Automatic Transformation of Natural to Unified Modeling Language: A Systematic Review

Sharif Ahmed  
*Boise State University*

Arif Ahmed  
*Boise State University*

Nasir U. Eisty  
*Boise State University*
Automatic Transformation of Natural to Unified Modeling Language: A Systematic Review

Sharif Ahmed  
Dept. of Computer Science  
Boise State University  
Boise, USA  
sharifahmed@u.boisestate.edu

Arif Ahmed  
Dept. of Computer Science  
Boise State University  
Boise, USA  
arifahmed@u.boisestate.edu

Nasir U. Eisty  
Dept. of Computer Science  
Boise State University  
Boise, USA  
nasireisty@boisestate.edu

Abstract—Context: Processing Software Requirement Specifications (SRS) manually takes a much longer time for requirement analysts in software engineering. Researchers have been working on making an automatic approach to ease this task. Most of the existing approaches require some intervention from an analyst or are challenging to use. Some automatic and semi-automatic approaches were developed based on heuristic rules or machine learning algorithms. However, there are various constraints to the existing approaches to UML generation, such as restrictions on ambiguity, length or structure, anaphora, incompleteness, atomicity of input text, requirements of domain ontology, etc.

Objective: This study aims to better understand the effectiveness of existing systems and provide a conceptual framework with further improvement guidelines. Method: We performed a systematic literature review (SLR). We conducted our study selection into two phases and selected 70 papers. We conducted quantitative and qualitative analyses by manually extracting information, cross-checking, and validating our findings. Result: We described the existing approaches and revealed the issues observed in these works. We identified and clustered both the limitations and benefits of selected articles. Conclusion: This research upholds the necessity of a common dataset and evaluation framework to extend the research consistently. It also describes the significance of natural language processing obstacles researchers face. In addition, it creates a path forward for future research.

Index Terms—Requirement Elicitation, Software Engineering, Natural Language Processing, Unified Modeling Language

I. INTRODUCTION

Requirements analysis in software development is an essential and rudimentary task. The quality analysis of requirements elicitation and further developing software work-products such as various diagrams and formatted textual descriptions leads to a successful project, product, or service. We human beings, as an analyst, have the innate power to understand the information in texts contextually regardless of the existence of misspelled words, incorrect grammar, and indirectly stated items. This performance has not been obtained yet in Natural Language Processing (NLP) or Artificial intelligence, and it is a long way to go. But we human beings are also limited by fatigue, drowsiness, distractions, fluctuation in concentrations, which makes us error-prone despite our aforementioned power. So, the machines’ ability to work relentlessly with error and humans’ ability to work perfectly for short periods is not as easy to put on two pans of a weighing scale’s beam and come to a conclusion. Berry et al. [1] suggested that NLP tools for requirements engineering can be an effective approach to activities that are tedious and rigorous.

Unified Modeling Language (UML) is “a general-purpose visual modeling language to specify, visualize, construct, and document the artifacts of a software system”- Rumbaugh et al. [2]. There is no fully working method or tool to generate UML diagrams from the informal natural language (NL) requirements. Most existing approaches have high-level complexity, are computationally costly, or have limitations. Researchers found that earlier approaches require developer intervention for UML diagram creation. Besides, there are a few approaches that are comparatively recent and are fully automated. Still, these approaches are working under some constrained input such as restricted NL or some particular form of texts [3]. A previous work takes informal NL text requirements and generates UML Use Case and Activity Diagrams without any developer's intervention or assistance [4]. Any fully automatic approach or tool can significantly help with requirement elicitation and modeling for software analysts. It can also save the overall cost and time within a software development process.

Many tools and approaches have been proposed and developed to extract information from natural language text to generate UML diagrams. However, natural language processing has many problems such as ambiguity, uncertainty, incompleteness, and incoherence [3]. Existing approaches perform poorly at times because of this ambiguity of NL. On top of that, large requirement documents can produce other issues while processing. Besides, crucial information is sometimes missed, which feeds the verb to develop complete understanding. A software analyst has to figure out such problems and fix these issues using her domain knowledge. Many tools and techniques have been proposed to solve this kind of problem. But in reality, these tools are not being used in Software Development Life-Cycle for many constraints. And most of the tools require frequent interventions of an analyst to finish the process. In addition, the majority of tools are limited to producing class diagrams only [5], [6].

Though there are some new solutions, they still require more rules to increase the domain knowledge, which is another problem [3], [7]. So, our work will provide a conceptual framework showing an overview of using rule-based heuris-
tic approaches and machine learning-based approaches. This framework will help researchers to make decisions while finding a new solution.

II. RELATED WORK

Though there are not many systematic studies related to our research questions, we found a few notable literature reviews that we discuss in this section.

Zaho et al. [8] conducted a robust systematic literature review on NLP for requirements engineering (RE), capturing almost every aspect of RE tasks: detection, extraction, classification, modeling, tracing and relating, search, and retrieval. They covered the breadth of RE, which is valuable for the software research community. In contrast, we give a depth of RE regarding NL to UML automation. As they covered on breadth, it is not expected to have a depth of any of the tasks mentioned above they covered.

Omer et al. [9] performed a survey that addressed techniques and outputs. Their analysis of strengths and weaknesses is limited to the number of diagrams each study generated.

Esra et al. [3] conducted another systematic mapping study on the techniques and approaches of NL to UML Class diagram. However, our analysis is neither limited to class diagrams nor mapping the techniques and approaches. Instead, we dig deeper into the existing approaches, tools, pros, and cons. In addition, we cover what the primary studies addressed, attempted, or overlooked and their success and failure anecdotes from their published works.

III. RESEARCH METHODOLOGY

In this section, we discuss our research methodology for this study. At first, we developed a research protocol following guidelines for systematic literature reviews by Kitchenham [10] to make our research process uncompromising and evident. Then, we followed the following steps: research question formation, study selection, data extraction, and data synthesis. Table I shows the protocol overview that we developed and executed.

A. Research Questions

Our research motivation and objectives led us to the following research questions:

RQ1: What are the existing approaches to automate the UML generation? The answer to this question will recapitulate the existing tools and techniques used by the software engineering community.

RQ2: How effective are the existing approaches? This inquiry will portray opportunities and obstacles of ongoing effort to solve NL to UML transformations.

B. Searching Strategy

We mainly focused following digital libraries for our study:
- ACM Digital Library (ACM)
- IEEE Xplore Digital Library (Xplore)
- Science Direct Digital Library (Elsevier)
- CiteSeerX
- Google Scholar

Our study comprises two phases. At first, we searched on aforementioned digital libraries using the query string, (((usecase OR “use case” OR class ) AND diagram ) OR UML ) AND generation

and performed the second phase of our search on abstracts only. We chose only ACM Digital Library in this phase as it retrieved 673 articles, the least number of articles than other libraries. We manually selected 30 articles after screening the title and abstract from these retrieved articles. Our searching period was Sep 2021 to Dec 2021, and the publishing years of papers were from 1994 to 2021.
A. RQ1: What are the existing approaches to automate the UML generation?

- Year-wise distribution of published studies
- Statistics of approaches to convert NL to UML
- Statistics of automation to convert NL to UML
- Relationships among UML components resolved
- Distribution of final/intermediate outputs acquired
- Usage of technologies or tools to solve the problems
- Analysis of dataset, metric, and evaluation techniques
- Analysis of strength and limitation of existing solutions

Table IV provides quantitative insights. Section 4.2 provides 16 facets with both quantitative and qualitative insights.

IV. RESULTS

We discuss our findings according to the research questions in this section.

A. RQ1: What are the existing approaches to automate the UML generation?

At first, we grouped all the approaches found in our primary selected papers into the following categories: Heuristic Rule-Based, Machine Learning, Automatic, Semi-automatic, and Hybrid. Fig. 2 shows the approaches and automation we found in our analysis. But as we moved forward, we figured out that many research works combined these approaches. So, we refined our cluster represented in Fig. 2. We found that most of the work was carried out by applying the heuristic rules, and a few other works used machine learning techniques. In addition, some of the papers obtained significant output by using both heuristic rules and machine learning techniques.

1) UMLs Generated: Table III shows the UML diagrams that existing studies covered. Table III presents the most common UML outputs such as usecase, sequence, and activity. Among them, the class diagram is studied the most.

2) Technologies Used: We have manually extracted the technologies used from each of the papers from our primary studies. Table IV shows the list of technologies used. Researchers used a variety of NLP techniques in their studies. We observed significant use of the Stanford CoreNLP library. Besides, we also found the use of OpenNLP and SharpNLP. Stanford’s POS tagger, Brill, TreeTagger, dependency parser,
Fig. 2. Approaches to solve NL to UML transformation

Here, hr: Heuristic rules based approach, ml: Machine learning approach

TABLE III
OUTPUTS OBTAINED IN STUDIES

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>class diagram</td>
<td>[4]–[6], [11]–[39]</td>
</tr>
<tr>
<td>usecase diagram</td>
<td>[7], [13], [40]–[45]</td>
</tr>
<tr>
<td>object diagram</td>
<td>[30], [46]–[49]</td>
</tr>
<tr>
<td>processed sentences</td>
<td>[5], [59]</td>
</tr>
<tr>
<td>sequence diagram</td>
<td>[11], [14], [20], [43]</td>
</tr>
<tr>
<td>comparison</td>
<td>[5], [59]</td>
</tr>
<tr>
<td>code</td>
<td>[14], [17]</td>
</tr>
<tr>
<td>use case</td>
<td>[37], [57]</td>
</tr>
<tr>
<td>test case</td>
<td>[35], [57]</td>
</tr>
<tr>
<td>processed srs</td>
<td>[58], [59]</td>
</tr>
<tr>
<td>activity diagram</td>
<td>[7], [60]</td>
</tr>
<tr>
<td>meta model</td>
<td>[53], [61]</td>
</tr>
<tr>
<td>Other Outputs:</td>
<td></td>
</tr>
<tr>
<td>traceability</td>
<td></td>
</tr>
<tr>
<td>graph</td>
<td></td>
</tr>
<tr>
<td>sbvr</td>
<td></td>
</tr>
<tr>
<td>er-diagram</td>
<td></td>
</tr>
<tr>
<td>collaboration diagram</td>
<td></td>
</tr>
<tr>
<td>test cases</td>
<td></td>
</tr>
<tr>
<td>natural language</td>
<td></td>
</tr>
<tr>
<td>feature diagram</td>
<td></td>
</tr>
<tr>
<td>gui prototype</td>
<td></td>
</tr>
</tbody>
</table>

TABLE IV
TECHNOLOGY USED IN EXISTING STUDIES

<table>
<thead>
<tr>
<th>Technology</th>
<th>Studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlp</td>
<td>[5], [8], [11], [12], [14], [15], [17], [19], [51], [53], [55], [58], [60], [62], [63], [65], [68]</td>
</tr>
<tr>
<td>rule</td>
<td>[11], [13], [21], [22], [24], [29], [133], [144]</td>
</tr>
<tr>
<td>pos tagger</td>
<td>[1], [5], [9], [15], [17], [25], [28], [40], [42], [135], [64]</td>
</tr>
<tr>
<td>parse</td>
<td>[1], [13], [21], [23], [29], [33], [59], [64]</td>
</tr>
<tr>
<td>stanford corenlp</td>
<td>[11], [13], [15], [17], [43], [45], [54], [62], [68]</td>
</tr>
<tr>
<td>ontology</td>
<td>[11], [15], [24], [27], [29], [32], [65]</td>
</tr>
<tr>
<td>parser</td>
<td>[1], [21], [29], [43], [54], [68]</td>
</tr>
<tr>
<td>wordnet</td>
<td>[5], [11], [13], [27], [51], [53], [59]</td>
</tr>
<tr>
<td>tree</td>
<td>[43], [11], [21], [25], [54], [61]</td>
</tr>
<tr>
<td>gui</td>
<td>[15], [21], [23], [29], [54]</td>
</tr>
<tr>
<td>open nlp</td>
<td>[2], [21], [29]</td>
</tr>
<tr>
<td>dependency</td>
<td>[1], [22], [44], [45]</td>
</tr>
<tr>
<td>sbvr</td>
<td>[13], [15], [24], [28]</td>
</tr>
<tr>
<td>graph</td>
<td>[24], [44], [62]</td>
</tr>
<tr>
<td>ml</td>
<td>[20], [36], [60]</td>
</tr>
<tr>
<td>ocl</td>
<td>[20], [24], [67]</td>
</tr>
<tr>
<td>traceability</td>
<td>[25], [60]</td>
</tr>
<tr>
<td>Other Technologies:</td>
<td></td>
</tr>
<tr>
<td>sleuth</td>
<td></td>
</tr>
<tr>
<td>treantagged</td>
<td></td>
</tr>
<tr>
<td>bayes</td>
<td></td>
</tr>
<tr>
<td>featureide</td>
<td></td>
</tr>
<tr>
<td>gkps</td>
<td></td>
</tr>
<tr>
<td>javapar</td>
<td></td>
</tr>
<tr>
<td>reasoning</td>
<td></td>
</tr>
<tr>
<td>sharpnel</td>
<td></td>
</tr>
<tr>
<td>spacey</td>
<td></td>
</tr>
<tr>
<td>spider</td>
<td></td>
</tr>
<tr>
<td>verinnet</td>
<td></td>
</tr>
<tr>
<td>semnet</td>
<td></td>
</tr>
<tr>
<td>lotilla</td>
<td></td>
</tr>
<tr>
<td>rtool</td>
<td></td>
</tr>
</tbody>
</table>

POS parser, and Stanford’s parser were most commonly used for Parts of Speech (POS) tagging. WordNet was most common for resolving ambiguity, but WSD and VerbNet were also used. A set of heuristic rules were defined for heuristic-rule-based solutions. We also found several works using the domain ontology technique.

3) Preprocessing techniques applied: We found that all the preprocessing treatments for NL are almost similar in nature. The first task in preprocessing is to split the text into sentences. Then the sentences are tokenized using either stemming or lemmatization. Surprisingly we found several works that applied both techniques together [5], [9], [72]. In stemming, the output can be meaningless words, affecting the use of tagger and parser. Then POS tagger and POS parser are used for identifying NP, VB, grammatical structures, etc. Then a set of rules is applied to identify actors, use cases, relationships, etc. But while normalizing the NL text automatically, there is a high chance of losing information [13]. So, the decision and treatment selection in preprocessing phase affects the overall performance.

B. RQ2: How effective are the existing approaches?

In this study, one of our objectives was to find “to what extent problem(s) were solved” and “what limitations were observed” from our primary studies. To do so, we extracted both pros and cons from each study. Then, we framed our findings into a conceptual framework comprising sixteen different facets in Fig. 3. Finally, we describe how far researchers could solve existing problems and what problems they could not solve while translating NL to UML in this section.

1) Ambiguity: The requirement document contains text where comprehension or meaning extracted by readers may vary based on readers’ perspectives. In particular, requirement analysts can resolve the ambiguity from text with their domain knowledge. But when it comes to NLP, it becomes much harder to capture the correct meaning of a word by resolving ambiguity. Any misinterpretation of a word may produce defects that can affect the overall context [72]. Some studies solved ambiguities but mostly resolved lexical ambiguity. In NLP, syntactic or semantic ambiguities are also common problems. But, existing approaches could not resolve such ambiguities completely [57], [38], [49], [55], [56], [58], [59].
A study stated six kinds of ambiguity in software requirements: lexical, syntactic, semantic, pragmatic, vagueness, and language error [73]. However, in their work, they only focused on lexical, syntactic, and language errors only [59]. We found some manual glossaries, machine learning, and ontology-based approaches to reduce ambiguity from the SRS. In addition, we found some tools for identifying and eliminating ambiguities either fully or partially. Tools are: WSD, QuaARS, ARM, RESI, SREE, NAI, SR-Elicitor, and NL2OCL [74].

2) Semantic Correctness: We found several studies resolved semantic incorrectness by using WordNet. But, in a study, they considered frequency counts while using WordNet RESI, SREE, NAI, SR-Elicitor, and NL2OCL [74].

3) Language: We did not find any study processing Non-English requirement text. We only found a work where the requirement text was translated from French into English [54].

4) Heuristic Rule: Studies using heuristic rules have shown promising work progress. However, most of them addressed the necessity of more heuristic rules for better performance [4], [7].

5) Relation Resolution: Several relationships connect the UML components. For instance, the association relationship connects each actor and the use case. Meryem et al. [42] identified the actors first and then discovered the use cases using the linkage from the actors. And for more than one property associated with a concept or candidate, it is considered as a class; otherwise, an attribute [5]. Prepositions are emphasized for identifying relationships and associations [17]. Verb phrases are used to determine inheritance, association, composition, and aggregation relations to extract actors or classes; association and composition relations are used for identifying actions, operations, or methods [43]. Another type of relationship is semantic relation comprising vertical and horizontal relationships. Vertical relations include broader, part-of, or instance-of. Horizontal relations are similarity or relatedness [5]. SBVR, which has Object Oriented information, easily retrieves the association, multiplicity, aggregations, generalizations, and instances. A few others used Named-Entity-Recognition, Stanford Open Information Extraction, Ontology, WordNet, and Dependency Parser to solve these relations.

6) Text Restriction: We found a few papers that mentioned different constraints on NL text [40], [52]. We also investigated the structure of the NL text used for transforming into UML. Some studies put restrictions on:

- Text length: No study explicitly mentioned the text length except Sandeep et al. [40]. On average there were 10 sentences per story which are comparatively shorter. In reality, the original user stories are much more lengthy and lack clarity.
- Sentence length: Some approaches only used 5-15 sentences with an average sentence length of 10 words in their user requirement document. Some studies observed that long sentences having more than one N (Nouns) or VB (Verbs) can create disturbingly long use cases [40].
- Grammatical structure: The constraint to use modal verbs [52] or active sentences [13], [27], [44], [77] only in requirement text imposes additional restrictions.
- Controlled language: A few approaches were carried out using controlled NL instead of NL [15], [55], [63].

7) Formal Expression Extraction: In formal expression extraction, an expression must be consistent with the formal representation of the system. Dependency graph construction based on grammatical structure and abstract syntax trees can show similarities. But, synonyms applied within a sentence cause major problem [59]. For example, “CPU” and “Processor” can be used within the same sentence. In that case, Word Sense Disambiguation (WSD) is a good choice to resolve such issues.

8) Dataset: Most of the primary studies applied their models to one or more NL software requirement text(s) (aka case-study) to demonstrate the models’ performance. We found input NL case-study appeared more than twice are: Library [12], [15], [16], [24], [31], [36], [64], ATM [37], [45], [60], [76], Elevator [45], [46], [60], Banking [22], [25], [26], [36], Home [53], [70], Arena [45], [60]. However, some studies didn’t even address their input case-study [6], [14], [15], [40], [66], [76]. The number of input case-study used in the experiment ranges from 0 to 7.

Meryem et al. [42] worked with NLP techniques along with OWL ontology and MAT, then simulated the performance, including accuracy, recall, precision on classifying actor, use case, and relationships on a dataset that is obsolete and inaccessible. There are a few papers that mentioned the dataset they used. But we did not find all of them available publicly. We also found a few studies using significantly small and poor datasets [15], [35].

9) Anaphora Resolution: This is one of the most common and challenging problems in NLP. A few studies solved the only basic or pronominal issues using JavaRAP by replacing proper nouns with their correct noun form [13], [49]. For example: “John found the love of his life” where ‘his’ refers to ‘John’.

10) Incompleteness: Requirement text with correct grammatical structure may have an information gap if the document is written considering readers understandability. Any human reader having field expertise can comprehend such incompleteness. But, it is difficult to capture such information gaps using incorporated NLP techniques [65], [72].

11) Atomicity: Another issue comes with a lack of atomicity which appears when the writer keeps the text unambiguous.
by providing all the information to make the text complete but addresses more than one thing. In NLP, if we can split a sentence into smaller sentences, then it doesn’t go along with atomicity. A few studies solved this problem by splitting a sentence into smaller sentences. But, for complex sentences, it required human intervention [72].

12) Misclassification: Researchers found a high level of misclassification for classes and attributes for using simple heuristic rules [16], [54]. Abdelkareem et al. [14] normalized the text before POS tagging then used Stanford NLP POS Tree. But, they found that the Stanford PoS tree misses Sub (Subject) by misclassifying VB (Verb)/NP (Noun Phrase).

13) Manual Intervention: We found studies producing outcomes with and without manual intervention. A few studies require it mandatorily, some of them require it to some extent [1], [7], [11], [30].

14) SRS Traceability: After identifying class, attributes, and operations (UML components), identified UML components are checked in a backward fashion to see whether the UML components are meaningful and working or not. This iterative process provides meaningful class diagrams [25]–[27].

15) Proof of Concepts: Many of the studies provided various software, IDE plugins, or other forms of deliverables, which are essential for the research community to believe their demonstrations and feasibility [12], [14], [17], [47], [63], [71].

16) Evaluation: The most common metrics found in our primary studies are based on counting proper classification or extraction of UML components from NL such as Precision, Recall Accuracy, F-measure, Enumeration, Over-specification, Over-generalization False Negative, False Positive. A few other less common metrics found are Completeness, Ambiguity, Conceptual Density of Dataset, Computational time, Weighted Average (in addition to enumeration), Verifiability, Consistency, and Atomicity. Some studies just contrasted their implementation with others as an evaluation such as Relationship, Output Type, Automation.

V. DISCUSSION

We have found most papers using a rule-based heuristic approach which indicates the common trend for solving, but machine learning approaches are also noticeable in recent years. Each of the studies had taken an almost similar treatment for NL preprocessing using different libraries.

We observed many patterns in the existing research works and discovered trends and relationships. As described in section [V] we identified sixteen facets of the solutions and limitations of the existing approaches. We found the class diagram as the most used UML diagram. Implementing other UMLs like use case or usecase diagrams does require additional tasks and effort. We also found a comparatively low number of research mentioning their proof of concepts, but in some cases, these were neither publicly accessible nor found on the web. The primary concern for most of the studies was the ambiguity of NL. Few works have significantly mentioned this problem, its limitations, and its treatment.

However, most implementations had constraints such as satisfying a specific grammatical structure, using a domain ontology, sentence length, and absence of ambiguity or anaphora. Some also failed to identify UML components or detect relationships among them in some cases. Many evaluation metrics and datasets emerged in this study. Claims from evaluations are not comparable in many cases because of differences in metrics or datasets. Moreover, the case studies used lack more extensive or diverse descriptions.

VI. THREATS TO VALIDITY

We did not use SpringerLink and Scopus due to not having access to these digital libraries within the period of this study. The SLR was independently executed by the first two authors and reviewed by the third author to minimize personal bias. We might have some selection bias in phase-1. We analyzed the most frequent words from the titles of selected 40 papers from phase-1. Using these most frequent words as query terms, we again searched the same digital libraries. Thus, the later searching phase minimizes query term selection bias as we constructed it from statistics instead of our choice. Among them, ACM digital library retrieved the least number of relevant papers. So we considered only ACM for the second phase and manually selected the most relevant 30 papers for our second phase study selection. The reduced number of relevant papers in the second phase indicates that our two-phased search strategy was effective and had good evidence coverage.

VII. CONCLUSION

This systematic literature review framed research works that generated UML components from NL. We focused on approaches used, their pros and cons, and metrics. In addition, we identified the contributions and limitations of these works in this study. We found several heuristic-rule-based solutions. It can be a potential avenue to build a robust framework using machine learning techniques by exploiting these heuristic rules. In many research fields, several common standard datasets exist that help researchers extend their research work
rapidly. We believe that establishing a benchmark dataset and designing metrics (quantitative and qualitative) for evaluating NL to UML transformation will help the community.

This systematic literature review will help the researchers of the NLP and RE fields. This review pointed out major NL problems that researchers are trying to resolve. Furthermore, we have analyzed and conceptually framed the existing approaches and their pros and cons. Hence, our work will help both the researchers and developers working on the automation of NL to UML transformation.

Fig. 3. Conceptual Framework

REFERENCES

[29] H. Herchi and W. Abdessalem, “From user requirements to uml class diagram,” Int. Conf. on Comp. Related Knowledge (ICCKR), Tunisia. arxiv, 2012.