A methodology of automatic class diagrams generation from source code using Model-Driven Architecture and Machine Learning to achieve Energy Efficiency

Abir Sajji[1], Yassine Rhazali[2], Youssef Hadi[1]

[1]Computer Research Laboratory, Faculty of Science, Ibn Tofail University, Kenitra, Morocco; ORCID: 0000-0003-3672-947X
[2] Information and Communication Systems Engineering Research Group, Higher School of Technology, Moulay Ismail University, Meknes, Morocco;

Abstract. The automated generation of class diagrams is a crucial task in software engineering, facilitating the understanding, analysis, and documentation of complex software systems. Traditional manual approaches are time and energy consuming, error-prone, and lack consistency. To address these challenges, this research presents an automated proposed approach that utilizes Graph Neural Networks (GNNs), a machine learning algorithm, to generate class diagrams from source code within the context of Model Driven Architecture (MDA) and reverse engineering. A comprehensive case study is conducted to compare the results obtained from the automated approach with manually created class diagrams. The GNN model demonstrates high accuracy in capturing the system's structure, associations, and relationships. Notably, the automated approach significantly reduces the time required for class diagram generation, leading to substantial time and energy savings. By advancing automated software documentation, this research contributes to more efficient software engineering practices. It promotes consistency, eliminates human errors, and enables software engineers to focus on higher-value tasks. Overall, the proposed approach showcases the potential of GNNs in automating class diagram generation and its practical benefits for software development and documentation.

Index Terms—MDA, ML, GNN, AI, energy, class diagrams, source code, reverse engineering.

1 Introduction

In modern software development, understanding the structure and dependencies of a software system is crucial for effective maintenance, evolution, and collaboration among developers. Class diagrams, which visually represent classes, attributes, methods, and their relationships, play a vital role in capturing the essence of a software system's architecture. However, creating class diagrams manually from complex and extensive source code can be time and energy-consuming. [1]

To address this challenge, researchers have turned to the power of Artificial Intelligence (AI) and machine learning to automate the process of generating class diagrams from

source code. Among the emerging AI techniques, Graph Neural Networks (GNNs) have shown promise in effectively analyzing graph-structured data and capturing the inherent relationships within. Several studies have been performed on the reverse engineering of UML diagrams.

In [2], the authors propose a method based on Petri nets that can generate a sequence diagram understanding the behavior of an object-oriented system using reverse engineering. In this paper [3] researchers represent a novel method for extracting UML2 state machine diagrams from object-oriented java source code using Nested choice patterns. Also in [4] a strategy is shown to investigate creative approaches to instruction and evaluation that encourage students to learn programming effectively, according to the suggested technique, when a user inserts source code into a visualization tool, it will be transformed into a class diagram and a sequence diagram based on what the user specifies. Also, an algorithm was discussed that creates suitable class diagrams and sequence diagrams for user-entered object-oriented language source code. To accomplish the reverse engineering aim of converting source code into design documents.

In [5] the research's goal was to create a tool that can automatically distinguish between UML class diagrams and other types of class diagrams. Earlier studies classified class diagrams using machine learning techniques. As a result, they must recognize picture attributes and look into how these aspects affect the categorization issue in the UML class diagrams.

The authors in [6], suggest an automatic classifier. They demonstrate how supervised machine learning methods were used to create such a classifier. They examine the elements that help categorize FwCD and RECD throughout its building. After examining other machine-learning techniques, they discover that the Random Forest approach is the best appropriate algorithm for our objective.

Also in [7], the research's goal was to create a tool that can automatically identify between UML class diagrams and other types of diagrams. Earlier they classified class diagrams using machine learning techniques. As a result, they recognize image attributes and look into how these aspects affect the categorization issue in the UML class diagrams. In the area of deep learning, they created a novel method for automatically identifying class diagrams. This method uses convolutional neural networks.

In [8]. The authors proposed representing a code snippet as a set of paths in its AST. They have used the AST paths with Conditional Random Fields (CRF) and evaluated the approach on method naming.

Our research aims to fill the gap between manual class diagram creation and the need for automated, efficient, and accurate approaches. It addresses the challenges of time and energy-consuming manual efforts, human errors, and inconsistencies in manual class diagrams. Automating the process using GNNs, offers a more reliable and consistent solution [5].

The value of this research lies in its practical applicability and the benefits it offers to the software engineering community.

The article is organized as follows, Section 2 describes our methodology, in Section 3 we display a case study that illustrates our methodology, the results and discussion are presented in Section 4, and finally, the conclusion is shown in Section 5.

2 Methodology

MDA is a recently widely used technique for developing software that is based on modeling and transformations. Modeling the software system is the primary activity that drives the MDA software creation process. In MDA, models and transformations serve as

the main resources. A set of rules referred to as transformation rules are used to transform models from source to target. [9]
Reverse engineering is the act of examining a system to determine its components and how they interact, then modeling those parts to represent the system in a different way or at a higher degree of abstraction.

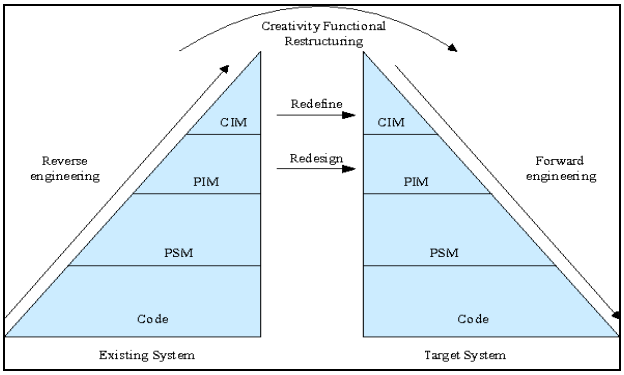

Fig. 1. Reverse engineering in MDA's approach.

The forward engineering transformation method and the reverse engineering transformation approach are depicted in Figure 1. The illustration depicts software transformations that immediately shift objects on the left to objects on the right in the directions where they are located. Model transformation involves the procedure that converts one model of a system into another [10].
Reverse engineering is required when learning about a software system would be time-consuming owing to incomplete, outdated documentation, the complexity of the system, and the maintainer's lack of expertise, which is frequently used to comprehend the structural design and maintain obsolete systems. Reverse engineering is used to find missing data, enhance or offer documentation, find negative effects, reuse parts, and lessen maintenance requirements. [11]
Graph neural networks (GNNs) are an effective method for solving a variety of NLP issues. GNNs have been used to complete tasks including relation extraction, user geographical location, semantic machine translation, and text categorization. GNNs have recently been used for question-answering as well. [12]
Our methodology is shown in Figure 2. We used reverse engineering, starting from the source code and respecting the MDA principles and machine learning algorithm to obtain the class diagram at the PIM level. By utilizing the code's graph representation, Graph Neural Networks (GNNs) can assist with generating a class diagram from the source code.

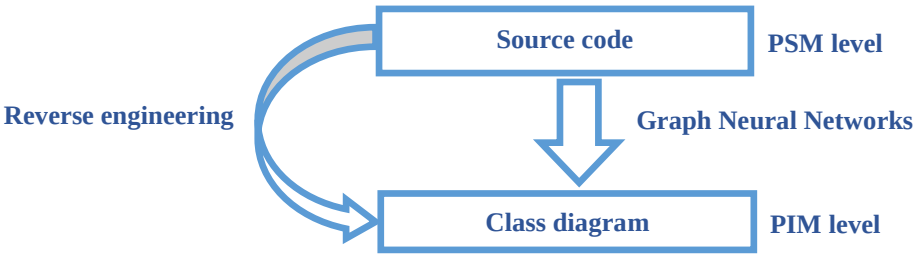

Fig. 2. The proposed methodology.

3 Case study

Suppose a software development company is engaged in a significant project with a complicated codebase. The project's insufficient documentation makes it difficult for developers to understand the relationships between the various pieces of code. Due to the longer development cycles, debugging efforts, and knowledge transfer, there is an increase in energy consumption. [13]
Energy-saving advantages come from using reverse engineering in MDA and applying GNNs to automatically create a class diagram from the source code. GNNs are trained on annotated datasets, learning to understand the source code's structure, relationships, and context. [14]
We use a labeled dataset consisting of pairs of source code graphs and the related class diagrams to train a GNN model. The GNN gains the ability to understand complex class relationships and interactions
We utilize the trained GNN model to produce class diagrams for the targeted codebase. The software's structure, including the class hierarchy, attributes, methods, and connections, is represented visually by the class diagrams.
In this case study, we focus on school management developed in Java. Our objective is to automatically generate a class diagram for the application using GNNs. The study aims to evaluate the effectiveness and accuracy of the GNN-based approach in capturing the system's structure and relationships.

- Data collection: the code base consists of Java classes, packages, and associated dependencies, which form the basis for our analysis.
- Source code analysis: we thoroughly analyze the application source code to gain insights into its architecture and design. We identify classes, interfaces, attributes, methods, and their relationships. We also examine the use of inheritance, composition, and dependency patterns within the system.
- Graph construction: Based on the source code analysis, we construct a graph representation of the system. Each class is represented as a node, and the relationships between classes, such as inheritance, associations, and dependencies, are captured as edges in the graph.
- Feature extraction: we extract relevant features from the graph representation. Node features include class names, attribute types, and method signatures. Edge features represent the type of relationship between classes, such as association, inheritance, or dependency.
- Training data preparation: The manually created class diagram is used as the ground truth for the labeled dataset. We prepare the training data by pairing the graph representation of the application source code with the corresponding manual class diagram.
- GNN Model Training: We train a GNN model using the labeled dataset. The GNN model learns to predict class diagrams based on the input graph structure and features. For this task, we employ graph neural network architectures such as Graph Convolutional Networks (GCNs) or Graph Attention Networks (GATs).
- Class diagram generation: based on the output of the GNN model, a class diagram is generated automatically. The node representations obtained from the GNN model correspond to the classes in the diagram, and the edges represent the relationships between the classes, such as inheritance, associations, and dependencies. The generated class diagram provides a visual representation of the system's structure.

4 Results and discussion

By employing reverse engineering in MDA's approach and utilizing GNNs for class diagram generation, the software development company achieves improved energy efficiency by streamlining the understanding of the codebase, reducing development time, optimizing debugging efforts, facilitating knowledge transfer, and promoting efficient software maintenance practices [15].

The comparison between the automated approach and manual class diagrams shown in Table 1, revealed several interesting insights. Overall, we found that the automated approach was able to capture a significant portion of the system's structure, associations, and relationships accurately. The generated class diagrams closely resembled the manually created ones in terms of class hierarchy, attribute associations, and method relationships [16].

Our automated approach showcased several practical benefits. The most significant advantage was the substantial time savings it offered compared to manual class diagram creation. Manually creating class diagrams for a system of the size and complexity, would typically require weeks of effort from domain experts. In contrast, the automated approach using GNNs was able to generate the class diagrams in a matter of hours. This time savings can significantly expedite the software understanding and documentation process, allowing developers to focus on other critical tasks [17].

Furthermore, the automated approach mitigated the risk of human error associated with the creation of manual class diagrams. Manually creating class diagrams requires careful attention to detail, and mistakes can easily occur, especially when dealing with complex codebases. Automating the process reduces the chances of inaccuracies and inconsistencies in the generated class diagrams, ensuring a more reliable and consistent representation of the software system. [18-21].

Table 1. Comparison between automatic and manual approach.

Aspect	Automated Approach using GNNs	Manual Class Diagram Creation
Accuracy	High accuracy in capturing the system's structure, associations, and relationships. Some discrepancies may exist, but overall, provides a reliable representation.	Relies on the expertise of domain experts, but human errors can occur. Inconsistencies in capturing the system's structure, associations, or relationships may arise.
Time Required	Significantly reduces the time required for class diagram generation. Generates class diagrams in a matter of hours.	A time-consuming process that can take weeks or longer depending on the complexity of the system.
Effort Required	Drastically reduces the effort needed for class diagram generation. Automated process requires minimal manual intervention.	Requires significant effort from domain experts who manually analyze the source code and construct the class diagrams.
Risk of Human Error	Reduces the risk of human error associated with manual class diagram creation.	Human errors can occur during the manual process, impacting the accuracy and consistency of the class diagrams.
Consistency and Reliability	Provides a consistent and reliable representation of the software system.	Consistency may vary depending on the expertise and interpretations of different domain experts involved in the manual process.
Applicability	Applicable to various software systems and programming languages. Enables reverse engineering, system comprehension, and architectural analysis.	Applicable to any software system, but the efficiency and accuracy depend on the expertise and effort of domain experts.

5 Conclusion

There are many benefits of using reverse engineering and GNNs for class diagram generation in the software project like reduced development time, with an automatically generated class diagram, developers can quickly grasp the code base's structure, reducing the time and energy spent on understanding the system's intricacies. Also efficient debugging, and maintenance, the class diagram provides a visual aid for developers to locate and troubleshoot issues, minimizing debugging efforts and reducing energy consumption associated with resolving software bugs, and facilitating knowledge transfer. The generated class diagram serves as a valuable documentation artifact, aiding in knowledge transfer among team members and reducing the energy required for on boarding and collaboration.

Finally, improved code understanding and the visual representation of the source code structure enhance developers' comprehension, enabling them to make informed design decisions and reducing the energy spent on rework and refactoring. [22]

In conclusion, this study proposes a novel method for automatically producing class diagrams from source code using Graph Neural Networks. The suggested method effectively substitutes time-energy-consuming and error-prone manual class diagram construction work by utilizing AI and reverse engineering approaches.

To increase developers' efficiency and deepen their understanding of software systems, the article intends to encourage higher investigation into and usage of automated approaches in software engineering.

References

[1] Mukhtar, M. I., & Galadanci, B. S. (2018). Automatic code generation from UML diagrams: the state-of-the-art. Science World Journal, 13(4), 47-60.

[2] Baidada, C., El Mahi, B., & Jakimi, A. Towards the reverse engineering of UML sequence diagrams for multithreaded java software.

[3] Aabidi, M. H., El Mahi, B., Baidada, C., Jakimi, A., & Ammar, H. (2017). Benefits of reverse engineering technologies in software development makerspace. In *ITM Web of Conferences* (Vol. 13, p. 01028). EDP Sciences.

[4] Singh, K. (2020). Transformation of source code into UML diagrams through visualization tool. *International Journal of Advanced Science and Technology*, *29*(8), 4861-1114.

[5] Gosala, B.; Chowdhuri, S.R.; Singh, J.; Gupta, M.; Mishra, A. Automatic, Classification of UML Class Diagrams Using Deep Learning Technique:

Convolutional Neural Network. *Appl. Sci.* 2021, *11*, 4267.

[6] Osman, M. H., Ho-Quang, T., & Chaudron, M. (2018, August). An automated approach for classifying reverse-engineered and forward-engineered UML class diagrams. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 396-399). IEEE.

[7] Mangaroliya, K., & Patel, H. (2020). Classification of reverse-engineered class diagram and forward-engineered class diagram using machine learning. *arXiv preprint arXiv:2011.07313*.

[8] Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2018). Code2vec: learning distributed representations of code. CoRR. *arXiv preprint arXiv:1803.09473*.

[9] Jing, D., Yang, H., & Hakeem, H. (2014, September). Using abstraction in

MDA-based reverse engineering for creative evolution. In *2014 20th International Conference on Automation and Computing* (pp. 67-72). IEEE.

[10] Sabir, U., Azam, F., & Anwar, M. W. (2017, December). A comprehensive investigation of model-driven architecture (MDA) for reverse engineering In *Proceedings of the 2017 International Conference on Software and e Business* (pp. 43-48).

[11] Aabidi, M. H., El Mahi, B., Baidada, C., Jakimi, A., & Ammar, H. (2017). Benefits of reverse engineering technologies in software development makerspace. In *ITM Web of Conferences* (Vol. 13, p. 01028). EDP Sciences.

[12] https://www.simplilearn.com/what-is-graph-neural-network-article

[13] Kehagias, D., Jankovic, M., Siavvas, M., & Gelenbe, E. (2021). Investigating the interaction between energy consumption, quality of service, reliability, security, and maintainability of computer systems and networks. *SN Computer Science*, *2*(1), 23.

[14] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI open*, *1*, 57- 81.

[15] Wu, Z., Pan, S., Long, G., Jiang, J., & Zhang, C. (2019). Graph wavenet for deep spatial-temporal graph modeling. *arXiv preprint arXiv:1906.00121*.

[16] ANDRE, P., GUERIN, A., ROZEN, A., & GICQUEL, A. Raffinement de protocoles de communication par transformation de modèle.

[17] Bergström, G., Hujainah, F., Ho-Quang, T., Jolak, R., Rukmono, S. A., Nurwidyantoro, A., & Chaudron, M. R. (2022). Evaluating the layout quality of UML class diagrams using machine learning. *Journal of Systems and Software*, *192*, 111413.

[18] Stikkolorum, D. R., van der Putten, P., Sperandio, C., & Chaudron, M. (2019). Towards Automated Grading of UML Class Diagrams with Machine Learning. *BNAIC/BENELEARN*, *2491*.

[19] Ciccozzi, F., Malavolta, I., & Selic, B. (2019). Execution of UML models: a systematic review of research and practice. *Software & Systems Modeling*, *18*, 2313-2360.

[20] Abdelnabi, E. A., Maatuk, A. M., & Hagal, M. (2021, May). Generating uml class diagram from natural language requirements: A survey of approaches and techniques. In *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA* (pp. 288-293). IEEE.

[21] Abdelnabi, E. A., Maatuk, A. M., Abdelaziz, T. M., & Elakeili, S. M. (2020, December). Generating UML class diagram using NLP techniques and heuristic rules. In *2020 20th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)* (pp. 277-282). IEEE.

[22] Ciancarini, P., Ergasheva, S., Kholmatova, Z., Kruglov, A., Succi, G., Vasquez, X., & Zuev, E. (2020). Analysis of energy consumption of software development process entities. *Electronics*, *9*(10), 1678.