Generating UML Class Diagram using NLP Techniques and Heuristic Rules

Conference Paper · December 2020

DOI: 10.1109/STA50679.2020.9329301

4 authors:

Esra A. Abdelnabi
University of Benghazi
6 PUBLICATIONS **67** CITATIONS
SEE PROFILE

Abdelsalam M. Maatuk
University of Benghazi
94 PUBLICATIONS **1,419** CITATIONS
SEE PROFILE

Tawfig Abdelaziz
Libyan Interntonal Mdical University
26 PUBLICATIONS **143** CITATIONS
SEE PROFILE

Salwa Elakeili
University of Benghazi
19 PUBLICATIONS **72** CITATIONS
SEE PROFILE

Generating UML Class Diagram using NLP Techniques and Heuristic Rules

Esra A. Abdelnabi
Faculty of Information Technology
Benghazi University, Libya
Esraaali9179@gmail.com

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

Tawfig M. Abdelaziz
Faculty of Information Technology
Libyan International Medical University, Libya
tawfigtawill@limu.edu.ly

Salwa M. Elakeili
Faculty of Information Technology
University of Benghazi, Libya
salwa.elakeili@uob.edu.ly

***Abstract*—Several tools and approaches have been proposed to generate Unified Modeling Language (UML) diagrams. Researchers focus on automating the process of extracting valuable information from Natural Language (NL) text to generate UML models. The existing approaches show less accurateness because of the ambiguity of NL. In this paper, we present a method for generation class models from software specification requirements using NL practices and a set of heuristic rules to facilitate the transformation process. The NL requirements are converted into a formal and controlled representation to increase the accuracy of the generated class diagram. A set of pre-defined rules has been developed to extract OO concepts such as classes, attributes, methods, and relationships to generate a UML class diagram from the given requirements specifications. The approach has been applied and evaluated practically, where the results show that the approach is both feasible and acceptable.**

Keywords - Software Engineering, Natural Language Processing, Requirement Engineering, UML, Natural language.

I. INTRODUCTION

Requirements engineering can be problematic, as it requires extra work in requirements elicitation, verification, and traceability. The extracted requirements that are documented as software requirements specification (SRS) are transformed during the system design phase into specific models, e.g., Object-Oriented (OO) models using UML [1]. After preparing the SRS document, some requirements might be missed. This might lead to an incomplete system, and extra cost and time.

During the analysis in System Development Life Cycle (SDLC), the natural language (NL) is used to describe the specific problem that needs to be solved [11, 12]. However, NLs are often ambiguous, uncertain, incomplete, and incoherent. Besides, the requirement document size and the script of NL can cause extra problems. Some essential information needed by the verb for completeness is missed. To facilitate developing accurate UML diagrams, the requirement analyst has to detect and fix these problems. However, if the analyst has no enough domain knowledge, this can miss the defects in NL, which can lead to various interpretations in implicit requirements recovering. Moreover, it would be expensive to fix such issues in the subsequent SDLC phases. Conversely, software systems require extra cleanness and correctness, which are missing in NLs [2]. Analyzing and reconstructing NL requirements and generating UML models is a challenging task, which needs automated support. For that reason, some approaches and tools have been developed to facilitate this process. NL processing (NLP) techniques play an essential role in analyzing NL requirements and generating UML diagrams from these requirements [3]. Software engineering developments depend on OO design using UML for software development. The UML class diagram is the most suitable tool to describe a comprehensive understanding of requirements [3].

To resolve such problems and automating the generation of UML models process from requirements, a lot of techniques have been proposed [1, 13, 14, 18]. Conversely, the experiments with these tools show that they are not used SDLCs because of their restrictions. Moreover, most of the tools concentrate on the class diagrams with high user interventions to finish the process. Besides, most of them are unable to extract the complete UML elements, e.g., methods, associations, and other advanced relationship types. Some solutions just extract the names of classes, and some of them deal with objects [2, 20]. In recent decades, several tools are proposed to analyze NL requirements and produce UML models, yet they have emphasized the analysis of NL requirements and extremely dependent on user intervention. Many approaches use NLP techniques, mapping, graphs, and patterns, whereas others use ontology and linguistic concepts.

This paper describes a proposed method for analyzing the NL requirements and extracting the relative software information and concepts to facilitate the process of generating a UML class diagram from unrestricted NL requirements. Our approach reduces the ambiguity and complexity of NL by using NLP techniques. We proposed a set of heuristics rules to perform the transformation process. The results were encouraging and support the combination of NLP techniques and heuristic rules, which combines the strength of automation and human reasoning. The method has been applied and evaluated using a case study, the results of which demonstrate that it is practical and satisfactory.

The remainder of the paper is organized as follows: Section II presents the related work. Section III introduces the

proposed approach. Section IV discusses some of the obtained results, and Section V concludes the paper.

II. RELATED WORK

Several approaches have been presented for generating UML diagrams from requirements. They have focused on the automation of the process by analyzing the NL requirements while generating the models.

DC-Builder is a tool, which uses NLP techniques to analyze requirements and ontologies to extract class diagrams [4]. The domain ontology advances the quality of outputs. The UML concepts such as classes, attributes, and associations are extracted using a set of heuristic rules. However, the heuristic rules presented are not exhaustive and certain sentence structures are not covered. DC-Builder cannot identify either methods or the multiplicity of relationships and requires manual intervention.

The architecture of SRS using an NLP is described in [5]. This work focuses on specification verifications. The architecture consists of three modules with a user interface, which are (i) a tokenizer that reads sentences from a document, (ii) an NLP parser that parses each requirements sentence and extracts all unique noun terms, and (iii) a term management system that performs the filtering of unimportant terms. However, this system is unable to identify the relationships between classes and objects or multiplicity.

An approach based on Grammatical Knowledge Patterns (GKPs) is proposed for class diagrams generation from requirements [6]. The requirements statements are transformed into a frame-based representation using a dependency analysis of requirements statements and the GKP. However, the approach can only generate class diagrams and unable to identify the multiplicity of relationships.

A tool was proposed to understanding and managing the requirements by using application-specific ontologies and NLP techniques [7]. The tool, known as NLTK, has been used to receive unstructured requirements and makes segmentation to the text to acquire sentences. Then, text entered to be tokenized to words or punctuation characters that then normalized via the stemming. Moreover, a part of speech (POS) tagging is performed to identify the word roles; then, identify groups of tokens, and recognize requirements. However, this approach ignores more enriched relationships such as generalization, composition, and dependency, yet many-to-many multiplicity is not identified.

An approach to generate class diagrams from requirements document as use case specifications (UCSs) is presented in [8]. The UCSs are taken as input, and the Stanford parser is used to produce type dependencies and POS tags of each sentence in the UCS. A set of comprehensive rules is applied to systematically process, interpret the sentences, and identify the elements for the generation of analysis class diagram. However, the approach requires the UCSs to be scripted using some restrictions to handle the NL issues such as ambiguity and inconsistency. Moreover, it does not use entity disambiguation techniques such as misspelling, abbreviation, and alias identifications, etc.

An NLP approach is proposed for UML class generation using NLP techniques to avoid requirements manual processing [9]. However, the approach does not consider relationships and other class diagram concepts such as methods and multiplicities.

The method described in [10] focuses on the formulation of rules for the extraction of class elements from semi-structured requirements using NLP techniques. The rules use keywords such as use case name, actor, etc., and the dependencies among the sentence words. The process starts with classes, attributes, methods, and relations extractions. However, the rule-based techniques cannot cover all the conditions, so that the approach does not include advanced relationships like aggregation, dependency and the multiplicity of relationships. Furthermore, more rules and patterns for different sentence arrangements are needed.

III. THE PROPOSED APPROACH

This section describes the proposed approach, which works in five phases as shown in Fig. 1. Phase I: Reconstruction and Normalization of NL text, Phase II: Processing NL requirements, Phase III: Applying mapping rules to extract UML elements, Phase IV: Refinement of the results, and finally, Phase V: UML diagrams generation.

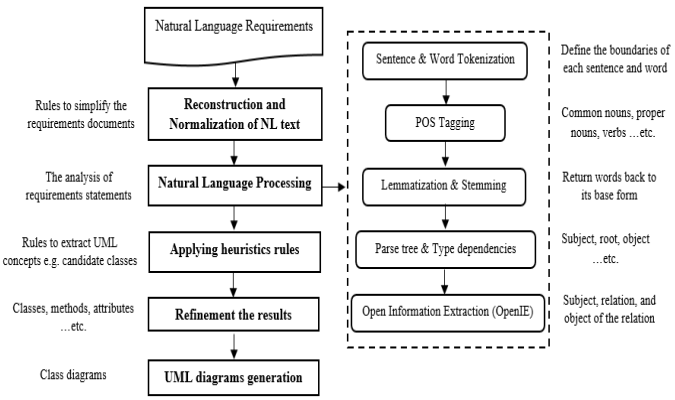

Fig. 1: Generating Class diagrams from NL requirements

A. Phase I: Reconstruction and Normalization of NL Text

To reconstruct the NL text and normalizing the complex requirements into simple statements, a set of syntactic rules is proposed to increase the knowledge accuracy extracted from the requirements document [14]. These rules could be used for writing and normalizing the requirements documents. These rules should be applied while writing the requirements document to facilitate the process of extracting the UML elements, to give better results, and to save time and effort spent on analyzing the requirements.

- **Rule 1**: The NL text is reconstructed to Subject-Predicate (S-O) or Subject-Predicate-Object (S-V-O) forms.
- **Rule 2**: If a sentence is of the form (S-V1-O1-V2-O2), then it splits into more sentences (S-V1-O1) (S-V2- O2).
- **Rule 3**: If a sentence consists of several actions or subordinate sentences, then separate it into more sentences, only one subject and one predicate or action.

(NP1-VP1-NP2- *and* -NP3- *and* -VP2-NP2- *and* -NP3) convert it into two sentences (NP1-VP1-NP2- *and* -NP3) (NP1-VP2-NP2- *and* -NP3).

- **Rule 4:** If a sentence is separated by connectives like *"and, or, but, yet",* e.g., (S-V-O1-, /*and*-O2-, /*and*,-O3), then split into two (or more) simpler sentences (S-V-O1) (S-V-O2) (S-V-O3).
- **Rule 5:** If a sentence is of the form of ((S)-V-O1-, /and-O2-, /and, - O3…) but the only verb leads equal parallel structure, then split it into ((S)-V-O1) (V-O2) (V-O3).
- **Rule 6**: If a sentence is a passive voice, translated it to an active voice, which can be translated to a message sent.
- **Rule 7**: Do not use diverse verbs for the same action, but use the same verb for the same action in different sentences.
- **Rule 8**: Replace a pronoun of the object with the object name in the normalized sentence to resolve the ambiguity.
- **Rule 9:** Start a conditional sentence with If-clause, which contains a condition, and terminate it with endIf-clause. **Rule 10:** Start a set of concurrent actions with StartConcurrency-clause, and terminate the concurrency description with endConcurrency-clause.
- **Rule 11:** Start a set of synchronized actions with StartSynchronization-clause, and terminate the synchronization with endSynchronization-clause.
- **Rule 12:** To introduce a condition, start an iteration with While-clause, and terminate it with endWhile-clause.
- **Rule 13:** If NP and VP are preceded by *"No"*, then convert it into *"NP not VP"*.
- **Rule 14:** If a sentence has no verbs (VP), then discard it.
- **Rule 15:** If a prepositional phrase (PP), adjective phrase (ADJP), determiner (DT), or adjective (JJ) precedes the subject of the sentence, then discard that phrases.
- **Rule 16:** If a sentence has a semicolon, then treat it after the semicolon as extra information for the preceding sentence and so discard the sentence after the semicolon.

B. Phase II: Natural Language Processing (NLP)

Several NLP tools are used for processing NL requirements text to avoid manual processing on NL requirements before the generation of UML. "Stanford CoreNLP" [16] is used to perform NLP as it offers outstanding results, as follows:

1. Text Tokenization

Sentence Tokenization (Sentence Splitting): The given input text is split into sentences to determine the borders of sentences and to define the structuring and transformation of the words to ease the further processing, e.g., [The library contains both books and journals.].

Word Tokenization: After sentence tokenization, each sentence goes via word tokenization, e.g., "The library contains both books and journals." is tokenized as [The] [library] [contains] [both] [books] [and] [journals] [.]

2. Parts-of-Speech (POS) Tagging

The tokenized NL text is categorized into various POS Tags, in which all the words in the input text are tagged to the corresponding POS, based on word meaning and the context, in which the word has been used. The Stanford POS tagger or NLTK is used to identify the basic POS tags of each sentence such as verbs, proper nouns, a common noun, adverbs, adjectives, prepositions, etc. based on predefined rules for categorization. For example, the POS analysis of "The library contains both books and journals." is as: "The/DT library/NN contains/VB both/P books/NS and/CN journals/NS. /.".

3. Stemming and Lemmatization

Stemming is the process of removing affixes and suffixes attached to the nouns and verbs, to remove non-word tokens like *s, es,* and *ies*. For example, the verb "contains" is analyzed as "contain+s" and "books" is analyzed as "book+s". This is to get word root form to convert actors and class names from the plural to singular. However, the transformed token may not be a linguistically correct word because of its simplicity. On the other hand, lemmatization always returns the true root form of a word. For lemmatization to work correctly, the original word has to be tagged (i.e., a tag to identify it as a noun, verb, or other parts of speech) so that lemmatization can restore the word to its correct root form.

4. Type Dependencies

To represent the syntactic structure of a sentence and perform the syntactic analysis, the dependency parse can be used to identify the noun and the verb phrases. The Stanford typed dependencies [15] provides a representation of grammatical relations between words that can be understood by people without linguistic expertise. Stanford dependencies (SD) are triplets: name of the relation, governor, and dependent. A type dependency (TD) is used in our approach to the extraction of relevant elements from the NL sentences. Our approach uses Stanford Dependency Parser to generate TDs.

5. Open Information Extraction

The Open Information Extraction (OpenIE) extracts domain relation triples, representing subjects, relations, and their objects, e.g., Customer may be candidate, Customer (subject), may-be (relation), and candidate (object). Open IE refers to the extraction of relation tuples text. It creates a triple (Customer; may-be; candidate), corresponding to the open domain relation may-be (Customer, candidate). The system splits sentences to be as several clauses, which are shortened, producing a set of entailed shorter sentence fragments [15].

C. Phase III: Mapping Rules to Extract UML Concepts

In this phase, the UML class diagram concepts are generated from the NL processing phase output after using the Stanford Corenlp NLP tool to parse the requirements text. To apply the transformation process, a set of heuristics rules are proposed, based on English grammar and UML diagrams constructing rules. Finally, the resulted class diagram components can be drawn manually or using UML drawing tools.

1. Class Identification Rules

- **C-Rule 1:** Extract the common nouns (e.g., Person, User) <NN> tags, and proper nouns (e.g., System, Human) <NNP> tags from the text and map them to classes.

- **C-Rule 2:** If a sentence in (Subject–Verb–Object); Subject <nsubj> and Object <dobj> forms are mapped to candidate classes.
- **C-Rule 3:** If the noun is post-fixed by preposition <IN> tag, then ignore it as a class and map it as a part of a method, e.g., System records details of a candidate. "System" and "candidate" are candidate classes, and "records details" is a method of the class "system".
- **C-Rule 4:** Identify candidate classes from the 'IsA' relationship (inheritance), e.g., EU-Rent is a car rental company. EU-Rent is a subclass of Car Rental Company.
- **O-Rule 5:** For a noun phrase (Noun+Noun), if the first noun is a candidate class, then the second noun is mapped into candidate instances (objects) of that class.

2. Attribute Identification Rules

- **A-Rule 1:** Extract the adjectives in sentences from <JJ> tag, e.g., "sale line item with description, price and total".
- **A-Rule 2:** Extract the sentences like "is-property-of" "identified by" that associated with a candidate class, e.g., "name is-property-of customer", "A student identified by student–id".
- **A-Rule 3:** Extract the possessed nouns (i.e., pre-fixed by's or post-fixed by of) that are associated with a candidate class, e.g., student's address or address of a student.
- **A-Rule 4:** Identify attributes from the 'HasA' relationship (aggregation), e.g., Librarian has a name.
- **A-Rule 5:** Identify attributes from common nouns (e.g., "Class Student attributes are first name, last name, address, etc."), and nouns after possessive pronoun <PP$> tag (e.g., Candidate updates his details.)
- **A-Rule6:** If two nouns appear in a sequence (Noun+ Noun), then if the first Noun is a candidate class, the second Noun is mapped into candidate properties (Class-Property) of this class according to position, e.g., "shoe size" assumed that "size" is a property of class "shoe".
- **A-Rule7:** If two nouns appear in a sequence (Noun+ Noun) including the underscore mark "_" between them, then if the first Noun is mapped to candidate class and the second Noun is mapped into candidate attribute of that class, e.g., "student_name".
- **A-Rule 8:** If a concept has one unique value, then it is an attribute, e.g., "name, date, ID, type, and number".

3. Methods Identification Rules

- **M-Rule 1:** Extract verb phrase that contains lexical verbs (e.g., see, want, act, make) associated with a noun from <VB> tags and map them to candidate methods of this noun as a candidate class.
- **M-Rule 2:** Extract verb phrase that contains action verbs (e.g., calculate, start, enter) associated with a noun from <VB> tags and map them to candidate methods of this noun as a candidate class.
- **M-Rule 3:** Extract verb phrases in the form (Verb <VB>+ Noun <NN>) and map them to candidate methods of the subject of the sentence as a class.
- **M-Rule 4:** If a sentence in (Subject–Verb–Object) or (Noun+Verb+Noun) forms are mapped to a class with Subject and Object as classes sharing the verb as a method and select the sender and receiver classes.

4. Relationship Identification Rules

This section presents the rules for relationship identifications as well as the multiplicity and recursive relationship rules.

Relationship Attributes:

- **RA-Rule:** An adverb <RB> can be identified as an attribute of a relationship type. It may refer to "time, date, place, degree, cause, duration, etc.".

Associations Identification Rules:

A transitive verb can be a candidate for an association relationship. The following are the association rules:

- **AS-Rule 1:** Extract the prepositional phrases from (<VB>+ <IN> or <To>) tags among noun phrases (two classes), such as "has", "next to", "part of", "works for", "contained in", "talk to", "order to", and identify the noun phrases associated with it. These phrases are mapped to association relationships and the associated verb is used as the caption of association.
- **AS-Rule 2:** Extract the verb phrases which is a collection of two verbs <VB>+ <VB> tags, e.g., *savings-checking*, and identify the noun phrases associated with it to identify the participant classes. This verb phrase is mapped to an association relationship.
- **AS-Rule 3:** If a sentence is in the form (Noun+Verb+ Noun) where both nouns are candidate classes, then the verb is an association relationship. Similarly, if the sentence is in the form of (Subject-Predicate-Object), then there is an association between subject and object.
- **AS-Rule 4:** Extract transitive verb (e.g., take, send, buy) from <VB> tags between two candidate classes and map it to association relationship, e.g., System records details of employer. "System" is the subject, "employer" is the object and they are candidate classes. The verb "records" is a transitive verb and it represents an association.

Multiplicity Identification Rules:

- **MR-Rule:** Extract indefinite articles (*a* and *an*) from <DT> tag, plural nouns (prefixed with s) from <NNS> and <NNPS> tags, and cardinal numbers (2 or two) from <CD> tag to identify multiplicity. For example: "library member can borrow books (s)" this is a one-to-many association between a library member and a book.

Participation Types Identification Rules:

- **PT-Rule 1:** Identify the participation type from a noun or a prepositional phrase whose occurrence is singular which gets a minimal and maximum cardinality of 1, e.g., "Each department is managed by only one employee". Here, "department" and "employee" are singular nouns so that the multiplicity is of type one-to-one.
- **PT-Rule 2:** Extract Modal verbs From <MD> tag to identify the participation types, "Can/could" and

"may/might" refer to optional participation, and "must/have" to refer to required participation.

Aggregations Identification Rules:

- **AG-Rule 1**: Extract phrases such as "have", "hold", "possess", "carry", "involve", "imply", "embrace", "contains", "consists of", "comprises of", "is-part-of", "included-in", "belong-to", "divided to", "has part" or "is-made-up-of" between subject and object.
- **AG –Rule 2:** If the concept is verb <VB> and it is equal to one of the phrases mentioned in **AG-Rule 1**, then they are mapped to composition or aggregation, e.g., Library contains Books, Library contains Journals.
- **AG-Rule 3:** If the phrases "comprises, have, include, possess, and contains" are found between object and subject, then a composition relationship exists from object to subject.
- **AG-Rule 4:** If the "is a part of" phrase is found between subject and object, then aggregation (weak) relation occurs from subject to object.
- **AG-Rule 5:** The subject-part is a class in aggregation and the object-part is considered as the subclass.
- **AG –Rule 6:** The collective nouns usually represent aggregation, e.g., a library of books.

Generalization Identification Rules:

- **GE-Rule 1:** Extract phrases such as "is-category-of" or "is-type-of", "is-kind-of" "maybe", "is-a" between subject and object, e.g., "EU-Rent is a car rental company", "Customer may be candidate or employer".
- **GE-Rule 2:** If the concept is verb <VB> and it is equal to one of the phrases mentioned in the previous rule, then they are mapped to generalizations, e.g., Service may be Standard, Silver, or Gold.
- **GE-Rule 3:** If the "maybe" phrase is found between subject and object, represent inheritance relation (direction) from object to subject, "maybe" and "can be" denotes top-down simple inheritance.
- **GE-Rule 4:** If the "is a type of" phrase occurs then inheritance is from subject to object, "is a type of" denotes bottom-up simple inheritance.
- **GE-Rule 5:** The phrase "N1+ is a N2+ and a N3" denotes multiple inheritances between N1, N2, and N3.
- **GE-Rule 6:** The subject-part is considered the main class in generalization, and the object-part, as the subclass.
- **GE-Rule 7:** Extract the objects that have similar attributes or methods and group similar things to an abstract class, e.g., Standard service and Silver service objects both are a type of Service object, therefore these two objects can be specializations of Service class.

Dependency Identification Rules:

- **DE-Rule 1:** Extract the following phrases "require", "depends on", "rely on", "based on", "uses", "follows" between subject and object.
- **DE-Rule 2:** If the concept is verb <VB> and it is equal to one of the phrases mentioned in the previous rule, then they are mapped to a dependency relationship.
- **DE-Rule 3:** If a sentence is in the form (Noun1+Relation + Noun2+ "AND"+ Noun3), where all the three nouns are classes. Then, the relation is between the classes (Noun1, Noun2) and between the classes (Noun1, Noun3).
- **DE-Rule 4:** If a sentence is in the form (Noun1+Relation +Noun2+"AND NOT"+Noun3) where the three nouns are classes, then the relation is between the classes (Noun1, Noun2) and not between the classes (Noun1, Noun3).

Recursive Relationship Identification Rules:

- **RR-Rule:** If a sentence is in the form (Noun1+Verb+ Noun1) when the subject is the same as the object of the sentence, then the relationship is recursive, called self-transitions, e.g., System enters request as a record.

D. Phase IV: Refinement of the Results

This phase deals with filtering and refining the extracted concepts. After applying the rules in Phase III, the extracted information is further processed to generate the class diagram. The output of Phase III produces a set of initial candidates, which may be poorly defined or not related to the problem domain. The collected candidates can be further analyzed to discover details through a set of refinement rules. This leads to more refined results, which can be an input to the production of UML class models. To refine candidate UML elements, the results are matched with the defined following rules.

- **C-Rule 1:** If a candidate class occurred only one time in the text and its frequency is less than 2 %, then ignore it as a class as well as an object name.
- **C-Rule 2:** If a candidate class does not have any attributes, then ignore it as a class.
- **C-Rule 3:** If a candidate class does not participate in any relationship; ignore it as a class.
- **C-Rule 4:** If a candidate class is related to the design, location name, people name, then ignore it as a class.
- **C-Rule 5:** If a concept is an attribute, ignore it as a class.
- **C-Rule 6:** If a candidate class contains information such as relationships, attributes, then that candidate is a class.
- **C-Rule 7:** Remove redundant candidates from the list of the output as they are not required.
- **C-Rule 8:** If two classes reveal the same information, remove a class to avoid redundant classes.
- **C-Rule 9:** If any two candidate classes are related by '*known as', 'same as'* and *'similar to'*, then they are treated as a single class only.
- **C-Rule 10:** If any class is found in both adjective and attribute class list, removes such class from the list.
- **C-Rule 11:** If a class takes a list of values (Boolean, list, etc.), such classes are taken as attributes of a class.
- **C-Rule 12:** Every class should have a specific purpose and be necessary for the system, if not remove it. Manual interference is needed to identify irrelevant classes.

E. Phase V: UML Diagrams Generation

The outcomes of Phase IV produces a set of final UML class elements. The resulted UML diagrams components are obtained to be drawn manually or using UML drawing tools.

IV. EXPERIMENTS AND RESULTS

In this study, two experiments have been conducted to validate the solution by the assessment of UML class components resulting from it. Firstly, the approach has been evaluated by comparing the obtained results using another tool called UMGAR [19], with the same scenario used in this tool to apply the proposed approach to as a case study. Secondly, the proposed approach was compared with some other approaches in terms of the UML class model and elements obtained from each approach. The approach outputs were given and the overall results of the evaluation were discussed. The class diagram generated by UMGAR lacks in identifying attributes of classes and fails to identify aggregation, composition, dependency, and recursive relationships. Moreover, the UMGAR approach does not identify multiplicity and participation types among classes. In contrast, our approach handles the issues in the resulted diagrams. Our approach has a distinct advantage in terms of the graphical models that present the knowledge extracted from the NL text. With our approach, UML class diagrams with a variety of elements can be extracted such as classes, attributes, methods, association, aggregation, composition, dependency, and recursive relationships. The approach also identifies multiplicity and participation types among classes. Further details of the experimental study can be found in [22].

V. CONCLUSIONS

This paper describes a method for generating UML class diagrams from NL requirements using NLP techniques and heuristic rules. The proposed approach uses an NLP tool to read and accomplish the user requirements analysis to facilitate the process of mapping the extracted knowledge into UML class models. Our approach uses NLP techniques and types dependency to parse the NL specifications using a set of pre-defined heuristics rules. Then, a set of rules is proposed to identify the UML concepts from the parsed requirement text. The approach was validated and evaluated using a comparison between its results with the results obtained from another tool using the same scenario of a well-known case study. This work could help engineers in the analysis stage in OO SDLC, hence reducing the cost and time required for manual processes and software developers reducing their time in the design process, specifically in manual generation of UML class diagrams. This approach can be used to generate diagrams by students who just learn about UML diagrams. This approach also can help software analysts to save more time from drawing diagrams and can focus more on developing the software and system.

REFERENCES

[1] O. Dawood and A. Sahraoui, "From Requirements Engineering to UML using Natural Language Processing – Survey Study", *European Journal of Engineering Research and Science*, vol. 2, no. 1, pp. 44-50, 2017.

[2] D. Thakore and R. P. Patki, "Generation of Software Artifacts and Models at Analysis Phase", *International Journal of Engineering Research and Applications (IJERA)*, vol. 2, no. 5, pp. 1624-1630, 2012.

[3] M. Mohanan and P. Samuel, "Natural Language Processing Approach for UML Class Model Generation from Software Requirement Specifications via SBVR", *International Journal on Artificial Intelligence Tools*, vol. 27, no. 06, pp. 1850027–1-1850027–22, 2018.

[4] H. Herchi and W. B. Abdessalem, "From user requirements to UML class diagram", in *International Conference on Computer Related Knowledge (ICCRK' 2012)*, Sousse, Tunisia, 2012.

[5] S. G. MacDonell, K. Min and A. M. Connor, "Autonomous requirements specification processing using natural language processing", *arXiv preprint arXiv: 1407.6099*, 2014.

[6] R. Sharma, P. K. Srivastava and K. K. Biswas, "From Natural Language Requirements to UML Class Diagrams", in *2015 IEEE Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, IEEE, 2015, pp. 1-8.

[7] A. Arellano, E. Carney and M. A. Austin, "Natural language processing of textual requirements", in *The Tenth International Conference on Systems (ICONS 2015)*, Barcelona, Spain, 2015, pp. 93-97.

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

[9] M. Ahmed, W. Butt, I. Ahsan, M. Anwar, M. Latif and F. Azam, "A Novel Natural Language Processing (NLP) Approach to Automatically Generate Conceptual Class Model from Initial Software Requirements", in *International Conference on Information Science and Applications*, Singapore, 2017, pp. 476-484.

[10] R. S. Shweta, and B. Ghoshal, "Automatic Extraction of Structural Model from Semi-Structured Software Requirement Specification", in *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, IEEE, 2018, pp. 543-58.

[11] 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

[12] S. F. Alshareef, A. M Maatuk, T. M., Abdelaziz. "Aspect-Oriented Requirements Engineering: Approaches and Techniques". In *DATA '18*, October 1–2, 2018, ACM. https://doi.org/10.1145/3279996.3280009

[13] T. M., Abdelaziz, A. M Maatuk and F. Rajab. "An Approach to Improvement the Usability in Software Products". In *Int. Journal of Software Engineering & Applications (IJSEA)*, Vol.7, No.2, March 2016. DOI : 10.5121/ijsea.2016.7202 11

[14] D. K. Deeptimahanti and R. Sanyal, "An Innovative Approach for Generating Static UML Models from Natural Language Requirements", *Advances in Software Engineering Communications in Computer and Information Science*, vol. 30, pp. 147–163, 2009.

[15] The Stanford NLP Group, "Stanford Parser", v.1.6, 2007. [Online]. Available: http://nlp.stanford.edu/software/lex-parser.shtml. [Accessed: 3- May- 2020].

[16] The Stanford NLP Group, "CoreNLP", v. 4.0. [Online]. Available: https://stanfordnlp.github.io/CoreNLP/. [Accessed: 7- May- 2020].

[17] D. Jurafsky and J. H. Martin, "Word Classes and Part-of-Speech Tagging," in *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*, Upper Saddle River, NJ: Prentice-Hall, 2004.

[18] M. C. De Marneffe and C. D. Manning, "The Stanford typed dependencies representation", in *the workshop on Cross- Framework and Cross-Domain Parser Evaluation (Coling 2008)*, Association for Computational Linguistics, 2008, pp 1-8.

[19] D. K. Deeptimahanti and M. A. Babar, "An Automated Tool for Generating UML Models from Natural Language Requirements (UMGAR)", in *Inter. Conference on Automated Software Engineering*, IEEE, 2009, pp. 680-682.

[20] A. O Mohammed, Z. A Abdelnabi, A. M. Maatuk, and A. S Abdalla. "An Experimental Study on Detecting Semantic Defects in Object-Oriented Programs using Software Reading Techniques". In *Pro. of ACM Int. Conf. on Engineering & MIS* (ICEMIS '15), 2015, 6 pp. DOI=http://dx.doi.org/10.1145/2832987.2833025

[21] 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.

[22] E. A. Abdelnabi, "Generating UML Diagrams using NLP Processing Techniques and Heuristics Rules", Benghazi University, 2020.