Generating UML Class Diagram from Natural Language Requirements: A Survey of Approaches and Techniques

Conference Paper · May 2021

DOI: 10.1109/MI-STA52233.2021.9464433

3 authors:

Esra A. Abdelnabi
University of Benghazi
6 PUBLICATIONS **76** CITATIONS
SEE PROFILE

Abdelsalam M. Maatuk
University of Benghazi
94 PUBLICATIONS **1,569** CITATIONS
SEE PROFILE

Mohamed Hagal
Faculty of information technology, benghazi university
18 PUBLICATIONS **55** CITATIONS
SEE PROFILE

Generating UML Class Diagram from Natural Language Requirements: A Survey of Approaches and Techniques

Esra A. Abdelnabi
Faculty of Information Technology
University of Benghazi, Libya
esra.ali@uob.edu.ly

Abdelsalam M. Maatuk
Faculty of Information Technology
University of Benghazi, Libya
abdelsalam.maatuk@uob.edu.ly

Mohammed Hagal
Faculty of Information Technology
University of Benghazi, Libya
mohamed.hagal@uob.edu.ly

***Abstract*— In the last years, many methods and tools for generating Unified Modeling Language (UML) class diagrams from natural language (NL) software requirements. These methods and tools deal with the transformation of NL textual requirements to UML diagrams. The transformation process involves analyzing NL requirements and extracting relevant information from the text to generate UML class models. This paper aims to survey the existing works of transforming textual requirements into UML class models to indicate their strengths and limitations. The paper provides a comprehensive explanation and evaluation of the existing approaches and tools. The automation degree, efficiency, and completeness, as well as the used techniques, are studied and analyzed. The study demonstrated the necessity of automating the process, in addition to combining artificial intelligence with engineering requirements and using Natural Language Processing (NLP) techniques to extract class diagrams from NL requirements.**

Keywords- System Development, Requirement Engineering, NLP, UML class diagrams.

I. INTRODUCTION

Natural languages are usually used to capture software requirements, and then the analysts analyze and generate the UML diagrams such as class, use case diagrams, etc. for system modeling [1]. Moreover, requirements explained in natural languages can be often complex, ambiguous, uncertain, incomplete, inconsistent, and incoherent. Moreover, the faults that occurred in the earlier phases can be very costly to fix in the software development process next phases. Therefore, it is better to handle these faults earlier and at a lower cost. As a result, analyzing requirements and generating Unified Modeling Language (UML) diagrams is a difficult process, which needs automated or semi-automated support [2].

Recent developments of software engineering depend on object-oriented analysis and design (OOAD) using UML for software requirements modeling, software development, and redevelopment. UML class model is the core for OOAD, where other models are resulting from [3]. Recently, UML Class Diagrams are one of the most useful tools for describing a comprehensive understanding of requirements [4].

In recent decades, several studies proposed automatic and semi-automatic tools to investigate requirements to generate class diagrams. However, the earlier studies have only highlighted the NL requirements analysis and are reliant on user involvement. On the other hand, the recent studies were highlighted both the NL analysis and extraction of UML diagrams from NL text using different techniques such as NLP, mapping rules, patterns, and domain ontology, and linguistic are also used [4,5].

Several approaches and tools have been presented for producing class diagrams from requirements automatically/semi-automatically, e.g., NL-OOPS [6], LIDA [7], CM-Builder [8], DC-Builder [9], and ABCD [4]. These methods focus on the NL requirements automating and analyzing, and generation of class models from these requirements [1]. Moreover, the majority of these approaches concentrate on the extraction of the class diagram and a few of them can extract other diagrams, e.g., the behavior diagrams. Most of the studies produce incomplete diagrams and require high user interventions and interactions. Moreover, the wide-ranging UML class elements are challenging be extracted, e.g., attributes, operations, and enhanced association types such as aggregation, composition, generalization, and dependency [10].

This paper surveys the approaches and tools proposed to generate class diagrams and provides a review of their strengths and limitations. Different requirement representations, e.g., unrestricted and restricted requirements, use case descriptions required by these approaches, and the techniques used by each approach to transforming NL requirements into UML class models are studied and analyzed.

The paper is organized as follows: Section II outlines the Requirements Engineering concepts, UML, and NLP. Section III summarizes the research of the transformation from NL requirements into the UML class diagram. Section IV presents the results, and Section V concludes the paper.

II. NATURAL LANGUAGE AND REQUIREMENT ENGINEERING

A. Requirement Engineering

Requirement engineering emphasizes the use of systematical and repeatable techniques to make sure that system requirements are complete, consistent, and relevant [11]. It includes the user, the developer in the process, hence it is a very complex process. The users understand the problem and know their needs nevertheless not how to develop a system, whereas developers know how to construct a system, yet, do not know what the problem is. The software system requirements are the descriptions of the services that a system should provide and its operational constraints [2, 11].

B. Requirements Documentation using Natural Language

Software requirements are often documented utilizing natural language manuscripts [32, 33]. In contrast, stakeholders are more acquainted with the NL and they do not have to learn a new notation. Moreover, requirements engineers can use natural language to express any kind of requirement, which is an advantage of natural language text [34, 35]. However, natural language allows requirements to be ambiguous, and requirements of different perspectives are at risk of being unintentionally mixed up during documentation [12, 36]. Requirements can be written using either an unrestricted or restricted NL. A restricted NL is obtained by placing restrictions on the natural language text. It aims to reduce the problems of unrestricted NL such as redundancy and ambiguity to facilitate automated analysis of the requirements [13, 37].

C. Unified Modelling Language (UML)

UML is the de facto standard for object-oriented software modeling, which was standardized and adopted by Object Management Group (OMG), and it became a software development standard [2]. UML a standard formal language for modeling and documenting software systems. UML can be used for describing and designing software systems graphically, both at the requirements and design phases of an SDLC [14, 38].

D. Natural Language Processing and Software Engineering

Natural Language Processing (NLP) and Software Engineering (SE) and are both branches of computer science and engineering, which can be applied to each phase in the SDLC [2, 39]. Requirements written in natural languages can be very problematic. Firstly, these natural language requirements need to be analyzed. Then, NLP tools and techniques are needed to be used to help in linguistic analysis and to create an automated requirement analysis support tool [15]. The use of NLP techniques in requirements engineering is very important as the NL requirement specifications are written by a software analyst in collaboration with the users, and the customers, and if the requirements document is written in formal language the customers would not sign a contract [2].

III. CLASS DIAGRAM GENERATION TOOLS

A semi-automatic tool that is used syntactic knowledge and needs to generate object diagrams from NL SRS is described in [16]. The tool uses OMT concepts and a link grammar parser for the transformation from the specifications into object diagrams. A list of guidelines has been collected, which is expressed in parsing rules. A post-processor is developed to apply these guidelines to the parser output and extract objects, attributes, and associations. The tool uses refined guidelines and a graph drawing tool to display the diagrams. However, the approach treats a small number of guidelines. The diagram is manually refined and validated and the user needs to have extensive domain knowledge. The generated diagrams were not completely acceptable due to many factors, i.e., parser insufficiency, ambiguous or incomplete descriptions, insufficient domain knowledge, the inadequacy of guidelines and transformation rules. The tool does not include relationships such as generalization, composition, and dependency, or the multiplicity of relationships.

NL-OOPS is a tool presented using a semantic network (SN) of words of an NLP system, which can generate object models from unrestricted NL requirements [6]. The requirements are morphologically, syntactically, semantically, and pragmatically processed, and then converted into an intermediate model SN, which is a semantic graph used to bridge the gap between the NL requirements and object models. Finally, the SN is transformed into object models. The tool considers nouns as objects and uses links to identify relationships. However, the tool lacks precision in picking the objects for large systems and cannot distinguish between objects, and their respective attributes, class nouns, and attribute nouns. The tool requires user intervention for editing produced classes and does not produce the class diagrams.

A semi-automatic system called RECORD is described in [17]. The system generates object models from NL requirements expressed in use case descriptions. Therefore, a form-based user interface is used to support the structured input of requirements. The nouns in the keywords are transformed into objects and the verbs to behaviors. The use cases are analyzed to extract objects and their components. Then, the use cases are classified using the extracted information for discovering object models. Finally, the results processes are reviewed and adjusted. However, the system requires excessive user interaction to manually links and edits the generated models.

A project called D-H, which presents a linguistic tool for knowledge extraction from NL requirements is presented in [18]. The D-H performs automatic syntactic analysis by DIPETT and semi-automatic semantic analysis by a separate module called HAIKU. The DIPETT and HAIKU are robust components of a text analysis system, called TANKA. The D-H can identify candidate objects from noun phrases and candidate processes from verbs, attributes from adjectives, and adjectival phrases. However, the approach should be duly assisted by humans and only identifies simple association relationships without identifying multiplicity.

LIDA is a semi-automatic tool presented to manually extract class and object diagrams from unrestricted NL requirements [7]. The NL description imports and the POS are identified from NL text. Then, the analyst works manually to identify candidate classes and removing poor classes or candidates to be attributes from the noun list. Finally, the analyst identifies candidate methods and roles from the verb list and then uses LIDA Modeler to graphically associate the extracted components with the appropriate classes. However, LIDA needs extensive user interaction and only capable to assist the users in generating class diagrams, but not automatically generates complete models. The approach does not support traceability and nor identify aggregation and generalization.

CM-Builder is an NLP-based tool proposed to extract a class model automatically from unrestricted NL requirements using NLP methods to examine the requirements and build a model represented in a Semantic Network (SN), which is used to build a class model that can be directly visualized for further refinements to generate a final class model[8]. However, the approach does support traceability. The linguistic analysis is limited due to the problems of NL. The approach can only derive structural model elements and they may not be connected. It does not identify methods, aggregation, generalization, and relationships multiplicity. The suggested transformation rules are not structured, and their completeness is not evaluated. Three requirements pre-processing techniques are used; hence the efficiency of the approach is low. For large specifications, the user may be overwhelmed with candidate classes.

An automated conceptual modeling prototype is proposed to produce a class diagram from a requirement document using NLP and domain ontology [19]. The NL requirements are analyzed using NLP. Then, the class identification performance is improved using a dictionary of domain-based ontology. This approach extracts classes employing NLP via a tagger, a link grammar parser, parallel structure, and linguistic patterns. The final results are further refined using a domain ontology dictionary. It can identify many relationship types. However, the approach requires the analysts to be involved to make many decisions during the modeling. Conversely, this approach deals with only the basic OO concepts.

REBUILDER UML is a class diagrams generation tool from NL text [20]. This module uses an approach based on CBR (Case-Based Reasoning) and NLP that performs morphology, syntax, and semantics analysis, then, a CBR engine is used to retrieve cases from the case library. It consists of four modules: the UML editor, the knowledge base manager, the knowledge base (KB), and the last module is the CBR engine, which is the reasoning module. However, this tool needs continuous up-gradation of the case base. Conversely, if a query-related case is not available in the case base, the case is not created. There are some performance issues, e.g., finding the semantic distance between two concepts can take several seconds due to the use of WordNet, which is unacceptable for the system usage. The objects represent abstract concepts and do not necessarily represent classes to be implemented. The shallowness of WordNet is not acceptable for specific domains like computing and software engineering.

MOVA is a tool designed to draw, measure, and validates class diagrams [21]. It allows users to analyze invariants and assess OCL metrics. However, the user had to be involved in all these processes to help in identifying OO concepts since the tool is incapable to identify them automatically. Finally, the class and object diagrams are saved in an XML format, which precludes the models from being exchange with other tools. However, MOVA requires high human intervention since it incapable of automatically identifying OO concepts. MOVA Meta model is only a subset of the UML meta-model; it does not support the full OCL syntax and has limited support for OCL and UML. Moreover, advanced relationships like aggregation, composition, and dependency are not included.

A semi-automated approach that aimed to solve the problems in an NL SRS is described in [22]. The approach consists of three steps. Firstly, parsing the Natural Language SRS; then eliciting OO elements to create an OO analysis model, and finally, the diagram is generated, which is reviewed by a human reviewer to detect ambiguities and inconsistencies. However, the static parser is used to construct the grammar. Thus, it is restricted to handle static relationships and cannot deal with the dynamic model. The models obtained are highly incomplete with many unconnected components, contain many isolated classes, and redundant classes and relationships. The diagrams lack several relevant classes and relationships. Moreover, several identified domain classes are difficult to be semantically termed as domain classes as they seem to violate the encapsulation principle.

A method called Relative Extraction Methodology is proposed to generate a class diagram from the NL problem statement [23]. Initially, NLP is used for sentence separation and to extract the subject, object, and predicate from the sentence. This information is used to produce a graphical representation named a dependency graph, which acts as a knowledge base. It allows the user to add new attributes or delete the wrong classes. However, it needs human intervention and the developer involvement in the refinement process which is a limitation of this approach. Moreover, the accuracy of the generated classes and their components decreases for complex problem statements. Relationships such as aggregation and dependency are missing, and the multiplicities between objects were not considered.

UMLG is an NLP-based system to generate a class diagram from NL requirements that follows NLP methods and a rule-based approach [10]. This system designs in six modules: Text input acquisition, text understanding, knowledge extraction, generation of UML diagrams, and finally multi-lingual code generation. First, UMLG reads and tokenizes the requirements text, POS tags are identified, e.g., nouns, adjectives, etc. The main parts of a sentence, e.g., subjects, objects, etc. are identified. Then, to extract the UML the NL text is semantically analyzed. Finally, the system generates a class diagram using the extracted information and provides the respective blocks of programming source code. However, the system needs more enhancement to extract more classes and diagrams. It does not identify attributes, multiplicity, and relationships such as aggregation, generalization, composition.

A method to generate a high-level class diagram from a structured NL requirement document is proposed in [24]. This approach is implemented as a tool named FDCT using heuristic rules and a domain-specific glossary. Requirements Analysis Tool (RAT) has been developed to be used to put the requirement sentences in restricted form, and perform lexical and semantic analysis. The process has three phases. First, the requirements statement converts into a set of tokens with the help of glossaries defined by the user. Second, the requirement statements' syntax is analyzed by using the state machines. The third phase comprises semantic analysis with the help of domain-specific ontology. The tool identifies only two kinds of relationships, i.e., association and generalization. The generated classes require expert intervention to transform them into implementation-level fine-grained classes. Also, the produced high-level design can be too fine-grained/coarse-grained depending on the statements' granularity, which may cause an enormous number of classes.

A methodology for class diagram generation from NL text, based on which a tool named RACE has been developed to extract the classes and relationships using NLP and domain ontology is proposed in [25]. This tool finds candidate classes through a POS tagger and uses the domain ontology to refine the output. The system can find concepts based on nouns, noun phrases, and verbs analysis, and defined association, aggregation, composition, generalization, and dependency relationships. However, RACE is not platform-independent, it is restricted to the Windows platform and not able to run on other platforms. It could not identify the multiplicity of relationships, and limited to processing simple statements, and does not focus on the program's internal structure.

A tool was proposed, which can perform OO analysis of SBVR software requirements specifications [1]. First, the user inputs a software requirements specification in English, and the NL to SBVR approach generates SBVR based controlled representation of requirement specification by performing lexical, syntactic, and semantic parsing and SBVR vocabulary is extracted. The OO information is extracted from the SBVR's rule-based representation. Finally, a class model is generated. However, this tool does not deal with natural language constraints.

A tool named SBVR2UML was proposed to map SBVR representation to a UML class model [26]. First, the user enters the SBVR specification as input, and then lexical, syntactic, and semantic analyses are performed. After that, SBVR vocabulary is extracted from given SBVR rules. Then, the SBVR rule is further processed to extract the relevant information that maps into UML class elements. Finally, a class model is graphically generated. However, requirements need to be written in the form of SBVR representation, because this approach only takes the requirements specified in SBVR syntax.

An approach to transforming informal NL requirements into UML class diagrams proposed in [27]. The approach is implemented as a tool named RAPID using several NLP technologies such as an OpenNLP to perform lexical and syntactical analysis; Stemming Algorithm to find the root of words; and WordNet, which performs analysis of semantics for semantic correctness validation. Then, the Class Extraction Engine module applies a set of heuristic rules on the output of the previous module to generate class diagrams, which are then refined using domain ontology. However, it is limited to processing simple statements as each sentence in the requirements document must meet a pre-defined structure.

An approach is proposed to convert textual requirements into class diagrams based on domain ontology and NLP [28]. A tool called RAUE filtering algorithm has been implemented along with applications such as OpenNLP parser, WordNet, and Java Native Interfaces. OpenNLP is for extracting information by used lexical, syntactic parser, and POS tagging. RAUE can identify concepts based on noun phrases, and verb analysis, and relationships, e.g., association, aggregation, generalization, dependency, and multiplicity of these relationships. However, RAUE is limited to processing simple statements.

DC-Builder is a tool to analyze textual requirements using NLP techniques and domain ontologies to extract a class diagram [9]. First, the GATE framework is used to analyze the NL requirements. Then, to extract UML elements from the text, a set of heuristic rules are defined. Thus, it produced an XML file that contains mistaken concepts. The ontologies are used to eliminate unrelated concepts, and then keep only the final class diagram elements. However, the heuristic rules do not cover all the sentence structures. DC-Builder requires manual intervention and relationships multiplicity not included.

An architecture of requirements specification using an NLP is developed in [29]. This work focuses on the verification of requirements and the automatic extraction of objects from a requirements document. The system comprises a tokenizer to tokenize the input sentences, an NLP parser to parse the requirement sentences and extract the nouns, and perform the filtering of irrelevant terms, classify the remaining terms into one of three categories, and insert objects into a project knowledgebase. However, the parsing system is unable to perform syntactic parse trees disambiguation, compound noun and proper noun processing, anaphoric resolution, and semantic interpretation of terms.

An approach to transforming NL requirements into class diagrams is described in [30]. The requirements statements are transformed into an intermediary frame-based structured representation using a dependency analysis and Grammatical Knowledge Patterns (GKPs). The class diagrams are generated from the knowledge stored in the frame-based structured representation by using a rule-based algorithm. This approach produced class diagrams based on linguistic analysis with annotation or manual intervention. The requirements representation is stored in an intermediate form that can accept user changes. However, this approach does not identify the multiplicity of the relationship and does not integrate with a graphical CASE tool to produce graphical class diagrams.

A tool is developed to managing textual requirements based on NLP and application-specific ontologies [31]. An NLP tool named NLTK receives unstructured requirement text and performs sentence segmentation. After that, the text entered into the word tokenization process to tokenize text into words or punctuation characters and normalize them through the stemming process. Then, POS tagging is performed to identify the role of each word in the sentence; noun, verb, adjective, etc. Then, groups of tokens especially noun phrases are identified through the chunking process. However, this approach generates a class diagram but some relationships like composition, dependency, generalization are not included.

ABCD is an automated tool designed to convert NL requirements to class diagrams [4]. This tool uses NLP techniques combined with pattern rules. It applies lexical and syntactical processing. The text preprocessing consists of four steps; sentence splitting, tokenization, POS tagging, and syntactic parsing. Then, a pattern-matching NLP technique is used to extract the class diagram concepts such as aggregation, composition, and generalization, which are saved into an XMI file. Finally, a CASE tool called ArgoUML is used to build the corresponding UML diagrams from the XMI file. However, the tool deficiencies to handle redundant information problems and confuses the concepts of association and method identification as both are identified by verbs.

An approach to convert SRS into UML class models and developed as a tool named SUCM is presented in [3]. The tool uses OpenNLP for semantic analysis to extract tokens and generate POS tags. Then, it uses the SBVR standard to extract the OO classes from the NL processed SRS. SUCM can identify associations, generalization, aggregation relationships, and multiplicity. The techniques used to obtain good accuracy in less time. Nonetheless, it can only generate UML class diagrams, which just models the structure of a system and one diagram is not enough.

IV. DISCUSSIONS

In summary, after the study and analysis of the existing literature, we could conclude that it seems that there is no comprehensive attempt has been made for the UML class diagrams generation from the NL requirements. All the approaches are either highly complex or have a lot of limitations. Some of these solutions could identify classes and generate object models; though, the generated diagrams often comprise redundant classes, while leaving the needed classes. Some important and more enriched relationship types such as association, generalization, aggregation, composition, and dependency are not provided in most existing tools.

There is no framework for an automatic generation of complete class diagrams or other UML diagrams from free-text requirements documents. Most of the earlier tools do not allow the user to visualize UML diagrams and some of the existing tools require human interactions for the automatic development of UML diagrams along with associated attributes and methods. Only a few approaches are fully automatic. Furthermore, most of the existing tools accept only a small set of requirements and require developers' support in the refinement process and identify inconsistencies in requirements. The existing tools more or less require the requirements to be written in a restricted language or to be written in a specific form instead of NL-free texts.

Table I presents a comparison of the discussed tools based on the input followed by the level of automation (viz. manual, semi-automatic, automatic), the output of the approach, and the used techniques.

V. CONCLUSION

This paper aims to provide a review of existing approaches and tools for generating UML class models from NL text. These approaches and tools use different techniques and diverse levels of linguistic analysis to extract the UML class diagrams from NL requirements. The paper deeply studied several works, compared them, and identified the strengths and weaknesses of each of them. Some of these tools can automatically extract the UML elements and produce class models from natural language text. In contrast, most of the tools require consistent user intervention and interaction in the process of UML class diagrams extraction. Even with the substantial enhancements that have been made recently, it seems that we cannot say that solutions could generate all the UML elements and data semantics automatically, i.e., class names, operations, and relationships, i.e., associations, and other advanced relationship types such as generalization, aggregation, and dependency.

REFERENCES

[1] I. Bajwa and M. Choudhary, "From natural language software specifications to UML class models", *Int. Conf. on Enterprise Inf. Systems (ICEIS)*, Berlin, pp. 224–237, 2011.

[2] O. Dawood and A. Sahraoui, "From Requirements Engineering to UML using Natural Language Processing – Survey Study", *European Jour. of Eng. Res. and Science*, vol. 2(1), pp. 44-50, 2017.

[3] M. Mohanan and P. Samuel, "Natural Language Processing Approach for UML Class Model Generation from Software Requirement Specifications via SBVR," *Int. Jour. on Artif. Intell. Tools*, vol. 27(06), 2018.

[4] W. Karaa, Z. Azzouz, A. Singh, N. Dey, A. Ashour and H. Ghazala, "Automatic builder of class diagram (ABCD): an application of UML generation from functional requirements," *Jour. of Soft. Practice and Experience*, vol. 46(11), pp. 1443-1458, 2015.

[5] M. Abdouli, B. Karaa and H. Ghezala, "Survey of Works that Transform Requirements into UML Diagrams," *14th Int. Conf. on Soft. Eng. Research, Manag. and App. (SERA)* , pp. 117-123, 2016

TABLE I: A COMPARISON OF THE CLASS GENERATION APPROACHES AND TOOLS

Study	Input	Automation	Output	Technique Used
[16]	High-level specification (SRS)	Semi-automatic	Object diagram	OMT guidelines+ NLP link grammar parser
[6]	Unrestricted NL	Semi-automatic	Object diagram	NLP heuristics
[17]	Use Case descriptions	Semi-automatic	Object diagram	NLP
[18]	NL requirements	Automatic	Object diagram	Linguistic tools
[7]	Unrestricted NL requirements	Semi-automatic	Class+ Object diagrams	NLP heuristics (Chen's rules)
[8]	Unrestricted NL requirements	Automatic	Class diagram	NLP rules
[19]	Unstructured NL requirements	Automatic	Class diagram	NLP techniques+ domain ontology
[20]	NL textual requirements	Automatic	Class diagram	Case-based reasoning + NLP
[21]	Textual requirements	Semi-automatic	Class+ Object diagrams	(Rewriting-based UML) programming + OCL
[22]	Restricted NL requirements	Semi-automatic	Class diagram	Grammar+ NL parser+ rules
[23]	NL problem statements	Semi-automatic	Class diagram	Dependency Graph
[10]	NL requirements	Automatic	Class diagram+code	NLP heuristic+ rule-based algorithm
[24]	Restricted NL requirements	Semi-automatic	Class diagram	Heuristic rules+ domain-specific glossary
[25]	Informal NL requirements	Automatic	Class diagram	NLP +domain ontology
[1]	NL software specification	Automatic	Class diagram	Linguistic Analysis+ SBVR business rules
[26]	SBVR specification of software requirements	Automatic	Class diagram	Linguistic Analysis+ SBVR business rules
[27]	Informal NL requirements	Semi-automatic	Class diagram	NLP + domain ontology
[28]	Informal NL problem statements	Automatic	Class diagram	NLP +domain ontology
[9]	NL textual requirements descriptions	Automatic	Class diagram	NLP heuristic rules +domain ontology
[29]	NL requirements document	Automatic	Object diagram	NLP
[30]	Informal NL textual requirements	Automatic	Class diagram	Syntactic dependency analysis + GKPs
[31]	Informal NL textual requirements	Automatic	Class diagram	NLP (NLTK+ ontologies)
[4]	NL textual requirements	Automatic	Class diagram	NLP techniques+ pattern rules
[3]	Software Requirements Specification (SRS)	Automatic	Class diagram	NLP + SBVR business rules

[6] L. Mich and R. Garigliano, “NL-OOPS: A requirements analysis tool based on natural language processing,” *the 3rd Int. Conf. on Data Mining Methods and Databases for Eng.*, Italy, pp. 322_330, 2002.

[7] S. Overmyer, B. Lavoie and O. Rambow, “Conceptual Modeling through Linguistics Analysis Using LIDA,” *the 23rd Int. Conf. on Soft. Eng. (ICSE’01),* Canada, pp. 401-410, 2001.

[8] H. Harmain and R. Gaizauskas, “CM-Builder: A Natural Language-based CASE Tool,” *Jou. of Auto. Soft. Eng.*, vol. 10(2), pp. 157-181, 2003.

[9] H. Herchi and W. Abdessalem, “From user requirements to UML class diagram,” *Int. Conf. on Comp. Related Knowledge (ICCRK’ 2012)*, Tunisia, 2012.

[10] I. Bajwa, A. Samad and S. Mumtaz, “Object-Oriented Software Modeling Using NLP Based Knowledge Extraction,” *Europ. Jour. of Sci. Research*, vol. 35(1), pp. 22-33, 2009.

[11] I. Sommerville, *Software engineering*, 10th ed. Boston: Pearson Education, 2016.

[12] K. Pohl and C. Rupp, *Requirements engineering fundamentals*, 2nd, California: Rocky Nook, 2015.

[13] T. Yue, L. Briand and Y. Labiche, “A systematic review of transformation approaches between user requirements and analysis models,” *Requirements Eng.*, vol. 16(2), pp. 75-99, 2010.

[14] OMG, “Unified Modeling Language (UML),” v. 2.5, 2013. Available at: http://www.omg.org/spec/UML/2.5.

[15] A. Lash, K. Murray and G. Mocko, “Natural language processing applications in requirements engineering,” *the Int. Design Eng. Technical Conf. & Computers and Inf. in Eng. Conf.*, USA, pp. 1-9, 2012.

[16] S. Nanduri and S. Rugaber, “Requirements Validation via Automated Natural Language Parsing,” *Jour. of Manag. Inf. Sys.*, vol. 12(3), pp. 9-19, 1995.

[17] J. Börstler, “User-Centered Requirements Engineering in RECORD - An Overview,” *Nordic Workshop on Programming Environment Research*, Denmark, pp. 149-156, 1996.

[18] S. Delisle, K. Barker and I. Biskri, “Object-Oriented Analysis: Getting Help from Robust Computational Linguistic Tools,” *the 4th Int. Conf. on Appl. of Natural Language to Inf. Sys.*, Austria, 167- 171, 1999.

[19] N. Zhou and X. Zhou, “Automatic Acquisition of Linguistic Patterns for Conceptual Modeling,” *INFO 629: Artificial Intell.*, pp. 1-19, 2004.

[20] A. Oliveira, N. Seco and P. Gomes, “A CBR Approach to Text to Class Diagram Translation,” *the 8th European Conf. on Case-Based Reasoning*, Turkey, 2006.

[21] M. Clavel, M. Egea and V. Silva, “The MOVA Tool: A Rewriting-Based UML Modeling, Measuring, and Validation Tool,” *the 12th Conf. on Sof. Eng. and Databases*, Spain, 2007.

[22] D. Popescu, S. Rugaber, N. Medvidovic and D. Berry, “Reducing Ambiguities in Requirements Specifications Via Automatically Created Object-Oriented Models,” *Monterey Workshop*, pp. 103-124, 2008.

[23] H. Krishnan and P. Samuel, “Relative Extraction Methodology for class diagram generation using dependency graph,” *Int. Conf. On Commun. Control & Comp. Tech*, pp. 815-820, 2010.

[24] V. Sharma, S. Sarkar, K. Verma, A. Panayappan and A. Kass, “Extracting high-level functional design from software requirements,” *16th Asia-Pacific Soft. Eng. Conf.*, pp. 35-42, 2009.

[25] M. Ibrahim and R. Ahmad, “Class diagram extraction from textual requirements using natural language processing (NLP) techniques,” *2nd Int. Conf. on Comp. Research and Develop.*, pp. 200-204, 2010.

[26] H. Afreen and I. S. Bajwa, “Generating UML Class Models from SBVR Software Requirements Specifications,” *23rd Conf. on Artif. Intell.*, pp. 23-32, 2011.

[27] P. More and R. Phalnikar, “Generating UML Diagrams from Natural Language Specifications,” *Jour. of Applied Inf. Sys.*, vol. 1(8), pp. 19-23, 2012.

[28] S. Joshi and D. Deshpande, “Textual Requirement Analysis for UML Diagram Extraction by using NLP,” *Jour. of Comp. Appl.*, vol. 50(8), pp. 42-46, 2012.

[29] S. MacDonell, K. Min and A. Connor, “Autonomous requirements specification processing using natural language processing,” *arXiv preprint arXiv: 1407.6099*, 2014.

[30] R. Sharma, P. Srivastava and K. Biswas, “From Natural Language Requirements to UML Class Diagrams,” *2nd Workshop on Artif. Intell. for Requirements Eng*, pp. 1-8, 2015.

[31] A. Arellano, E. Carney and M. A. Austin, “Natural language processing of textual requirements,” *The 10th Conf. on Sys.*, Spain, pp. 93-97, 2015.

[32] J. Thakur and A. Gupta, “Automatic generation of analysis class diagrams from use case specifications,” *arXiv preprint arXiv:1708.01796*, pp. 1-41, 2017.

[33] A. M. Maatuk and E A. Abdelnabi. “Generating UML Use Case and Activity Diagrams Using NLP Techniques and Heuristics Rules”, 2021. In the 3rd International Conference on Data Science, E-learning and Information Systems 2021 (Data'2021), Petra, Jordan, 6pp.

[34] F. Mohammed, Z. Abdelnabi, A. M. Maatuk, and A. Abdalla. “An Experimental Study on Detecting Semantic Defects in Object-Oriented Programs using Software Reading Techniques”. In *Pro. of ACM* ICEMIS, 2015, DOI=http://dx.doi.org/10.1145/2832987.2833025

[35] A. M. Maatuk, M. A. Ali and S. Aljawarneh. “Translating Relational Database Schemas into Object-based Schemas: University Case Study”. In *Recent Patents on Computer Science*. Innovations in Educational Technology and E-learning Social Networking. Vol. 8, No 2, pages 122-132, 2015. **DOI:** 10.2174/2213275908666150710174102.

[36] S. F. Alshareef, A. M. Maatuk, T. M., Abdelaziz, and M. Hagal. “Validation Framework for Aspectual Requirements Engineering (ValFAR)”. *In Proc. of the 6th Int. Conf. on Eng. & MIS,* 2020, pp. 1–7. DOI:https://doi.org/10.1145/3410352.3410777</bib>

[37] S. Alshareef, A. M. Maatuk, T. Abdelaziz. “Aspect-Oriented Requirements Engineering: Approaches and Techniques”. In *DATA '18*, 2018, https://doi.org/10.1145/3279996.3280009

[38] T. M., Abdelaziz, A. M. Maatuk and F. Rajab. “An Approach to Improvement the Usability in Software Products”. In *Int. Jour. of Soft. Eng. & Applications*, Vol.7(2), 2016.

[39] E. A. Abdelnabi, A. M. Maatuk, T. M. Abdelaziz, and S. Elakeili. 2020. “Generating UML Class Diagram using NLP Techniques and Heuristic Rules”, In 20th Int. Conf. on Sciences and Techniques of Automatic Control and Computer Engineering (STA). IEEE, Tunisia, 277-282. https://doi.org/10.1109/STA50679.2020.9329301