



Enhancing Model-Driven Reverse Engineering Using Machine Learning

Hanan Abdulwahab Siala

hanan.siala@kcl.ac.uk

King's College London

London, UK

ABSTRACT

Organizations often rely on large applications that are classified as legacy systems due to their dependence on outdated programming languages or platforms. To modernize these systems, it is necessary to understand their architecture, functionality, and business rules. Our research aims to define a novel model-driven reverse engineering (MDRE) approach to extract Unified Modeling Language (UML) and Object Constraint Language (OCL) representations from source code using Large Language Models (LLMs).

KEYWORDS

Application programs, Model Driven Reverse Engineering (MDRE), Unified Modeling Language (UML), Object Constraint Language (OCL), Machine Learning, Large language models (LLMs), Program comprehension.

ACM Reference Format:

Hanan Abdulwahab Siala. 2024. Enhancing Model-Driven Reverse Engineering Using Machine Learning. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal*. ACM, New York, NY, USA, Article 111, 3 pages. <https://doi.org/10.1145/3639478.3639797>

1 RESEARCH PROBLEM AND MOTIVATION

Legacy systems are essential for daily organizational operations, but they need to be modernized to meet the changing requirements of stakeholders. As legacy systems become increasingly complex and more difficult to maintain, there is a growing need for new and better ways to understand and maintain them. Reverse engineering plays a crucial role in achieving this goal by analyzing a subject system to identify the system's components and their inter-relationships and create representations of the system at a higher level of abstraction [2]. Traditional reverse engineering starts by analyzing program source code to extract program specifications [21]. However, another approach is to use a model-driven perspective following OMG standards such as KDM [4], based on the Model-Driven Architecture (MDA) [16]. In MDA, all artifacts are represented as models using modeling languages. These models are organized into a hierarchy of abstraction levels, and models at

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0502-1/24/04

<https://doi.org/10.1145/3639478.3639797>

different levels can be automatically transformed into each other.

Reverse engineering techniques for software systems have been extensively studied over the past 20 years, and numerous approaches, including the model-driven approach (MDRE) —the subject of our study— have been introduced. Raibulet et al. [19] presented a comprehensive analysis of MDRE approaches in the literature, up to 2017. Through a systematic literature review (SLR), the authors highlight 15 MDRE approaches and discuss their various features and objectives. We also conducted an SLR of 938 relevant publications in this domain from 2000 to 2023 to gain a comprehensive understanding of current research in MDRE approaches. We found 64 distinct MDRE approaches, of which 29 focus on enhancing program understanding and documentation. The majority of the identified MDRE approaches extract information using static techniques, which involve parsing the source code and extracting an abstract syntax tree. Others, however, employ dynamic techniques, where information is extracted from the system during runtime or a combination of both static and dynamic techniques (hybrid techniques). With regard to the representations of these approaches, UML diagrams [15] are the most widely targeted, with class diagrams being the top choice (14 approaches), followed by sequence diagrams (12 approaches). Nine approaches target the Knowledge Discovery Metamodel (KDM) [17]. Only 3 approaches extract Object Constraint Language (OCL) specifications [14]. Regarding models extracted by reverse engineering approaches, Lano et al. [11] argue that UML/OCL representations are a good choice to present system abstractions because they are: 1) Widely used and supported, 2) International standards, 3) Supported by many tools.

Large Language Models (LLMs) are a type of machine learning model trained on massive datasets of text data to achieve powerful knowledge-based capabilities. In 2017, Vaswani et al. [23] introduced the transformer architecture, which has become the dominant architecture for LLMs. Recent advances in LLMs have the potential to revolutionize numerous domains, including software engineering (SE) [8]. Combining traditional MDRE techniques with LLMs offers a promising approach for automatically extracting program specifications from source code, thereby improving the efficiency and effectiveness of the reverse engineering process. Yet, no approaches currently employ LLMs in this area, as evidenced by our SLR. Consequently, there is a need for an MDRE method with tool support for automatically extracting program specifications from source code using LLMs.

2 RESEARCH HYPOTHESIS

Our research hypothesis is that developing an MDRE tool using LLMs is an effective way of generating UML and OCL specifications from source code.

3 PROPOSED APPROACH

We propose an MDRE method with tool support to extract UML and OCL specifications from source code. Our research questions are:

- (1) How can LLMs be utilized to abstract UML/OCL specifications from source code?
- (2) What information should be gathered by applying LLMs to facilitate the abstraction of UML/OCL specifications from source code? and How?

Figure 1 shows the general overview of the proposed toolchain for our method, which shows the following consecutive phases. Pre- and post-processing phases are used to overcome the errors that can be generated by using ML-based techniques for code processing, as suggested by Malyala *et al.* [12].

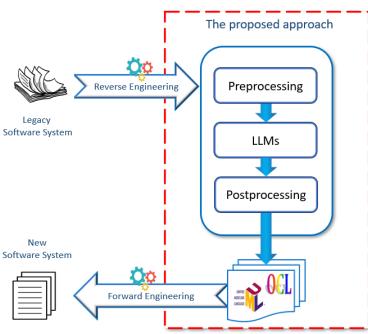


Figure 1: Proposed approach overview.

- (1) **Input/Output:** The input to our approach is Java or Python source code files, and the output consists of UML/OCL specifications. Existing code LLMs have limited model knowledge [3], so the creation of a new LLM or fine-tuning an existing LLM may be necessary.
- (2) **Training:** Appropriate training techniques will be investigated, such as supervised or unsupervised co-training of forward and reverse translations from code to UML/OCL, analogous to the work of [9, 10] for program translation. Appropriate datasets will be selected/synthesised. Data sources can include existing repositories of code and models [6], existing code abstractors [11], by combining code-to-text summarisation with text-to-OCL tools [1] or by using MDE code generators.
- (3) **Pre-processing:** The pre-processing phase can comprise the following: 1) Tokenizing: a text string that represents a software system is broken down into a sequence of tokens. 2) Code simplification: replacing complex or unusual code constructs with simplified equivalents to reduce the difficulty of the ML task.
- (4) **Large Language Models (LLMs):** In this phase, the encoder takes a sequence of words (tokens) and turns them into a numerical representation by using a variety of techniques. The LLM produces a textual UML/OCL intermediate representation.

(5) **Post-processing:** Automatic model repair techniques will be applied to generate more reliable UML/OCL representations. Further transformations may be necessary to construct the desired specifications with enhanced accuracy.

(6) **Documentation/Program Translation:** Once the diagram elements are identified, UML class diagrams and OCL representations can be constructed in several ways. For example, graphical representations can be produced using a diagramming tool like Graphviz [22], PlantUML [18] or Modelio [13] to visualise class diagrams. In addition, after obtaining a UML/OCL specification, the system can be translated to another language by using AgileUML [5].

4 EXPECTED CONTRIBUTIONS

The expected contributions include conducting an SLR to investigate the current state of research in MDRE and model-driven re-engineering. This involves identifying gaps in the current state of research and outlining areas where further work is needed. Furthermore, answering the research questions will contribute to developing an MDRE approach capable of extracting UML/OCL representations from source code to enhance program comprehension and migration. Achieving this proposed approach requires:

- (1) Developing a method and tools for automatically generating graphical representations from source code using LLMs, which includes:
 - (a) Defining appropriate training techniques and selecting appropriate training/validation datasets.
 - (b) Constructing the LLM.
 - (c) Defining appropriate pre- and post-processing techniques.
 - (d) Defining UML and OCL generation techniques to extract UML/OCL specifications from LLM outputs.
- (2) Comparing the results with the results of a state-of-the-art approach and evaluating using two case studies.

5 METHODOLOGY

To develop the proposed approach for MDRE, we will employ the Design Science Methodology (DSM) [7]. According to DSM, we will rigorously implement both the building and evaluation of the developed artifacts (i.e., tools, methods, models, constructs, or instantiations), following the seven guidelines presented by Hevner *et al.*. We will iterate through a build-and-evaluate loop multiple times to generate the final artifact.

6 EVALUATION METHODOLOGY

The approach will be evaluated and validated by: 1) selecting an existing state-of-the-art MDRE approach and demonstrating how ML techniques can outperform this approach for the same inputs. 2) two case studies with diverse characteristics, sizes, and complexities will be used to measure the effectiveness of the proposed artifact. One case is likely to be in the domain of ML systems [20].

Evaluation criteria for reverse engineering include (i) semantic correctness and completeness of the models compared to the code; (ii) the quality and comprehensibility of the diagrams. We will comprehensively evaluate the tool's performance and ensure its applicability across a wide range of real-world scenarios.

REFERENCES

- [1] Seif Abukhalaf, Mohammad Hamdaqa, and Foutse Khomh. 2023. On Codex prompt engineering for OCL generation: an empirical study. *arXiv:2303.16244v1* (2023), 148–157.
- [2] Elliot J. Chikofsky and James H Cross. 1990. Reverse engineering and design recovery: A taxonomy. *IEEE software* 7, 1 (1990), 13–17.
- [3] Javier Cámarra, Javier Troya, Lola Burgueño, and Antonio Valleccillo. 2023. On the assessment of generative AI in modeling tasks. *SoSyM* 22 (2023), 781–793.
- [4] Gaëtan Deltombe, Olivier Le Goarer, and Franck Barbier. 2012. Bridging KDM and ASTM for Model-driven software modernization. In *SEKE 2012*. Knowledge Systems Institute Graduate School, 517–524.
- [5] Eclipse. 2022. Eclipse AgileUML Project. <https://projects.eclipse.org/projects/modeling.agileuml>.
- [6] Robert France, Jim Bieman, and Betty H. C. Cheng. 2007. Repository for Model Driven Development (ReMoDD). In *Models in Software Engineering*, Thomas Kühne (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 311–317.
- [7] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. 2004. Design science in Information Systems research. *MIS QUARTERLY* 28, 1 (MAR 2004), 75–105.
- [8] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large Language Models for Software Engineering: A Systematic Literature Review. *arXiv:2308.10620 [cs.SE]*
- [9] Prithwish Jana, Piyush Jha, Haoyang Ju, Gautham Kishore, Aryan Mahajan, and Vijay Ganesh. 2023. Attention, compilation, and solver-based symbolic analysis are all you need. *arXiv:2306.06755v1* (2023).
- [10] Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, and Guillaume Lampe. 2020. Unsupervised translation of programming languages. *arXiv:2006.03511v3* (2020).
- [11] Kevin Lano, Howard Haughton, Ziwen Yuan, and Hessa Alfraihi. 2023. Program abstraction and re-engineering: an Agile MDE approach. In *SAM 2023*. ACM, New York, 10 pages.
- [12] Aniketh Malyala, Katelyn Zhou, Baishakhi Ray, and Saikat Chakraborty. 2023. On ML-Based Program Translation: Perils and Promises. *arXiv:2302.10812 [cs.PL]*
- [13] Modelio. 2023. Modelio. <https://www.modelio.org/index.htm>.
- [14] OMG. 2014. Object Constraint Language 2.4 Specification, OMG document formal. <https://www.omg.org/spec/OCL/2.4/About-OCL>.
- [15] OMG. 2017. OMG Systems Modeling Language (UML), Version 2.5.1. <https://www.omg.org/spec/UML>.
- [16] OMG. 2023. Model-Driven Architecture (MDA). <https://www.omg.org/mda>.
- [17] Ricardo Pérez-Castillo, Ignacio García-Rodríguez De Guzman, and Mario Piattini. 2011. Knowledge discovery metamodel ISO/IEC 19506: A standard to modernize legacy systems. *Computer Standards and Interfaces* 33 (2011), 519–532.
- [18] PlantUML. 2023. PlantUML. <https://plantuml.com/>.
- [19] Claudia Raibulet, Francesca Arcelli Fontana, and Marco Zanoni. 2017. Model-driven reverse engineering approaches: A systematic literature review. *Ieee Access* 5 (2017), 14516–14542.
- [20] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. *Advances in neural information processing systems* 28 (2015), 2503 – 2511.
- [21] Harry Sneed. 2010. Migrating from COBOL to Java: A report from the field. In *IEEE Proc. of 26th ICSM*. IEEE Press, 1–7.
- [22] Graphviz Development Team. 2023. Graphviz - Graph Visualization Software. <https://www.graphviz.org/>.
- [23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. *arXiv:1706.03762 [cs.CL]*