generating two class diagrams from textual requirements and analyzed the results.

The exercise requirements were sourced from past exams of the Information Systems course at Politecnico di Torino, where conceptual modeling is taught as a course topic. We used teacher-defined reference solutions to evaluate TIGRE's diagrams, using the same input descriptions given to the students during the exam. The exercises are labeled *Hackathons* and *Ski Passes*. The full text of the two exercises, the reference solutions, the generated diagrams, the results of our evaluation, and the prompt used for the generation of class diagrams are all available as an online resource[1].

We evaluated the diagrams using the method by Nikiforova et al. [15], which compares UML diagrams based on semantic similarity in four categories: classes, attributes, methods, and relationships. Matching elements are paired by human judgment and a similarity score is assigned to each pair; the distance scores among all pairs are used to create an Euclidean distance vector that yields the final distance score between two diagrams. The smaller the score, the higher the similarity, with 0 meaning two equivalent diagrams.

The comparison followed these steps: i) We matched each reference class with the most semantically similar generated class (and vice-versa), according to the

[1] `https://doi.org/10.6084/m9.figshare.28574861`

best combination of name, attributes, and associations; unmatched classes received a distance score of 1 and enumerations were treated as classes and matched following the same criteria. ii) For attributes, we first checked pairs of matching classes, and matched their attributes according to name and type; unmatched attributes, and attributes of unmatched classes scored 1. Enumeration literals followed similar criteria, and were matched by name only. iii) Associations were matched if both classes were matched and the type (e.g. regular association, inheritance) and cardinality were equal; links between classes and enumerations were also considered associations and scored accordingly. iv) Once all elements were compared, we calculated the individual distances for classes, attributes, and associations, and we then computed the overall diagram distance based on them. This procedure was performed by one author of the paper; the remaining two authors then reviewed independently the evaluation, leading to the three authors converging their interpretations of the results into the final version.

Given the small sample size, we opted for qualitative analysis over statistical methods. Figures 1 and 2 display the two diagrams generated by the LLM, while Table 1 reports all evaluation metrics: counts of elements and computed distances for each exercise.

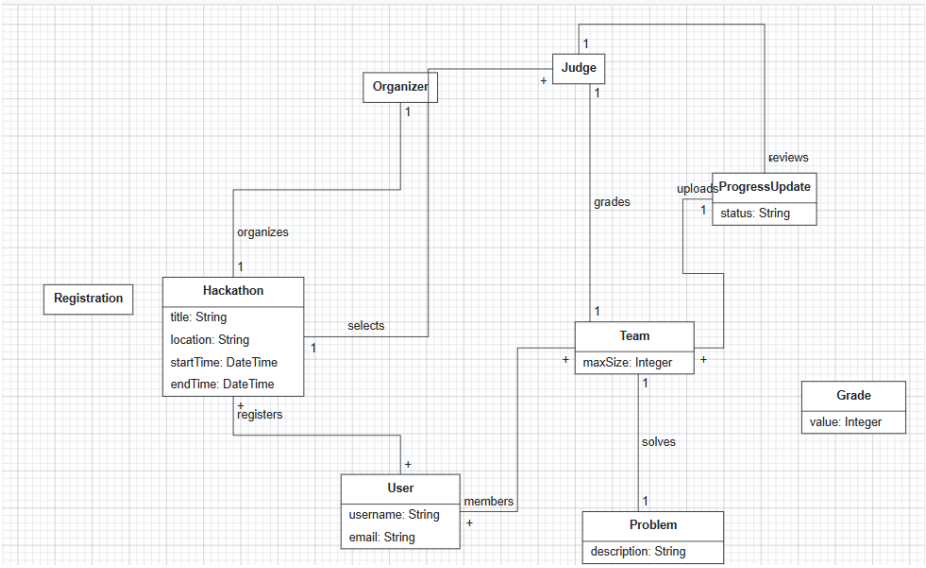

Fig. 1. Generated solution for the *Hackathon* exercise

Class distance scores were relatively low in both exercises, indicating somewhat good class generation capabilities. In *Hackathon*, 7 generated classes matched the reference, though 3 had name mismatches and 2 were unnecessary, with one representing an irrelevant concept and the other replacing an attribute. In *Ski*

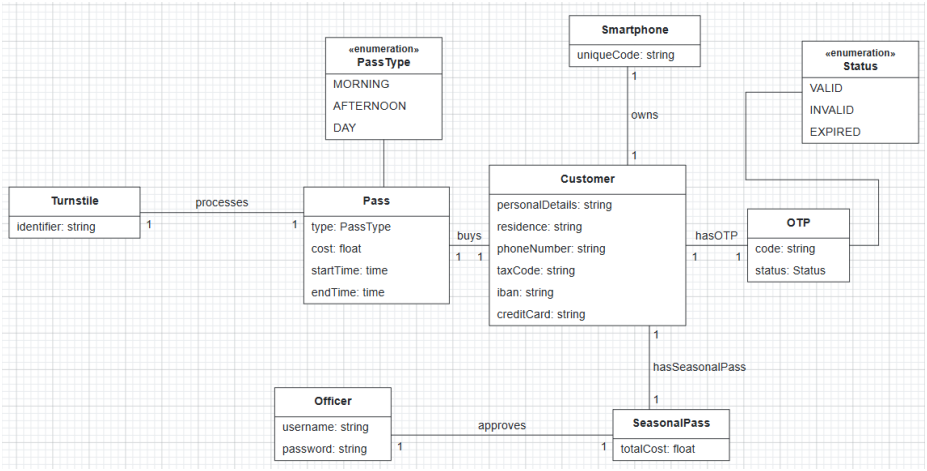

Fig. 2. Generated solution for the *Ski Passes* exercise

Table 1. Summary metrics for the two exercises

Metric	Hackathons	Ski Passes
Reference Classes	10	7
Reference Attributes	19	26
Reference Associations	14	6
Generated Classes	9	9
Generated Attributes	10	22
Generated Associations	8	8
Class Distance	2.40	2.96
Attribute Distance	4.58	5.83
Association Distance	3.87	3.16
Diagram Distance	6.46	7.26

Passes, only 4 out of 7 reference classes had matches, and the generated diagram included 5 unnecessary classes that would replace attributes or represent irrelevant content.

Attributes were the elements with the highest distance scores. In *Hackathon*, only 7 of 19 reference attributes had matches, and only 2 were exact matches with both name and type. *Ski Passes* showed similarly low results: 9 of 26 reference attributes had matches, with 2 attributes and 3 literals being exact. The second exercise also had 9 unmatched generated attributes and an unnecessary enumeration, compared to 4 unnecessary attributes in *Hackathon*.

Despite having relatively low distance scores, association generation also had some issues. In *Hackathon*, 6 of 14 reference associations had a matching name but only one also had the correct cardinality. This issue was more prominent in *Ski Passes* where, although 3 reference associations had a match, all generated

associations used incorrect 1-1 cardinalities, showing a flawed understanding of the problem to represent.

Overall, the generated diagrams appear to have captured the most relevant concept in both exercises and can offer a good starting point for defining modeling exercises' solutions. However, some limits such as introducing unnecessary elements, placing attributes in wrong classes, using classes for concepts that should be attributes, and focusing on one-to-one cardinalities are present and reduce the overall correctness.

The integration of LLMs in the diagram drawing process is not intended to fully replace human intervention, meaning that the produced diagrams are not supposed to be used as final output, but rather templates with the basic information that can then be edited and improved, since the textual answer is already parsed and converted in an easily editable class diagram. The ability to remove/edit unnecessary elements with a simple interaction offer more flexibility compared to the existing works in the literature, where the output was only textual or in PlantUML format, making it less flexible to edit.

5 Conclusions and Future Work

In this paper, we described TIGRE, a prototype web application that enhances the definition of UML modeling exercises through interaction with LLMs and the definition of dedicated reference structures for the evaluation of student-produced diagrams (both manual and automated) . We performed a proof-of-concept test by generating the solutions to two exercises using the tool and then compared them with the original solutions: results were on the mixed side, since while the produced diagrams contained most of the required elements, the LLM also included different unnecessary elements and had trouble representing associations correctly.

These issues are however mitigated by the implementation of the UML modeler: rather than being simple textual content, the answers given by the LLM are automatically converted in an easily editable class diagram, facilitating the creation of reference solutions. The presence of a form where solution structures can be edited directly after creating a diagram also ensures that synonyms and semantic variations are covered, especially thanks to the integration with the LLM for this functionality.

Our future plans for the development of TIGRE include a refinement of the prompting strategies and the development of an evaluation engine that uses the reference structures as the base for assisting students by providing direct feedback through automated diagram comparisons. Lastly, we envision the extension of all functionalities (automated diagram creation, reference solutions, automated evaluation) to other diagram types such as process and use case diagrams.

References

1. Huber, F. & Hagel, G. Work-in-Progress: Towards detection and syntactical analysis in UML class diagrams for software engineering education. *2020 IEEE Global Engineering Education Conference (EDUCON)*. pp. 3-7 (2020)
2. Huber, F. & Hagel, G. Tool-supported teaching of UML diagrams in software engineering education - A systematic literature review. *2022 45th Jubilee International Convention On Information, Communication And Electronic Technology (MIPRO)*. pp. 1404-1409 (2022)
3. Gómez-Abajo, P., Guerra, E. & Lara, J. Automated generation and correction of diagram-based exercises for Moodle. *Computer Applications In Engineering Education*. **31**, 1845-1866 (2023), https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.22676
4. Foss, S., Urazova, T. & Lawrence, R. Automatic Generation and Marking of UML Database Design Diagrams. *Proceedings Of The 53rd ACM Technical Symposium On Computer Science Education - Volume 1*. pp. 626-632 (2022), https://doi.org/10.1145/3478431.3499376
5. Gómez-Abajo, P., Guerra, E. & Lara, J. Wodel: a domain-specific language for model mutation. *Proceedings Of The 31st Annual ACM Symposium On Applied Computing*. pp. 1968-1973 (2016,4), https://dl.acm.org/doi/10.1145/2851613.2851751
6. Gómez-Abajo, P., Guerra, E. & Lara, J. A domain-specific language for model mutation and its application to the automated generation of exercises. *Computer Languages, Systems & Structures*. **49** pp. 152-173 (2017,9), https://www.sciencedirect.com/science/article/pii/S147784241630094X
7. Robeer, M., Lucassen, G., Werf, J., Dalpiaz, F. & Brinkkemper, S. Automated Extraction of Conceptual Models from User Stories via NLP. *2016 IEEE 24th International Requirements Engineering Conference (RE)*. pp. 196-205 (2016,9), https://ieeexplore.ieee.org/document/7765525, ISSN: 2332-6441
8. Saini, R., Mussbacher, G., Guo, J. & Kienzle, J. Machine learning-based incremental learning in interactive domain modelling. *Proceedings Of The 25th International Conference On Model Driven Engineering Languages And Systems*. pp. 176-186 (2022), https://dl.acm.org/doi/10.1145/3550355.3552421
9. Yang, S. & Sahraoui, H. Towards automatically extracting UML class diagrams from natural language specifications. *Proceedings Of The 25th International Conference On Model Driven Engineering Languages And Systems: Companion Proceedings*. pp. 396-403 (2022,11), https://dl.acm.org/doi/10.1145/3550356.3561592
10. De Bari, D., Garaccione, G., Coppola, R., Torchiano, M. & Ardito, L. Evaluating Large Language Models in Exercises of UML Class Diagram Modeling. *Proceedings Of The 18th ACM/IEEE International Symposium On Empirical Software Engineering And Measurement*. pp. 393-399 (2024), https://dl.acm.org/doi/10.1145/3674805.3690741
11. Shehata, M., Lepore, B., Cummings, H. & Parra, E. Creating UML Class Diagrams with General-Purpose LLMs. *2024 IEEE Working Conference On Software Visualization (VISSOFT)*. pp. 157-158 (2024,10), https://ieeexplore.ieee.org/abstract/document/10794946, ISSN: 2832-6555
12. Wang, B., Wang, C., Liang, P., Li, B. & Zeng, C. How LLMs Aid in UML Modeling: An Exploratory Study with Novice Analysts. *2024 IEEE International Conference On Software Services Engineering (SSE)*. pp. 249-257 (2024,7), https://ieeexplore.ieee.org/abstract/document/10664407/references

13. Krusche, S., Frankenberg, N., Reimer, L. & Bruegge, B. An interactive learning method to engage students in modeling. *Proceedings Of The ACM/IEEE 42nd International Conference On Software Engineering: Software Engineering Education And Training*. pp. 12-22 (2020), https://dl.acm.org/doi/10.1145/3377814.3381701
14. DeepSeek-AI DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. (2025), https://arxiv.org/abs/2501.12948, _eprint: 2501.12948
15. Nikiforova, O., Gusarovs, K., Kozacenko, L., Ahilcenoka, D. & Ungurs, D. An Approach to Compare UML Class Diagrams Based on Semantical Features of Their Elements. (2015,11)