Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 236 (2024) 452–459

Procedia Computer Science

www.elsevier.com/locate/procedia

International Symposium on Green Technologies and Applications (ISGTA'2023)

Extraction of UML class diagrams using deep learning: Comparative study and critical analysis

Zakaria Babaalla[a,*], Hamza Abdelmalek[a], Abdeslam Jakimi[a], Mohamed Oualla[a]

[a]*GL-ISI Team Department of Informatics, Faculty of Sciences and Technics, UMI-Meknes,Errachidia 52000, Morocco*

Abstract

Several approaches and tools have been proposed to facilitate the automatic generation of Unified Modeling Language (UML) class diagrams from natural language specifications, based on advances in Natural Language Processing (NLP). However, these tools suffer from difficulties due to the inherent imprecision and ambiguity commonly found in natural language expressions. In this article, we present an overview and a study of approaches and tools designed to extract UML diagrams from textual requirements using NLP and computational linguistics techniques. Furthermore, we introduce approaches employing deep learning techniques. Next, we provide a descriptive study and comparative analysis of the limitations and contributions of these automatic and semi-automatic tools, as well as solutions for improving the existing state of the art.

Keywords: Natural Language Processing; UML; class diagram; machine learning; deep learning

1. Introduction

To fully understand and determine system requirements, the software development lifecycle begins with requirements analysis. This step describes what is expected of the system, and collects the necessary information to understand and define the requirements. It is important to address the whole problem, and not just the individual parts, before starting a software project, otherwise a poor requirements analysis can lead to delays, additional budgets, or project failure [1]. These requirements are expressed in natural language and collected from various sources such as documents, surveys, and interviews. The heterogeneity of requirements sources leads to a series of problems related to ambiguity, lack of clarity, incompleteness, and context dependency. Developing complex software systems using object-oriented (OO) approaches requires the use of robust tools, approaches, and standards during the analysis and design phases [2]. Therefore, the Object Management Group (OMG) adopted the Unified Modeling Language (UML) standard, which is the result of the unification of the most popular OO notations, namely Booch, Object-Oriented Soft-

* Corresponding author. Tel.: +212 652-717-295
E-mail address: za.babaalla@edu.umi.ac.ma

1877-0509
10.1016/j.procs.2024.05.053

ware Engineering (OOSE), and Object Modeling Technique (OMT) [3]. Through extensive standardization efforts by the OMG, UML has emerged as the most adopted standard for the analysis, design, specification, visualization, and development of OO software systems [4]. However, the analysis of requirements expressed in natural languages to extract UML models is a difficult task, requiring automated techniques such as those proposed in [5, 6, 7, 8]. On the other hand, experiments with these tools have raised certain limitations related to their complexity, as these tools require extensive user inputs to complete the process, and to their reliability, as most tools are unable to extract complete UML elements such as operations and relationships. This article presents a descriptive study and comparative analysis showing the limitations and contributions of automatic and semi-automatic tools for extracting UML class diagrams. It also discusses directions to overcome the shortcomings of existing approaches based on linguistic practices, heuristic rules, and domain ontologies. The discussion also attempts to showcase the accuracy of recent systems involving natural language processing (NLP) and machine learning techniques.

This article is organized as follows. Section 2 presents the literature review regarding the extraction of UML diagrams using NLP, machine learning, and deep learning techniques. Section 3 introduces a comparative study between the tools. In Section 4, we briefly explain our proposed approach. Section 5 concludes the article.

2. Related works

Traditionally, the task of extracting UML class diagrams from informal specifications has been addressed through the development of systems that rely on NLP techniques, supported by heuristic rules or domain ontologies. However, advances in automatic learning using deep learning algorithms have motivated researchers to propose more efficient systems for extracting UML diagrams compared to rule-based systems. A vast literature exists concerning this topic, describing a range of approaches based on three different techniques: NLP, machine learning, and deep learning.

2.1. Natural Language Processing techniques

Kumar and Sanyal employed the RUP (Rational Unified Process) [9] iterative method in their work to develop the tool SUGAR (Static UML Model Generator from Analysis of Requirements). This tool is designed to generate static UML diagrams from natural language requirements, based on object-oriented analysis and design use cases [10]. The SUGAR tool initiates the process by generating a use case diagram, which in turn is used by RUP alternatives to associate various fragments of the UML class diagram.

Sharma et al. generated high-level class diagrams from requirements, using a tool that implements their approach called Functional Design Creation Tool (FDCT) [11]. The FDCT consists of three modules, responsible for preprocessing and analyzing natural language requirements to identify model entities, and then assembling these generated units into a coherent class diagram.

Ibrahim and Ahmad proposed a tool named RACE that facilitates the process of requirements analysis to extract class diagrams using NLP techniques [12]. The tool is organized into three components. The first is used to perform syntactic and lexical analysis of requirements, the second extracts and refines diagram elements, and the third ensures proper management of extracted UML concepts. However, the tool can generate incomplete class diagrams lacking attributes.

Deeptimahanti and Sanyal presented a semi-automatic technique called UMGAR, which uses a set of eight syntactic rules to reconstruct natural language requirements provided by users [13]. The UMGAR process involves creating an analysis tree for each requirement, facilitating the extraction of UML elements. Next, it performs a morphological analysis and corrects the ambiguities created by inaccurate requirements. Finally, it uses an object-oriented model builder to provide a variety of UML diagrams. Nevertheless, UMGAR requires human interaction when eliminating irrelevant classes and identifying certain relationships between objects.

The RAPID tool developed by More and Phalnikar, uses a similar architecture as RACE [12], with the only exception that RAPID can generate other types of UML diagrams, such as use cases, sequences, and collaboration [14].

Shinde et al. have proposed a tool consisting of an NLP analysis component responsible for pre-processing requirements by applying a set of NLP algorithms [15]. A second knowledge extraction component uses 12 heuristic rules to extract elements from UML diagrams, encompassing classes and sequences. In addition, the tool generates Java source code from the extracted diagrams.

Herchi and Abdessalem aim to automate the analysis of software requirements documents using NLP to build class diagrams [16]. In this context, they developed an NLP tool called Diagram Class Builder (DC-Builder). The tool relies on the GATE API[1] to process the input scenario. It then uses certain linguistic rules to find UML concepts such as class names, attributes, and associations. This process leads to the generation of an initial XML file, which is subsequently enhanced by incorporating ontologies containing foundational domain knowledge.

The UDRA tool proposed by Amune and Naik demonstrates the use of NLP and domain ontology techniques to reflect natural languages in UML diagrams [17]. The tool can generate UML diagrams such as class diagrams, object diagrams, package diagrams, and deployment diagrams. It generates a complete class diagram including attributes, operations, and relationships such as generalization, association, and composition.

Jaiwai and Sammapun presented an approach that extracts class diagrams from requirements written in the Thai language, employing NLP tools and extraction rules [18]. The process begins with requirements pre-processing, followed by word segmentation. These words are then tagged with parts of speech to identify class nouns and attributes using extraction rules.

Utama and Jang detailed in their article an algorithm for extracting class diagrams from a problem statement freely expressed in natural language [19]. To increase the accuracy of the system, the requirements must first be normalized and then passed to the system's NLP module, whose functionalities are provided by the SpaCy NLP library [20]. Finally, to create the class diagram, the extraction algorithm is created based on a set of 15 linguistic rules.

Abdelnabi et al. described in their article a method for generating class diagrams from informal requirements using NLP techniques and heuristic rules [21]. To facilitate the process of mapping extracted knowledge to UML class models, the approach uses the Stanford CoreNLP library [22] to read and perform the analysis of user requirements. Tests show that the proposed approach has a clear advantage in detecting a variety of elements, such as relationships (aggregations, reflexive relations…) as well as multiplicities and types of participation between classes.

To overcome the limitations and problems faced by previous tools, Bashir et al. implemented a tool called READ to automate the process of analyzing and designing software requirements using NLP techniques [2]. NLTK (Natural Language ToolKit) [23] is used to process requirements written in natural language, followed by a rule base to extract classes, attributes, and operations from the requirements. The results obtained confirm that READ is more accurate than existing automatic and semi-automatic tools.

Alharbia et al. designed a tool that extracts a class diagram from software specifications using NLP techniques efficiently and with fewer errors [24]. The proposed method initially uses the functions of a first text preprocessing module supported by the Open NLP parser to normalize the input requirements of a second module whose objective is to explore the classes, their attributes, and relationships between these classes. The user can extract the class diagram corresponding to the initial requirements, via a user-friendly interface, and enables the generation of C# source code.

2.2. Machine Learning techniques

Narawita developed a web application called UML Generator, which generates both UML use case and class diagrams from informal software specifications in a short time and at low cost [25]. The features of the application's NLP module are provided by the SharpNLP library. To increase the extraction accuracy, the approach uses a set of XML rules to remove noise words from the list of identified entities. Next, a classification model called Weka takes these unfiltered entities and classifies them according to whether they represent instances of a class diagram or a use case diagram. The user then accepts or rejects the generation of use case and class diagrams using the Visual Studio Modeling module.

Yang and Sahraoui proposed an approach for automatic transition from a specification in English to a UML class diagram using language models learned by machine learning algorithms [26]. First, requirements are pre-processed by substituting pronouns to make the sentences less dependent on each other. Then, the SpaCy library [20] and an English model divide the preprocessed text into individual sentences. The next step is to use the Naive Bayes classifier to classify each sentence obtained according to whether it describes a "class" or a "relation". The tf-idf method was chosen to convert sentences into vector representations when training the classifier. Next, analysis and

[1] https://gate.ac.uk/

extraction procedures are employed to generate UML fragments from each labeled sentence. Finally, these fragments are combined to build the associated class diagram.

2.3. Deep Learning techniques

Arachchi proposed an approach for generating UML use case and class diagrams from functional requirements, by combining NLP, machine learning, and deep learning techniques [27]. The proposed approach is similar to [26] because it takes as input the requirements in natural language. Then, the NLP module applies these functions to extract word fragments and named entities. Afterward, the Naive Bayes algorithm and a recurrent neural network (RNN) are learned and implemented in the ML module of the tool, to identify the use cases, actors, classes, and associations.

Saini et al. have proposed a tool called DoMoBOT which combines all the possibilities of NLP with the power of deep learning to automate the process of generating domain models represented by class diagrams from a textual description in natural language [28]. It promotes user interaction and ensures traceability between the various artifacts produced during model development. The first component of DoMoBOT applies the functionality of the SpaCy library [20] to pre-process the problem description. A second component called descriptive, captures only relevant domain concepts (classes, attributes, ...) by applying certain NLP techniques based on extraction rules. The third component implements a pre-trained predictive model based on machine learning and deep learning techniques to improve the accuracy of participating concepts and their relationships. The fourth component uses two algorithms to combine the results obtained by NLP and machine learning techniques, to build a final domain model.

Rigou and Khriss used deep learning techniques developed in the context of NLP, as the core extraction process of the Platform Independent Model (PIM) [29] from requirements written in natural language [30]. The proposed model is learned by discovering the business concepts that construct the UML class diagram, and the relationships that exist between these concepts. To accomplish this task, it is necessary to go through a first common encoding layer in the form of a bi-LSTM RNN neural network, BERT model [31], or GPT-2 model [32], which produces a vector representation of each token of the sentence, including understanding the context in which it appears. Then, a second Feed Forward Neural Network (FFNN) layer specific to the first entity detection task is trained based on these contextual vectors to determine whether a token will be part of an interesting mention to build the diagram or whether it will be a useless binding token. However, the two remaining tasks concern tokens that are likely to be useful, and relate to the classification of a pair of mentions composed of one or more tokens, rather than performing individual token classification. The next layer of the model is learned to perform two distinct tasks. The first is to find the various references to the same entity in the requirements, to avoid any possible redundancy of certain entities in the diagram. The second is to identify the relationships and their types that exist between the entities.

The results obtained in [27, 28, 30] are encouraging and show that by increasing the size of the dataset used in the model training, the produced diagrams will present users' requirements with higher fidelity.

3. Comparative analysis

In the previous section, we saw that the methods for designing systems capable of understanding software requirements and translating them into UML diagrams are not recent, and their results differ depending on the techniques involved in each architecture.

Table 1 presents a comparative study of the available approaches and their respective limitations, which motivate research in this direction.

Table 1. Comparison between the performance of the approaches.

Approach	Generated diagrams	Internal System Components	Used Technics	Relationships detected	User interface	XML support	Code generation	Medium precision	Medium recall
[10]	Use case Class	NLP analysis layer + Use case identification layer + Class diagram element extraction layer	NLP (StanfordNLP) + Heuristic rules + RUP (Rational Unified Process)	Association + Aggregation + Inheritance	Available	No	No	———	———
[11]	Class	RAT (Requirements Analysis Tool) + Heuristic module + UML model generator module	Heuristic rules + Domain ontologies	Generalization + Association	Not available	No	No	———	———
[12]	Class	NLP concept extraction module + Class extraction module	NLP (OpenNLP) + Heuristic rules + Domain ontologies	Generalization + Association + Aggregation	Available	No	No	———	———
[13]	Use case Collaboration Class	NLP processing layer + Use Case Extraction Layer + Extraction layer of class and collaboration diagrams	NLP (Stanford CoreNLP) + Heuristic rules + RUP (Rational Unified Process)	Generalization + Association	Not available	Yes	Java	———	———
[14]	Class Sequence Use case Collaboration	NLP concept extraction module + Class extraction module	NLP (OpenNLP) + Heuristic rules + Domain ontologies	Generalization + Association + Aggregation	Available	No	No	———	———
[15]	Class Sequence	NLP analysis block + Diagram & code generation block	NLP (OpenNLP) + Heuristic rules	Composition + Association + Aggregation	Available	No	Java	———	———
[16]	Class	NLP analysis block + Block for extracting UML concepts + refinement block	NLP (ANNIE) + Heuristic rules + Domain ontologies	Generalization + Association + Aggregation	Not available	Yes	No	93 %	83 %
[17]	Class Object package deployment	NLP analysis block + Block for extracting UML concepts	NLP (OpenNLP) + Heuristic rules + Domain ontologies	Generalization + Association + Aggregation + Dependency	Not available	No	No	90 %	90 %
[18]	Class	NLP Block + UML concept extractor	NLP (NLTK+NAIST+ORCHID) + Heuristic rules	———	Not available	No	No	81 %	84 %
[19]	Class	NLP module + UML extraction module	NLP (SpaCy) + Heuristic rules	———	Available	No	No	92 %	100 %
[21]	Class	Normalization module + NLP module + Concept extraction module + Refinement module	NLP (Stanford CoreNLP) + Heuristic rules	Generalization + Association + Aggregation + Dependency + Recursion + Multiplicities	Not available	No	No	———	———
[2]	Class	NLP module + Knowledge Extractors + Refinement module	NLP (NLTK) + Heuristic rules + Domain ontologies	Association	Available	No	No	69.56 %	94 %
[24]	Class	NLP module + UML extraction module	NLP (OpenNLP) + Heuristic rules	Generalization + Association	Available	No	C#	91.97 %	88.66 %
[25]	Use case Class	NLP module + XML refinement module + Weka Module (Classifier) + Visualization module	NLP (SharpNLP) + Machine Learning (SMO Algorithm)	Generalization + Association + Aggregation + Composition	Available	No	No	———	———
[26]	Class	NLP module + Binary classifier (Bernoulli Naive Bayes)	NLP (SpaCy) + machine-learning	Generalization + Association + Aggregation	Not available	No	No	17.1 %	25 %
[27]	Use case Class	NLP module + Machine Learning module + Modeling module (Plant UML)	NLP (Stanford CoreNLP) + Machine Learning (Naive Bayes) + Deep Learning (RNN)	Association	Available	No	No	64 %	70 %
[28]	Class	Pretreatment component + Descriptive component + Predictive component + Analysis and decision-making component + Communication component + Recommendation component + Tracing component + Query Response Component	NLP (SpaCy) + machine-learning + Deep Learning (RNN bi-LSTM)	Generalization + Association + Cardinalities	Available	No	No	94.33 %	84 %
[30]	Class	Embedding layer + Feature detection layer + Coreference resolution layer & relationship classification	Deep learning: RNN bi-LSTM – BERT – GPT – FFNN	Association	Not available	No	No	70.73 %	76.7 %

This study clearly shows that all these methods regardless of the used techniques, have several drawbacks, as shown in Table 2.

Table 2. Limitations of the approaches.

Approach	Used Technics	Limitations
[2, 10-19, 21, 24-26]	NLP	• Language dependency. • Restrictions imposed on the initial requirements before the extraction process begins. • Difficulty in terms of maintenance. • Need for user intervention to complete the extraction process. • Incomplete extraction of diagram elements,(relationships such as aggregations, compositions...) • Inability to operate on large requirements documents.
[25-28, 30]	Machine Learning Deep Learning	• Underfitting caused by insufficient data provided to the models during training. These data must be provided in large numbers so that the models can properly learn to capture new data that does not exist in the training set

4. Proposed approach

To overcome these limitations, we have proposed an approach based on the advances in deep learning in the field of NLP. Nowadays, we find powerful language models offered to the scientific community in the form of open-source projects. Our first task was to find out which types of NLP tasks were appropriate for our project. The study of the state of the art provided us with an initial three-level scheme, as presented in Figure 1.

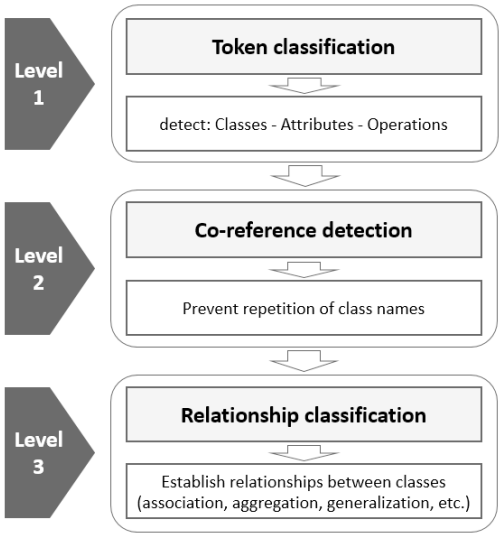

Fig. 1. Our proposed approach.

Level 1 uses a token classifier based on powerful linguistic models (LLaMA, GPT-2 [32], BERT [31], or others) pre-trained to obtain vector representations of tokens (words or other signs), followed by a Bi-LSTM layer or a multi-layer perceptron to provide contextual knowledge. Next, a decoder equipped with a single softmax layer is applied to predict the probabilities of a word belonging to one of the following classes: “Class”, “Attribute”, “Method”, or “Other” if the word contains no relevant information.

Level 2 inspired by the co-reference resolution task, to establish the links between multiple references of the same entity by involving other RNN layers in our model. Concretely, this sub-task aims to eliminate redundant class names in the extracted diagram.

Level 3 implements a relationship classifier by adding additional RNN layers to the model, to identify relationships between the different classes in the diagram (association, aggregation, composition, etc.) and links between each class and its elements (attributes and operations).

To train this model, we need to prepare three different datasets, each associated with one of the sub-tasks provided by the three levels of the model.

5. Conclusion

This article presents approaches for analyzing requirements expressed in natural languages and extracting the concepts used to create UML diagrams. The results differ from one approach to another and confirm that these tools are still immature and cannot be fully exploited in real applications. Indeed, to understand text, artificial intelligence techniques are still less effective than human intelligence at understanding text, particularly for high-level NLP tasks [33]. To overcome these limitations, we have proposed an approach that aims to:

- Create a system based on deep learning, in particular on recent language models that can take initial requirements and translate them as faithfully as possible into UML fragments.
- Detect the various relationships that exist between these extracted UML fragments such as aggregation, composition, and dependency.
- Provide the requirements to the system without the need for additional NLP pre-processing.
- Map requirements to a correct and complete class diagram to make the tool usable even by novice designers, while minimizing user intervention to complete the class diagram generation process.
- Improve performance over previous approaches by providing a dense, rich, and compact dataset enabling the system to capture the relationships between its inputs and outputs when training, and to achieve good scores when generalizing to new requirements.

References

[1] M. Bilal, A. Gani, M. Liaqat, N. Bashir, and N. Malik, "Risk assessment across life cycle phases for small and medium software projects," Journal of Engineering Science and Technology, vol. 15, no. 1, pp. 572-588, 2020.
[2] N. Bashir, M. Bilal, M. Liaqat, M. Marjani, N. Malik, and M. Ali, "Modeling class diagram using nlp in object-oriented designing," in 2021 National Computing Colleges Conference (NCCC), 2021: IEEE, pp. 1-6.
[3] G. Booch, I. Jacobson, and J. Rumbaugh, "The unified modeling language," Unix Review, vol. 14, no. 13, p. 5, 1996.
[4] P. Fitsilis, V. C. Gerogiannis, and L. Anthopoulos, "Role of unified modelling language in software development in Greece–results from an exploratory study," IET software, vol. 8, no. 4, pp. 143-153, 2014.
[5] M.-C. De Marneffe and C. D. Manning, "The Stanford typed dependencies representation," in Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation, 2008, pp. 1-8.
[6] D. K. Deeptimahanti and R. Sanyal, "An innovative approach for generating static UML models from natural language requirements," in Advances in Software Engineering: International Conference, ASEA 2008, and Its Special Sessions, Sanya, Hainan Island, China, December 13-15, 2008. Revised Selected Papers, 2009: Springer, pp. 147-163.
[7] T. M. Abdelaziz, A. M. Maatuk, and F. Rajab, "An approach to improvement the usability in software products," International Journal of Software Engineering & Applications (IJSEA), vol. 7, no. 2, pp. 11-18, 2016.
[8] O. S. Dawood, "From requirements engineering to uml using natural language processing–survey study," European Journal of Industrial Engineering, vol. 2, no. 1, pp. pp. 44-50, 2017.
[9] P. Kruchten, The rational unified process: an introduction. Addison-Wesley Professional, 2004.
[10] D. D. Kumar and R. Sanyal, "Static UML model generator from analysis of requirements (SUGAR)," in 2008 Advanced Software Engineering and Its Applications, 2008: IEEE, pp. 77-84.
[11] V. S. Sharma, S. Sarkar, K. Verma, A. Panayappan, and A. Kass, "Extracting high-level functional design from software requirements," in 2009 16th Asia-Pacific Software Engineering Conference, 2009: IEEE, pp. 35-42.
[12] M. Ibrahim and R. Ahmad, "Class diagram extraction from textual requirements using natural language processing (NLP) techniques," in 2010 Second International Conference on Computer Research and Development, 2010: IEEE, pp. 200-204.
[13] D. K. Deeptimahanti and R. Sanyal, "Semi-automatic generation of UML models from natural language requirements," in Proceedings of the 4th India Software Engineering Conference, 2011, pp. 165-174.
[14] P. More and R. Phalnikar, "Generating UML diagrams from natural language specifications," International Journal of Applied Information Systems, vol. 1, no. 8, pp. 19-23, 2012.
[15] S. K. Shinde, V. Bhojane, and P. Mahajan, "Nlp based object oriented analysis and design from requirement specification," International Journal of Computer Applications, vol. 47, no. 21, 2012.

[16] H. Herchi and W. B. Abdessalem, "From user requirements to UML class diagram," arXiv preprint arXiv:1211.0713, 2012.
[17] AmrutaAmune and R. Naik, "Reflecting Natural Language Text in to UML Diagrams," 2016.
[18] M. Jaiwai and U. Sammapun, "Extracting UML class diagrams from software requirements in Thai using NLP," in 2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2017: IEEE, pp. 1-5.
[19] A. Z. Utama and D.-S. Jang, "An Automatic Construction for Class Diagram from Problem Statement using Natural Language Processing," 멀티미디어학회논문지, vol. 22, no. 3, pp. 386-394, 2019.
[20] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spaCy: Industrial-strength natural language processing in python," 2020.
[21] E. A. Abdelnabi, A. M. Maatuk, T. M. Abdelaziz, and S. M. Elakeili, "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), 2020: IEEE, pp. 277-282.
[22] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations, 2014, pp. 55-60.
[23] E. Loper and S. Bird, "Nltk: The natural language toolkit," arXiv preprint cs/0205028, 2002.
[24] F. Alharbia, S. R. Masadeh, and F. Alshrouf, "A Framework for the Generation of Class Diagram from Text Requirements using Natural language Processing," International Journal, vol. 10, no. 1, 2021.
[25] C. R. Narawita, "UML generator-use case and class diagram generation from text requirements," The International Journal on Advances in ICT for Emerging Regions, vol. 10, no. 1, 2017.
[26] S. Yang and H. Sahraoui, "Towards automatically extracting UML class diagrams from natural language specifications," in Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, 2022, pp. 396-403.
[27] K. D. Arachchi, "AI Based UML Diagrams Generator," 2022.
[28] R. Saini, G. Mussbacher, J. L. Guo, and J. Kienzle, "Automated, interactive, and traceable domain modelling empowered by artificial intelligence," Software and Systems Modeling, pp. 1-31, 2022.
[29] J. Miller and J. Mukerji, "MDA guide version 1.0. 1, Object management group," Inc., June, 2003.
[30] Y. Rigou and I. Khriss, "A Deep Learning Approach to UML Class Diagrams Discovery from Textual Specifications of Software Systems," in Proceedings of SAI Intelligent Systems Conference, 2022: Springer, pp. 706-725.
[31] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2018.
[32] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," OpenAI blog, vol. 1, no. 8, p. 9, 2019.
[33] P. Lemberger and F.-R. Chaumartin, Le traitement automatique des langues: comprendre les textes grâce à l'intelligence artificielle. Dunod, 2020.