Hindawi
Scientific Programming
Volume 2021, Article ID 6616564, 20 pages
https://doi.org/10.1155/2021/6616564

Research Article

An Intelligent Analytics Approach to Minimize Complexity in Ambiguous Software Requirements

Fariha Ashfaq,[1] **Imran Sarwar Bajwa**,[1] **Rafaqut Kazmi**,[1] **Akmal Khan**,[1] **and Muhammad Ilyas**[2]

[1]*Department of Computer Science, The Islamia University of Bahawalpur, Bahawalpur 63100, Pakistan*
[2]*Department of Computer Science, University of Malakand, Chakdara, Pakistan*

Correspondence should be addressed to Imran Sarwar Bajwa; imran.sarwar@iub.edu.pk

Received 16 November 2020; Revised 13 January 2021; Accepted 8 March 2021; Published 23 March 2021

Academic Editor: Fabrizio Riguzzi

An inconsistent and ambiguous Software Requirement Specification (SRS) document results in an erroneous/failed software project. Hence, it is a serious challenge to handle and process complex and ambiguous requirements. Most of the literature work focuses on detection and resolution of ambiguity in software requirements. Also, there is no standardized way to write unambiguous and consistent requirements. The goal of this research was to generate an ambiguity-less SRS document. This paper presents a new approach to write ambiguity-less requirements. Furthermore, we design a framework for Natural Language (NL) to Controlled Natural Language (CNL) (such as Semantic Business Vocabulary and Rules (SBVR)) transition and develop a prototype. The prototype also generates Resource Description Framework (RDF) representation. The SBVR has a shared meaning concept that minimizes ambiguity, and RDF representation is supported by query language such as SPARQL Protocol and RDF Query Language (SPARQL). The proposed approach can help software engineers to translate NL requirements into a format that is understandable by all stakeholders and also is machine processable. The results of our prototype are encouraging, exhibiting the efficient performance of our developed prototype in terms of usability and correctness.

1. Introduction

Requirements engineering (RE) has been dealing with similar issues since the beginning in software engineering. RE is a common phase in business process development as well as software development for user requirement specification. Typically, the four stages of RE in the software development process include requirement elicitation, analysis, requirement validation, and specification. Here, requirement elicitation and requirement analysis phases are significant to ensure complete, consistent, and unambiguous requirement specifications. The end product of the RE phase is the SRS document. The SRS document becomes the base of the later stages [1] of the software development process. The initial phases of the software development process have more impact on the software quality as compared to later phases [2]. The SRS document maintains user end story in descriptive form using NL [1], because NL is the most convenient way to communicate while gathering requirements in business process development as well as in software development. Almost 79% of requirement specification documents are found in "Real Natural Language" [3]. NL is expressive, universal, flexible, widespread, and above all understandable for all stakeholders [4]. Along with the key features, NL is inherently ambiguous [5] and has extensively been recognized as a challenge [6].

In The Ambiguity Handbook, Berry et al. [7] define ambiguity as, "A requirement is ambiguous if it admits multiple interpretations despite the reader's knowledge of the requirement engineering context." There are 39 types of ambiguity, vagueness, or generality [7]. NL such as English is ambiguous owing to its informal sentence construction. There exists an average of 23 meanings for the 500 most used English words [8]. Ambiguity is more intractable than other requirement defects and, thus, results in more frequent misunderstandings [9].

If the SRS document contains ambiguous, incomplete, or contradictory requirements, it results in erroneous software. According to S. McConnell, more than 50% of corporate software projects do not fulfill the expectations [10]. CHAOS report of Standish Group 2015 states that 52% of software exceed budget as well as time or does not meet the basic requirements, while 19% of projects are a complete failure [11]. All such failures are due to misunderstandings while interpreting stakeholder's ambiguous requirements. The SRS document is considered unambiguous only if there is only one possible meaning for each software requirement [12]. The SRS document contains requirements understood and agreed upon by all stakeholders [13].

Inconsistency and ambiguity in the SRS document propagate to the next phases of software development [14], such as software modeling; affect the accuracy of the end software product; and results in more manual effort and high-cost budgets. Hence, the functional requirements gathered and quoted in the SRS document must be complete, consistent, and clear, i.e., unambiguous.

A possible solution to handle ambiguity can be the exercise of a mathematical formal logic illustration instead of NL to represent user requirements. However, the use of formal logic is not only a complex task. A wrongly written formal logic will be difficult to handle, and it will create a serious problem in later stages of software development. Furthermore, stakeholders are usually not able to understand mathematical logic. Hence, this solution does not seem feasible.

Another possible way to deal with the above-discussed ambiguity problem is the use of CNL [15]. It can work as a bridge between NL and formal representation. Since Requirement Analysis is based on communication and the analyst's experience, it can be modeled up to a certain limit. This limit gives birth to controlled language. If the document is written in CNL, it will be feasible for a development team to use a simpler and less costly linguistic tool. Several CNL could be found in literature such as ACE [16], PENG [17], CPL [18], Formalized-English [19], SBVR, etc. We intend to apply an SBVR-based CNL to write stakeholder's requirements and generate an SRS document. Such software requirements will be syntactically unambiguous as well as semantically consistent. In this paper, a novel approach is presented to soften the complex requirements by representing the requirements in standardized forms such as SBVR and RDF. By "soften the complex requirement," we mean to rewrite a complex requirement in such a standardized form so that the ambiguity caused by the complexity of the sentence can be avoided.

2. Literature Review

Two classes' approach is found in RE to handle ambiguity [13]. The first-class reactive approaches include ambiguity detection and resolution. Such class deals tend to base on cases where ambiguity is already present in the SRS document's text. The second class is proactive, i.e., ambiguity prevention/avoidance deals with ambiguity in the first place, i.e., the requirement elicitation phase. It detects ambiguity in requirements and demands an explanation from the user. The five major approaches in which research work is being done to deal with both classes of ambiguity include checklist-based inspection approach; style guides; knowledge-based approach; heuristics-based approach; and controlled language [20]. Bano [6] and Sandhu and Sikka [18] have conducted a deep survey on ambiguity handling approaches.

In a literature review [6], the author argued that there is more focus on ambiguity detection, whereas ambiguity avoidance and resolution have been largely neglected in empirical work. Also, more work is done on syntactic and semantic ambiguities than other types. In literature, different methods and techniques are proposed to tackle ambiguity in the SRS document [6, 20].

Friedrich et al. [21] present an automatic approach to produce Business Process Model and Notation models from the NL text. The purpose is to minimize ambiguities and time. The researchers combined existing tools from NLP and improved them using an appropriate anaphora resolution mechanism. The generated SVBR specifications are validated and verified using the SBVR2UML approach proposed by [22]. This tool translates SBVR specifications documented in the SRS document to Unified Modeling Language (UML) class diagram. The technique extracts Object-Oriented information from SBVR-SRS and maps it to a class model.

RESI [23] is an ontology-based "common sense engine" supporting analysts by providing a dialog-system to make suggestions and inquires for an ambiguous, faulty, or inaccurate specification. RESI works in four steps, and utilizes various ontologies such as ResearchCyc, WordNet, ConceptNet, and YAGO to discover problems and to provide common-sense solutions. Mich and Garigliano [24] have proposed a scheme to define indices of structural and semantic ambiguity. Researchers have investigated the application of defined indices for the system called LOLITA. It uses the knowledge base (a kind of conceptual graph) and output of the parsing phase to calculate the feasibility of these indices. LOLITA is designed to prevent ambiguity. LOLITA detects structural and semantic ambiguity.

A tool SREE [25] is designed to detect instances of potential ambiguity in NL requirement specifications and reports them back to the user to take appropriate action. SREE uses a lexical analyzer, searching for instances of only precise words in the SREE's database. The tool uses guiding rules that help to write less ambiguous NL SRS documents. The provided guiding rules can also serve as an inspection checklist to find ambiguities in requirement specifications. SREE only detects coordination ambiguity and semantic ambiguity. Yang et al. [26] constructed a machine-learning-based antecedent classifier, to detect anaphoric ambiguities. The antecedent classifier trains itself on a set of heuristics and human judgments. The output is then used to spot nocuous ambiguity. The used classifier alerts the requirements analyst about the risk of misinterpretation. A Comparative Analysis is mentioned in Table 1.

Instead of generating user understandable format of the SRS document, research focused on NL to model/prototype generation. All of the abovementioned tools motivate us to

Table 1: Comparative analysis of work done on ambiguity.

Sr #	Source	Data type	Identified ambiguities	Proposed approach	Technologies/ models/methods/ algorithms used	Ambiguity avoidance	Ambiguity detection	Ambiguity resolution
1	Kamsties et al. [27]	NL: English	Lexical, Polysemy, systematic Polysemy, Referential Discourse, Domain	Checklist-based inspection approach UML-based heuristics-based approach	UML Foundation package, SCR metamodel	✘	✓	✘
2	Mich and Garigliano [24]	NL: English	Semantic Syntactic	Knowledge base, NLP, indices	LOLITA	✘	✓	✘
3	Ashfa and Imran Sarwar Bajwa [11]	NL: English office time management system	Syntactic Semantic	SBVR, controlled NL	Stanford POS tagger v3.0rule-based bottom-up parser	✓	✘	✘
4	Ferrari et al. [28]	NL: EnglishOutbreak Management Functional Requirements, issued in 2005 by the Public Health Information Network (PHIN) of the Centers for Disease Control and Prevention (CDC)	Pragmatic	Knowledge-based approach	Least-cost path search	✘	✓	✘
5	Friedrich et al. [21]	NL:47 text-model pairs from industry and textbooks	Referential anaphora resolution	NLP-based	Stanford Parser, FrameNet, WordNet, World Model, the anaphora resolution algorithm	✘	✓	✘
6	Kaiya and Saeki [29]	NL: Japanese, software music player	Semantic	Ontology-Based	Spreadsheet, a diagram editor with macro processing	✘	✘	✘
7	Gleich et al. [30]	NL:data set consisting of approximately50 German and 50 English sentences	Lexical Syntactic Semantic Pragmatic	Automated ambiguity detection, NLP-based	Unix-based grep-like technique	✘	✓	✘
8	Al-Harbi et al. [31]	NL: English200 NL questions for the university domain from various universities' websites	Lexical. semantic (nouns-based ambiguity)	Context knowledge and concepts ontology, shallow NL processing based	Rule-based chunker, rule-based shallow semantic analyzer, semantic role labeling method, WordNet Domains	✘	✘	✓
9	Verma and Beg [32]	NL: English	Syntactic Incompleteness	NLP-based	Shift-reduce style parser, maximum entropy parser, Penn Tree Bank	✘	✘	✓
10	Gill et al. [33]	NL: English	Lexical Syntactic Semantic	NLP-based	NL Processing, Word Sense Disambiguation, Text Mining Language Perceptions, Human Judgment, Contextual Knowledge	✓	✘	✘

Table 1: Continued.

Sr #	Source	Data type	Identified ambiguities	Proposed approach	Technologies/ models/methods/ algorithms used	Ambiguity avoidance	Ambiguity detection	Ambiguity resolution
11	Massey et al. [34]	NL: English (23 paragraphs from HITECH Act, 45 CFR Subtitle A, § 170.302)	Lexical, Syntactic Semantic, Vagueness Incompleteness, Referential	Manual guide-based	Case Study Ambiguity Taxonomy	✗	✓	✗
12	Sandhu and Sikka [18]	NL: English	Lexical Syntactic Semantic Pragmatic	Knowledge-based approach	Rule-based framework	✗	✓	✗
13	Bhatia et al. [5]	NL: English	Pragmatic Semantic Vagueness and Generality, Language error	Knowledge-based ontology-based	Ontology-based framework	✗	✓	✗
14	Sabriye and Zainon [1]	NL: EnglishOpen-source software requirements specification documents	Syntactic Syntax	NLP-based	Stanford POS tagger, rule-based ambiguity detector	✗	✓	✗
15	Ali et al. [35]	NL: EnglishServices Proz: workforce software	Syntactic Syntax	NLP-based ontology-based	IEEE 830 Template, RUP template, Pragmatic Quality Model (PQM), and Perspective-based Reading (PBR)	✗	✓	✓
16	Popescu et al. [36]	NL SRS	Semantic	Controlled NLP-based	Dowser parser	✗	✓	✗

design an approach that can assist analysts in creating ambiguity-less SRS documents from English. The above-mentioned tool tends to deal with a certain type of ambiguity, thus others remain a concern. Along with syntactic and semantic ambiguities, there is a need to work on other types of ambiguities as well.

To minimize the effect of ambiguity and informality of NL in the SRS document, the best approach is to use some sort of combined approach. CNL (a subset of NL) such as Attempto [37] and SBVR [38] minimize formality as well as help to generate ambiguity-less SRS document. For business process modeling, SBVR has emerged as a vital tool for capturing software requirements. It is an efficient way to get machine processable and unambiguous rules written in a formal pattern from NL.

Most of the work is based on the reactive approach; previous solutions tend to base on cases where ambiguity is already present in the text. Most of the works deal with ambiguity detection. There is a huge gap in research to find a way to avoid ambiguity in the first place. Moreover, the proposed solution deals with a certain type of ambiguities. There is a lack of a standard approach that successfully deals with all types of ambiguities. Bajwa et al. [39] have proposed an automated approach, NL2SBVR. NL2SBVR uses the UML class model as a business domain to map NL to SBVR vocabulary. It translates NL specifications to formal SBVR rules. The SBVR provides a set of logical formulations that can help in the semantic analysis of English language text. NL2SBVR will give efficient results in less time and with lesser errors. The proposed approach achieved an average accuracy of 87.33%. The approach lacks a method to specify the meaning for entities in the dataset.

Ramzan et al. [40] developed a tool for NL to SBVR model transformation using the Sitra transformation engine. To accomplish the transformation, the concept of English metamodel is introduced. The transformation is based on SBVR 1.0 document. The proposed approach achieved an average recall of 83.22%, average precision of 87.13%, and average F-value of 85.14. Danenas et al. [41] propose the M2M conversion. Authors extract data from UML case diagrams using Text rumblings. Preprocessing is performed on the extracted result of Text rumblings. In the end, the SBVR model is generated. A common understanding of things is the prerequisite for nonautomated parts and, thus, can cause ambiguity.

Njonko and El Abed [42] transform NL business rules into SBVR Rules using the NLP framework. SBVR rules are then transformed into Executable Models. The evaluation of the proposed approach is missing. Hence, the success factor of the proposed approach is unknown. Siqueira et al. [43] combine the Behavior-Driven Development approach and SBVR to refine requirements through iterations. Chittimalli and Anand [44] check the inconsistencies among SBVR rules using Satisfiability modulo theories. Arnold and Rahm [45] propose an approach to identify the semantic similarity between concepts from Wikipedia articles.

We propose a proactive approach to handle all the parameters that actively play a role in raising ambiguities in

NL software requirements. Such a proactive approach focuses on representing software requirements, represented in NL, in a CNL such as SBVR that is based on formal logic. It provides a shared meanings concept. The output of our proposed approach will not only be ambiguity-less and understandable by all stakeholders but also machine processable at the same time. Table 2 presents a comparative analysis between some existing tools for ambiguity prevention and the proposed approach.

2.1. Semantic Business Vocabulary and Rules (SBVR). The SBVR was first released by Object Management Group (OMG) in 2008. The SBVR can define clear, syntactically, and semantically unambiguous, meaning-centric business vocabulary and rules not only in business but also in the software industry. The SBVR has mathematical foundations (first-order logic), which make it easy to be processed by a machine. The SBVR's English-like syntax makes it easy to understand for all stakeholders.

The SBVR has the same expressive power as standard ontological languages [24]. The produced SBVR XML schema makes the interchange of information easy among organizations and software tools. In the end, the use of the SBVR will reduce the overall cost of the software development process. The SBVR 1.4 [38] is the latest standard by OMG. The SBVR vocabulary (Concepts, Fact Types) and rules constitutes a standardized SBVR representation [38].

2.2. How the SBVR Can Help to Eliminate Ambiguity? The SBVR vocabulary is not only a set of definitions of terms (concepts) and other representations but is an organized set of interconnected concepts. For a particular business domain, the SBVR vocabulary defines each term with one exact meaning in a given context, thus eliminating the chance of ambiguity. Such definitions are formal enough to be used by other software tools and informal enough to be understood and used by all stakeholders.

2.2.1. Removing Lexical Ambiguity. In the SBVR, an expression represents a concept. Each expression/wording is uniquely associated with one meaning (concept) in a given context. Thus, one concept–one meaning association eliminates the chance of lexical ambiguity. The definition of a concept incorporates delimiting (a concept must hold) and implied (a concept must not hold) characteristics, eliminating the chances of homonymy lexical ambiguity. Along with the definition, each concept has a definite description along with examples and notes, hence removing polysemy ambiguity. By following the SBVR rules to create a vocabulary, the result will be well-defined concepts that cannot be taken in multiple senses within the community, thereby eliminating all types of lexical ambiguities.

2.2.2. Removing Syntactic Ambiguity. The SBVR includes "Verb Concept Wording" to formally specify syntactic representations of concepts and rules of any domain in NL. Such a feature makes the SBVR well-suited for describing syntactically ambiguity-less software requirements. The SBVR identifies concepts along with the grammatical roles they play in a certain situation. Such a feature removes the analytical ambiguity from a given requirement. For each verb concept, the SBVR identifies related categorization, classification, characterization, and situational roles. Such identification will minimize attachment ambiguity. Furthermore, the SBVR uses logical formulation such as logical operations and, thus, eliminates the coordination ambiguity.

2.2.3. Removing Semantic Ambiguity. The SBVR provides semantic formulations to make English statements controlled and semantically ambiguity-less. Such a semantic formulation includes atomic formulation, instantiate formulation, modal formulation, logical formulation, quantification, objectification, and nominalizations. The use of logical formulation such as quantification eliminates the scope of ambiguity. The SBVR incorporates "Reference Scheme." Such schemes serve as a link for noun phrases and prepositions to their corresponding concepts and, thus, eliminate Referential Ambiguity.

2.2.4. Removing Pragmatic Ambiguity. The SBVR avoids pragmatic ambiguity by identifying associative fact types. The binary fact type is a typical example of the Associative Fact Type. In the example "The car conveys the parts," there is a binary association between the car and parts concepts. This association is one-to-many as the "parts" concept is plural. In the conceptual modeling of the SBVR, Associative Fact Types are mapped to associations.

3. Proposed Methodology

The main objective of this research is to prevent the SRS document from inducting ambiguity in the first place. To achieve this, the proposed methodology comprises four main phases, i.e., preliminary investigation and analysis, proposed framework design and implementation, data collection and experimental evaluation, and research findings and conclusion.

3.1. Preliminary Investigation and Analysis. In the first phase of the research, existing literature associated with the study has been studied. The purpose is to acquire knowledge related to RE techniques—the use of NL in SRS documents and the ambiguity caused by NL. Possible solutions in the literature to handle ambiguity in NL have been studied.

3.2. Proposed Approach Design and Implementation. Based on preliminary investigation and analysis, an ambiguity prevention approach is proposed to avoid ambiguity. A prototype is developed to assess the anticipated approach.

3.3. Data Collection and Experimental Evaluation. In the third phase, the data for the evaluation of the approach will

Table 2: A comparative analysis of existing tool with the presented approach.

Feature Support	Korner and Brumm [23]	Mich and Garigliano [24]	Ashfa, and Imran Sarwar Bajwa [11]	Al-Harbi et al. [31]	Verma and Beg [32]	Our proposed approach
Approach used	Knowledge-based to ontology	Knowledge base	Controlled language	Ontology	NLP	Controlled language
Technologies/ Models/Methods/ algorithms/ Approach used	RESI Stanford parser, ConceptNet, WordNet	LOLITA Indices	SR-Elicitor SBVR, Stanford POS tagger rule-based bottom-up parser	Shift-reduce style parser, maximum entropy parser, Penn Tree Bank	NL Processing, Word Sense Disambiguation	SBVR, Stanford POS tagger, the rule-based bottom-up parser Wikipedia
Syntactic ambiguity	✘	✓	✓	✘	✓	✓
Lexical Ambiguity	✓	✘	✓	✓	✘	✓
Semantic Ambiguity	Scope	✓	✓	✓	✘	✓
Pragmatic Ambiguity	✘	✘	✘	✘	✘	✓
User interaction	High	Medium	Low	Low	Low	Medium

be collected from open-source SRS documents. Once data collection is complete, it will be analyzed using the developed prototype.

Evaluation is conducted to validate the research findings and to measure the performance and accuracy of the proposed approach. Evaluation is conducted in two steps, i.e., performance evaluation and output document verification.

3.3.1. Performance Evaluation. To evaluate the performance of the system, an evaluation methodology is used proposed by Hirschman and Thompson [46]. The performance evaluation methodology is based on three aspects, i.e., Criterion, Measure, and F-measure. To evaluate the results of the developed system, each element (Noun concepts, Verb concepts, and SBVR rules) of the system's generated output was compared with the expert's opinion (N*expert*) (sample solution). The element is classified as correct (N*correct*), incorrect (*Nincorrect*), or missing (N*missing*).

3.3.2. The Output Document Verification. The resultant output document is stored in the XML format. To verify XML schema, the XML file is parsed to validation service W3C RDF/XML validation service [47]. If the document is correctly formatted, the W3C web service replicates the document into Triples and the graph format.

3.4. Research Findings and Conclusion. In the end, conclusion, scope, limitations, and further work improvements are being listed.

4. Design of the Proposed Approach

This section describes the design of the prototype tool of semiautomated NL-Requirements–SBVR-Rules transformation. The prototype comprises six basic stages, as shown in Figure 1.

4.1. Input Documents. The proposed approach takes two inputs:

(a) An English text document (.txt file): the input is taken as a plain text file containing English written software requirements. It is assumed that the given English text is grammatically correct

(b) Software Artifact: the system will accept a software artifact/model such as the UML class model (.ecore file) to validate the SBVR vocabulary

(c) Wikipedia: the system will use Wikipedia's assistance to associate meanings to the validated SBVR vocabulary.

4.2. Stage-1: Parse NL Text of Software Requirements. The parsing phase includes lexical and syntax analysis of the input software requirements. This phase involves processing units (ordered in a pipeline) to analyze complex English sentences. The parsing steps are as follows:

Lexical processing: this phase takes a plain text file containing an English SRS document as an input. Lexical processing has further subphases:

Tokenization: the first subphase of lexical processing is the tokenization of English text software requirements. The text is sliced into tokens. A single token is in fact a "sequence of characters" with collective meaning. During the tokenization phase, a sentence is sliced into token instances. Each such instance is known as lexemes.

Delimiters are used to identify lexemes: based on our requirement, each sentence of the English text is tokenized using StringTokenizer (str) and the output is stored as an array-list.

An example of input text: "A designer may work on many projects." The generated tokenized output is: [A] [designer] [may] [work] [on] [many] [projects] [.]. Such tokenized data will be used in syntactic analysis.

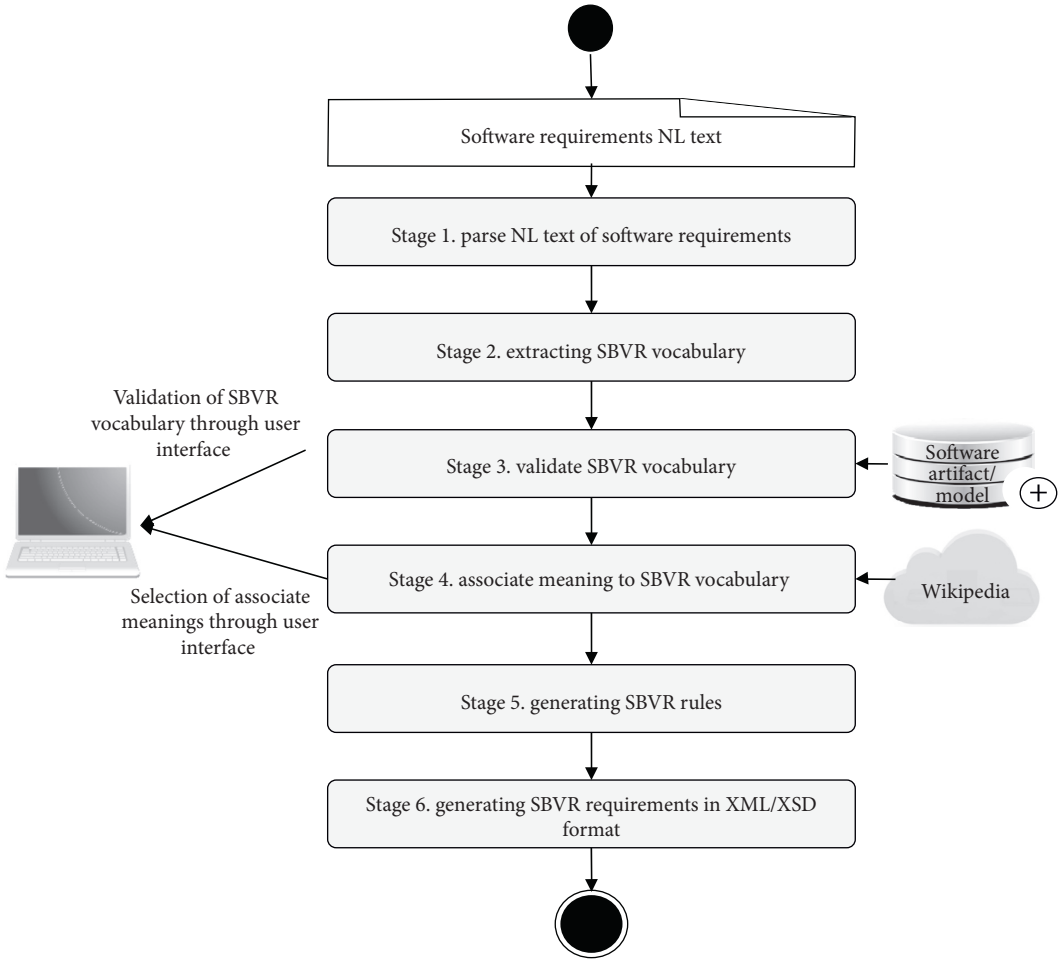

Figure 1: Algorithm of the proposed methodology.

Sentence splitting: the splitter spots the ends of a statement usually by a period ".". It can also be used to split the sentence itself. We used Java Split() method to split the strings into substrings. Such substrings are stored in a String array. The Spit() method splits the string using the regular expression(RE) given inside the parameters while calling the method. Such split sentences will be used in later stages. Each identified sentence is stored separately in an array-list.

Parts-of-speech (POS) tagging: a POS Tagger reads text as an input in some language (in our case English language), tokenizes the input, and assigns POS tags to each token, such as nouns, adjectives, etc. In this subphase, basic POS tags are identified using Stanford POS tagger v3.0 [48]. The tagger is Java implemented software of the log-linear part-of-speech taggers. The overall accuracy of the Stanford POS tagger is 96.86% [48]. Penn Treebank is being used for tokenization. The tagger takes requirements as the input. The given input is transformed into tokens. After the completion of the tokenization stage, the tagger assigns POS tags to each identified token. The generated result is stored in an array-list for further processing.

An example of the tagger generated output:

[A/DT] [customer/NN] [can/MD] [have/VB] [two/CD] [Credit/NNP] [Cards/NNP] [./.].

Syntactic processing: in this phase, the input is syntactically analyzed to produce POS tags. The above subphase is also performing syntactically analysis and generating POS tags using the Stanford POS tagger. The output generated by the Stanford POS tagger includes tagging, parse tree, and universal dependencies. Such output formats are not enough to perform semantic analysis and to generate the SBVR vocabulary and Rules. The SBVR generated rules have an English sentence format. Such format requires tags identification to generate an English sentence. The desired structure is

Subject + Verb + Object

Furthermore, the Stanford tokenizer is unable to identify action verbs (active voice and passive voice), models, and demonstratives. Stanford tokenizer is unable to convert English written quantifications to equivalent decimal numbers. Such identifications are crucial for SBVR vocabulary and Rules generation. To

generate an SBVR rule, we need to do a deep semantic analysis. Along with the Stanford POS tagger's generated tags, our proposed methodology requires additional information such as:

Logical formulation: logical formulations comprise logical operators such as AND, OR, NOT, implies, etc. Table 3 shows the possible tokens mapping to logical formulations:

Quantification: quantification specifies the scope of a subject. Possible quantifications can be identified from tokens by following mapping rules mentioned in Table 4:

Modal formulation: modal formulation depicts the relationship linking the meaning of another logic formulation and possible words or to acceptable words. Modal formulation specifies the seriousness of a constraint. Table 5 shows possible tokens mapping to the modal formulation:

To extract such information, based on methodology requirement, an enhanced version of a Stanford parser, the rule-based, bottom-up parser proposed by Bajwa et al. [49], is used to analyze input at the next level. The rule-based, bottom-up parser is based on English grammar rules. Such a parser takes tagged tokens as the input. Such tokens were generated in the previous subphase in Section 4.2. The enhanced version of the rule-based parser is capable of accepting more than one requirement at a time. Such a feature was not available in the original version. The parser overcomes the abovementioned limitations of the Stanford parser and will extract all necessary information required to perform semantic analysis. The generated output will be used to perform semantic analysis and to generate SBVR vocabulary and rules. The algorithm used by the rule-based parser to identify each tag/chunk is shown in Figure 2.

Sample input for the rule-based parser is [A/DT user/NN can/MD have/VB two/CD Debit/NNP Cards/NNP./.]

The generated output for the given input is shown in Figure 3. The generated output is saved in an array for further use.

Semantic Analysis: The phase identifies the meanings and inference of a certain string. A program is considered semantically reliable if all its variables, functions, classes, etc., must be appropriately defined; expressions and variables are following the model. This is a crucial phase of analysis as a semantically incorrect requirement will eventually produce an incorrect system. The proposed semantic analyzer analyzes the tags generated in syntactic processing and assigns corresponding roles to each tag. The process followed is shown in Figure 4.

The user uses this semantic table (see Table 6) to verify the roles of each token/chunk and hence the semantics of requirements. A single table can display the whole set of input requirements. Such identified roles assist in identifying the SBVR vocabulary in the later phase. All identified roles are stored in an array-list. Figure 5 exemplifies the output generated by the Semantic Parser.

Table 3: Tokens to logical formulations mapping.

Tokens	Logical formulation
"not," "no"	*negation* ($\neg a$)
"that," "and"	*conjunction* ($a \wedge b$)
"or"	*disjunction* ($a \vee b$)
"imply," "suggest," "if," "infer"	*implication* ($a \Longrightarrow b$)

Table 4: Tokens to quantification mapping.

Tokens	Quantification
"more than," "greater than"	*at least n*
"less than"	*at most n*
token "equal to" or a positive statement	*exactly n*

Table 5: Tokens to quantification mapping.

Tokens	Modal formulation
Model verbs ("can", "may")	Structural requirement
Model verbs ("should," "must")	Behavioral requirement
Verb concept ("have to")	Behavioral requirement

4.3. Stage-2: Extracting the SBVR Vocabulary. This stage identifies primary SBVR vocabulary elements from the English input that was preprocessed in the previous stage 2. To write SBVR rules, we need to identify SBVR vocabulary elements. The steps to extract SBVR elements include

(i) Extracting Unitary Noun Concept
(ii) Extracting Individual Noun Concept
(iii) Extracting Individual Verb Concepts
(iv) Extracting Binary Verb Concepts
(v) Extracting Characteristic/Unary Verb Concepts
(vi) Extracting Unitary Verb Concepts
(vii) Extracting Associative Verb Concept
(viii) Extracting Partitive Verb Concept
(ix) Extracting Quantification
(x) Extracting Categorization

The final step is to create facts. An SBVR fact is a basic building block of the formal representation [38]. A fact type is based on identified verb concepts. A list of noun concepts and verb concepts is available in the SBVR vocabulary array-list. A fact type is generated by mapping such concepts. Atomic formalization is used to map the input requirement to an appropriate fact type. The generated list of the SBVR vocabulary consists of concepts and fact types. Such a vocabulary will be used as a reference throughout the generation of SBVR rules. A list of possibly extracted vocabulary elements from English text is shown in Table 7.

4.4. Stage-3: Validation of the SBVR Vocabulary. This phase validates that the elements of the SBVR vocabulary are consistent with the domain. The validation phase takes the list of the extracted SBVR vocabulary in the form of an array-list that comes from the previous phase and the second one is

(1.1) Identify is, are, am, was, were as "subject in state"
(1.2) Identify has, have, had as "subject in possession"
(1.3) Identify EX (existential there) as "there"
(1.4) Identify WDT (Wh-determiner) as "that"
(1.5) Identify CD (cardinal number) and DT (determiner) as "quantifications"
    (1.5.1) Convert english text number to decimal number ('one' and 'a' –> 1)
(1.6) Identify CC (coordinating conjunction) as "conjunction"
    (1.6.1) Convert to equivalent symbol ('and' –> $$, 'or' –> ||, 'not' –> !, ',–? $$)
(1.7) Identify IN and TO: (preposition or subordinating conjunction) as "preposition"
(1.8) Identify NN (noun, singular or mass), NNP (proper noun, singular), NNPS (proper noun, plural,
    NNS (noun, plural), POS (possessive ending), and quantification
    (1.8.1) Identify "subject" along with a conjunction
    (1.8.2) Identify "object" along with conjunction and preposition
(1.9) Identify "helping-verb" and "action-verb" using VB, MD, VBZ, VBD, VBN, VBDN, VBP

Figure 2: The rule-based parser algorithm.

tag	Chunk
asb	1 user
hvb	can
avb	have
ob	2 debit cards
./.	eos|

Figure 3: An example of syntax analysis.

a software artifact/model such as a UML class model. To input such a model, the following two options can be used:

(1) The user provides a UML class model of the respective business domain that will be used as a software artifact for the validation process.
(2) The approach provides a repository of a large number of UML class models from various business domains. The user will perform a manual selection of relevant software artifacts/model from the available repository.
(3) Once the selection of a UML class model has been made, the selected UML class model is parsed using the parseEcore parser to extract the metadata.
(4) UML parser performs extraction on the resultant content. The extraction process includes the following steps:
    (a) Extract classes
    (b) Extract attributes
    (c) Extract operations list
    (d) Extract associations
    (e) Extract generalization

Once the extraction process is complete, nodes are created and the parseEcoreparser represents the extracted features of the UML model in a hierarchical tree format, as shown in Figure 6.

The user will manually perform the mapping of extracted metadata to SBVR elements to validate the SBVR vocabulary, as shown in Table 8.

4.5. Stage-4: Associate Meanings. The key success of the SBVR is that it defines every vocabulary element; hence, eliminate ambiguity in terms and statements. This phase associates meanings to the validated SBVR vocabulary and eliminates "single word–multiple senses" ambiguity. Before the conversion of English text to SBVR rules, the meaning is associated with the input SBVR vocabulary to ensure that the resultant SBVR rules will be semantically associated with the relevant business domain. The system will look for associate meaning for each requirement in the Knowledge Base.

This phase receives two inputs; the SBVR vocabulary and Knowledge Base. Wikipedia is used as a knowledge base. Wikipedia is a reliable and general-purpose Knowledge Base accommodating all possible sets of meanings from different domains. Wikipedia is a fast and lightweight application; requires only an Internet connection; is easily accessible and has no specific memory requirement. A list of synonyms is also available on Wikipedia; hence, it eliminates "multiple words–one sense" ambiguity.

(1) To associate meanings, the user has to select an SBVR vocabulary element.
(2) The proposed system uses a Wikipedia parser to extract a list of possible meanings from Wikipedia. All possible scenarios for a selected SBVR vocabulary element will be displayed from Wikipedia using an interface, and the final selection will be left to the domain expert/analyst.
(3) The user can select the associated meaning for all SBVR vocabulary elements.
(4) The SBVR vocabulary is updated by adding associated meanings. Such associated meanings are added to the SBVR vocabulary for future assistance. The association of meanings will be of great assistance to the Analysis and Design team. There will be no ambiguity in the SBVR vocabulary in terms of
    (i) Single word–multiple senses
    (ii) Multiple words–one sense

Figure 7 shows added associate meanings in the SBVR vocabulary.

(1) The input is an array from the previous syntactic processing phase.
(2) Semantic analyzer tokenized the input using java stringtokenizer() method.
(3) Each token/chunk (an english word) perform a specific role within a sentence such as "subject", "object", "adverb", "preposition" etc. it is very important to identify such roles to understand the semantics of a requirement written as english text. The semantic analyzer identifies roles as:
(4) Parser generates higher-order logic-based semantic representation. This will identify status of "subject" and "object" and label type as "state", "possession" and "active".
(5) In the end parser performs atomic formulation. Atomic formulation includes the role binding for a particular role of the verb concept that is the basis of the atomic formulation. Atomic formulations are labeled as "is a", "attribute of", "akind of", "belongs to", "quantification" and "relation."
(6) End of each requirement is marked with "EOS" determiner. It helps to distinguish among different requirements.
(7) The identified roles are displayed in the semantic table.

Figure 4: Steps followed for semantic analysis.

Table 6: An example of a semantic table.

Tag	Syntax	Type
Ssb	Subject	State
|	OR	
Avb	Verb	
For, of, in, on, from, at, etc.	Preposition	
Integer	Quantification	
./.	Eos	

#	Chunk	Syntax	Quant	Logical	Type	Prep	EOS
1	Customer	Subject	1		Active		
2	Can	H.Verb					
3	Have	A.Verb					
4	Credit cards	Object	2				
5	Visa card	Object	1	AND			True

Figure 5: Algorithm for identifying subject part state sentence.

4.6. Stage-5: Generating SBVR Rules. Once the SBVR vocabulary is validated and associated with related meanings, the System is ready to generate SBVR rules. In this phase, SBVR rules are generated by using a rule-based parser. Such rules will be used to get the SBVR-based requirements specifications. Rule base parser contains a set of rules that maps SBVR elements with the SBVR vocabulary. The SBVR rule generator follows three steps to rule generation, which include extracting the SBVR Rule type, applying Semantic Formulation, and finally applying structured English notation.

Extracting the SBVR rule type: in this phase, each requirement is categorized either as a structural or a behavioral requirement. Such classification will be used to generate corresponding advice or behavioral rule. Following rules are applied to classify a requirement type.

Extracting advices: The requirements written in English language having auxiliary verbs, such as "can," "may," etc., are identified and classified as advice. For example, sentences representing state, e.g., "NBP is a bank," or possession, e.g., "Bank cab has two cashiers," can be categorized as advice. Moreover, the English written requirements using general action verbs, such as consists, composed, equipped, etc., are also classified as structural requirements.

Extracting behavioral requirements: the English written requirements having auxiliary verbs such as "*should,*" "*must*" are identified and classified as a behavioral rule. Furthermore, the requirements having an action verb can be classified as a behavioral rule, e.g., "*Cardholder provide a valid password.*"

Applying semantic formulation: a set of such formulations are exercised to each fact type to generate an SBVR rule. The SBVR version 1.5 proposes two basic semantic formulations. These include Logical Formulation and Projections. Logic Formulation is further categorized as Atomic Formulations, Instantiation Formulations, Modal Formulations, Logical Operations, Quantifications, Objectification, Projecting Formulations, and Nominalizations of Propositions and Questions. Here, we are using the following three formulations concerning the context of the scope of the proposed research.

Logical formulation: an SBVR rule may consist of various Fact Types via logical operators such as AND, OR, NOT, implies, etc. Table 9 shows the possible tokens mapping to logical formulations:

Quantification: quantification specifies the scope of a concept. Possible quantifications can be identified from tokens by following mapping rules mentioned in Table 10:

Modal formulation: such formulation stipulates the weight of a constraint. Table 11 shows possible tokens mapping to the modal formulation:

The steps followed by the rule generator are as follows:

(1) Identify new sentence
(2) Identify numeric value
(3) Identify each
(4) Identify object types
(5) Identify individual concept
(6) Identify the verb concept

Table 7: Possible extracted vocabulary elements from English text.

Tags from English text requirement document	SBVR elements
Proper nouns	Individual concepts
Common nouns appearing in the subject part	Noun concepts or general concept
Common nouns appearing in the object part	Object type
Auxiliary and action verbs	Verb concepts
Auxiliary verbs and noun concepts	Fact types
Common nouns in the object part	Unary fact type: object type/individual concept without an action verb
Common nouns in the object part + auxiliary Action verbs	Unary fact type with an action verb
Common nouns in the object part/proper nouns + auxiliary and action verbs + common nouns in the object part	binary fact type: object type/individual concept + verb + object type
Characteristic, adjectives or attributes, possessed nouns	Characteristic: Is-property-of fact type
Indefinite articles, plural nouns, and cardinal numbers	Quantification with the respective noun concept
Associative or pragmatic relations	Associative fact types
"Is-part-of," or "included-in," or "belong-to"	Partitive fact types
"is-category-of," or "is-type-of," "is-kind-of	Categorization fact types

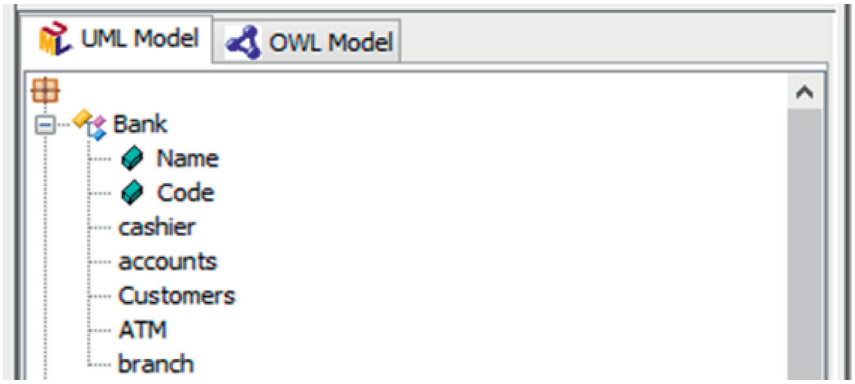

Figure 6: UML parser output.

Table 8: Equivalence between SBVR elements and UML model components.

SBVR elements	UML components
Individual concepts	Instances
Noun concepts or general concept	Classes
Object type	Classes
Verb concepts	Methods, operations
Fact types	Associations and generalizations
Unary fact type: object type/individual concept without an action verb	Attributes of a class
Unary fact type with an action verb	Unary relationship
binary fact type: object type/individual concept + verb + object type	Associations and generalizations
Characteristic: Is-property-of fact type	Attributes of class
Quantification with the respective noun concept	Multiplicity
Associative fact types	Caption of association
Partitive fact types	Generalizations
Categorization fact types	Aggregations

Visa card
Concept type: role
General concept type: noun concept
Concept type: individual concept
Logical term: prepaid cards
Description: (pay from a cash account that has no checkwriting privileges)

Figure 7: An example of associated meaning to the SBVR vocabulary.

The algorithm followed to identify a new sentence is given in Figure 8.

4.6.1. Applying the SBVR Notation. The last phase is to apply an SBVR notation to generate SBVR-based requirements specifications. Such a format contains unambiguous and

Table 9: Tokens to logical formulations mapping.

Tokens	Logical formulation
"not," "no"	*negation* ($\vdash a$)
"that," "and"	*conjunction* ($a \wedge b$)
"or"	*disjunction* ($a \vee b$)
"imply," "suggest," "if," "infer"	*implication* ($a \Longrightarrow b$)

Table 10: Tokens to quantification mapping.

Tokens	Quantification
"more than," "greater than"	at least n
"less than"	at most n
token "equal to" or a positive statement	exactly n

Table 11: Tokens to modal formulation mapping.

Tokens	Modal formulation
Model verbs ("can," "may")	Structural requirement
Model verbs ("should," "must")	Behavioral requirement
Verb concept ("have to")	Behavioral requirement

constant specifications. The end document will be an XML format file that uses the SBVR XMI XSD as its XML Schema. Such documents will be easy for a machine to process. The proposed approach supports SBVR Structured English. To apply Structured English:

(i) The noun concepts are underlined, e.g., **card**

(ii) The verb concepts are written as italic, e.g., *can, has*

(iii) The keywords are bolded, i.e., SBVR keywords, e.g., **each, at least, at most, obligatory**, etc.

(iv) The individual concepts are double-underlined, e.g., black cat

For example: "A person's nationality should be British." will be translated as per SBVR rule as: "It is obligatory that each a person's nationality should be 'British'."

4.7. Stage-6: The Output. The output is saved and exported in an XML format. The file contains the SBVR vocabulary and SBVR rules. To verify the XML schema, the XML file can be parsed to any validation service such asW3C RDF Validation Service [47]. Such a file is an interchangeable, platform-independent, easy machine process document, providing a clear, unambiguous, consistent, and complete SRS document.

5. Implementation of the Approach

To better understand the stages, let us observe the interaction of the components during their defined scenarios of NL to SBVR Rules transformation.

(i) The system takes two inputs—user requirements and the UML model for validation. Requirements are written by the user either in a text file (A1 from Figure 9) or directly on the SBVR editor pane (A2 from Figure 9). Such requirements are used to extract the SBVR vocabulary. The evaluations of the proposed approach use a text file to input requirements. The UML model was selected from the available UML model repository. The user selects a domain-specific UML model to perform mapping from the UML model to SBVR vocabulary set.

(ii) As the "Generate SBVR" button is clicked, the Rule-Based Parser [24] extracts the SBVR vocabulary from the NL text software requirements. The Parser identifies the concepts along with the concept type and its general concept (A3 from Figure 9). Along with the concepts, the parser also identifies the fact type of related concepts.

(iii) The UML model is used to validate the generated SBVR vocabulary. This validation procedure was performed manually. The user mapped the identified SBVR vocabulary elements to the UML model items.

(iv) The user can add semantics using the Wikipedia parser. The Wikipedia parser extracts the list of meanings from the Internet related to the SBVR vocabulary (A4 from Figure 9). The user can select the domain-specific meaning (A5 from Figure 9). Such a selected meaning is associated with the corresponding vocabulary concept (A6 from Figure 9).

(v) The Rule-Based Parser generates the SBVR Rules. Such SBVR rules include Structural rules and Behavioral rules (A7 from Figure 9). Once the SBVR rules are generated, the SBVR Structured Notation is applied to the rules. The SBVR notation makes the rules more readable.

XML Parser generates a RDF/XML schema (A8 from Figure 9). Such a file provides a consistent, interchangeable, platform-independent schema. The generated RDF schema is also validated through the RDF validator available at http://www.w3.org. An example is shown in Figures 10 and 11.

6. Results

The proposed approach of NL to SBVR Rules extraction was evaluated using seven sets of requirements ($T1$, $T2$,. . ., $T7$). Each set consists of 50 randomly selected requirements. The selected requirements were syntactically valid. The requirement set was parsed to prototype. A sample of extracted Noun concepts, Verb concepts, Fact types, and generated SBVR rules are presented in Table 12. The sampling requirement set T is a subset of the requirements set $T1$. $T1$ is related to the domain "car rental services." For 5 sample input sentences of the requirement set T, our designed prototype has extracted 8 Noun Concepts and 5 Verb Concepts. These Noun Concepts and Verb Concepts are processed to create 5 Fact types. Once the necessary extraction is performed, Semantic formulation and SBVR notation are applied on Fact type to generate SBVR Rules.

(1) Identify new sentence
  (1.1) Identify the negative sentence
    (1.1.1) Mark sentence as "negative"
  (1.2) Identify POS tag as "JJR" and "than" in the sentence
    (1.2.1) Identify "more", "greater" and "most"
      (1.2.1.1) If sentence is negative: add "at most" in rule
      (1.2.1.2) Else add "at least" in rule
    (1.2.2) Identify "less", "smaller", "least"
      (1.2.2.1) If sentence is negative: add "at least" in rule
      (1.2.2.2) Else add "at most" in rule
  (1.3) Identify "maximum" in sentence
    (1.3.1) If sentence is negative: add "at most" in rule
    (1.3.2) Else add "at least" in rule
  (1.4) Identify "minimum" in sentence
    (1.4.1) If sentence is negative: add "at least" in rule
    (1.4.2) Else add "at most" in rule
  (1.5) Identify "equal to" in sentence
    (1.5.1) If sentence is negative: add "not exactly" in rule
    (1.5.2) Else add "exactly" in rule
  (1.6) Identify "PRP" in POS tags
    (1.6.1) Add corresponding vocabulary element in rule
  (1.7 Identify "exactly" in sentence
    (1.7.1) If sentence is negative: add "not exactly" in rule
    (1.7.2) Else add "exactly" in rule
  (1.8) Identify "RB" in POS tags
    (1.8.1) If corresponding vocabulary element belongs to {more, greater, less, exactly, least, most}
      then add new sentence
    (1.8.2) Else add corresponding vocabulary element in rule
(2) End

Figure 8: SBVR rule generation.

Table 13 actually specifies the extracted elements for each requirement set ($T1$, $T2$, . . ., $T7$) used for the construction of SBVR rules. First of all, using the prototype, we identify Noun Concepts and Verb Concepts. Using these concepts, the prototype generates Fact types. Such fact types are further processed to generate SBVR rules.

The Requirement set $T1$ is processed by the prototype and 89 Noun Concepts and 48 Verb Concepts are extracted. These concepts are processed to construct 50 Fact types. The prototype finally generates 50 Rules. The prototype has extracted a total of 237 elements for $T1$. The process is followed for each requirement set $T1$, $T2$, . . ., $T7$. A sample of such extractions for rule generation is depicted in Table 12.

After seven iterations of seven case studies, the total number of extracted elements such as Noun Concept are 624 and Verb Concept are 332. The prototype has constructed the 346 Fact Type. The prototype has successfully generated 350 Rules for seven case studies consisting of 350 requirements collectively.

7. Analysis and Discussion

To evaluate the results of the approach, we prepared a sample test case requirements set ($T1$, $T2$,. . ., $T7$). An expert manually evaluated the requirements sets to create sample data N_{expert}, as shown in Table 14. N_{expert} comprises extracted Noun Concepts, Verb Concepts, constructed fact type, and generated Rules. The purpose is to validate the prototype-generated output. The sum of the output generated by the prototype is labeled as N_{total}. N_{correct} is the element correctly identified by the prototype. $N_{\text{incorrect}}$ is the element that is identified by the prototype but is incorrect when compared with N_{expert}. N_{missing} is the element that the prototype is unable to identify or process. A comparison is performed in Table 14 between the extracted concepts, constructed fact types, generated Rules by the prototype, and with manually evaluated requirements; N_{expert}.

The expert (N_{expert}) has identified 94 Noun Concepts and 50 Verb Concepts from requirements set $T1$. The expert (N_{expert}) then constructed 50 fact types based on extracted Noun and Verb Concepts. In the end, the expert (N_{expert}) has generated 50 SBVR Rules. In comparison, the prototype has identified 89 Noun Concepts out of which 86 are correct (N_{correct}), 3 are incorrect (N_{correct}), and 5 are missing (N_{missing}). The prototype then extracted 48 (N_{total}) Verb Concepts, from which 46 are correct (N_{correct}) and 2 are incorrect (N_{incorrec}). Two Verb Concepts are missing in the list. Afterward, the prototype has constructed 50 Fact types (N_{total}). Two Fact types are incorrect ($N_{\text{incorrect}}$). In the end, the SBVR Rules are constructed using Fact types. A total of 237 elements are identified out of which 230 are correctly identified, 7 are incorrect identifications, while 7 elements are missing. The process is repeated for each requirement set ($T1$, $T2$,. . ., $T7$), as shown in Table 14.

The expert successfully extracted 1752 elements for seven case studies. According to the used evaluation methodology, Table 14 shows 1652 identified elements, of which 1595 are correct, 57 are incorrect, and 100 are missing SBVR elements. For each requirement set, the rules generation rate is 100%. The Fact Type identification results are also very satisfactory. Out of 350 total

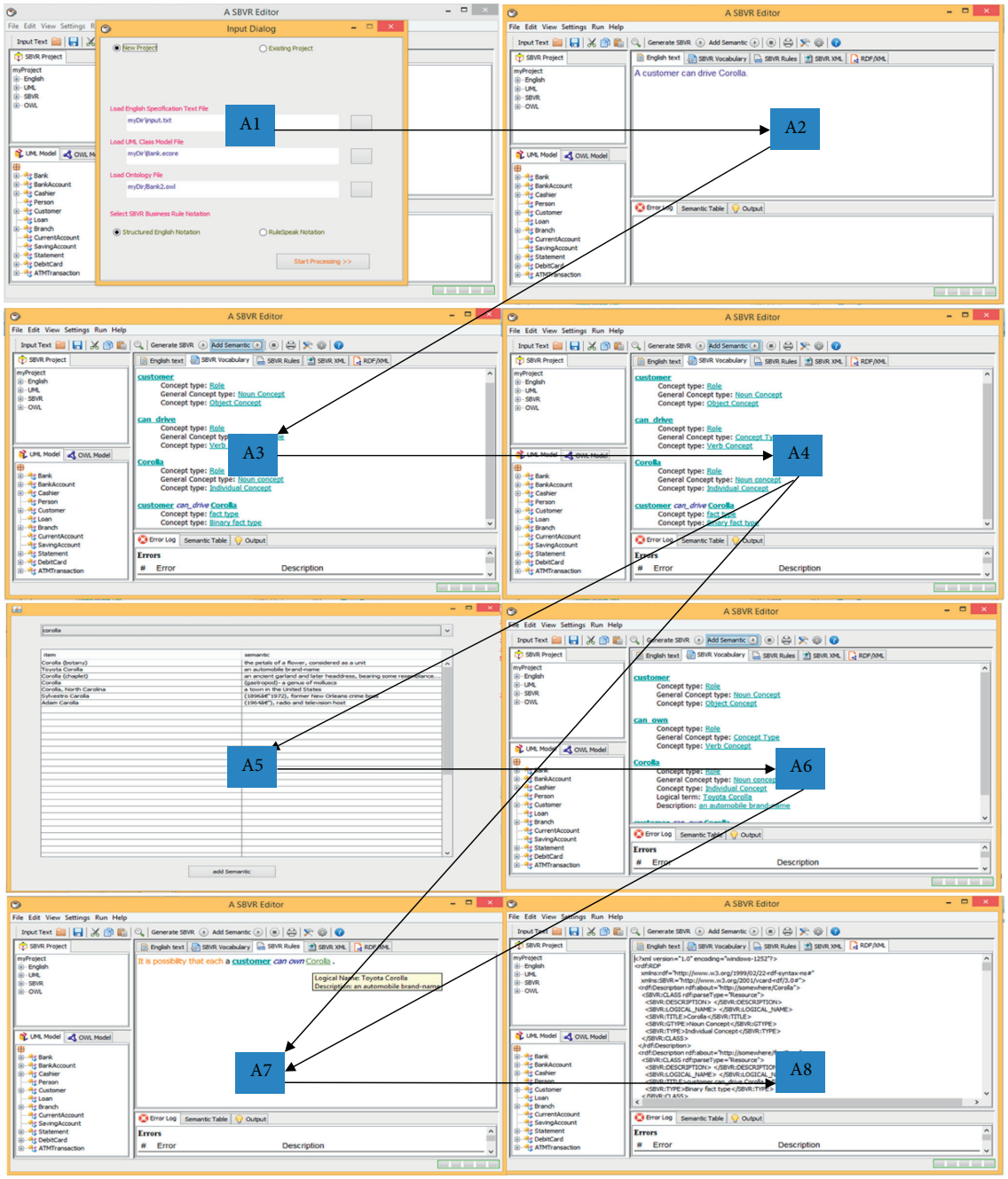

Figure 9: Interaction of different components implementing the NL to SBVR rules transformation.

requirements, 339 fact Types are correctly identified. The approach has incorrectly tagged some Noun Concepts as Verb Concepts and vice versa. The reason is the lack of a strong dataset. For example, in set $T1$, 3 Noun Concepts and 2 Verb Concepts are incorrectly tagged. However, overall initial results are very encouraging.Prototype performance is measured using three metrics: Recall, Precision, and F-value. Such metrics are widely used to assess NL-based data extraction systems. Such metrics help to have a comparison of system predictions versus actual values. The Recall is a measure of the correctly identified elements by the system.

$$R = \frac{N_{\text{system}}}{N_{\text{expert}}}, \tag{1}$$

where N_{system} is the number of the system's generated correct results, and N_{expert} is the number of human expert's generated sample results.

The precision is a ratio between correct and incorrect elements identified by the system.

$$P = \frac{N_{\text{correct}}}{N_{\text{incorrect}} + N_{\text{correct}}}, \tag{2}$$

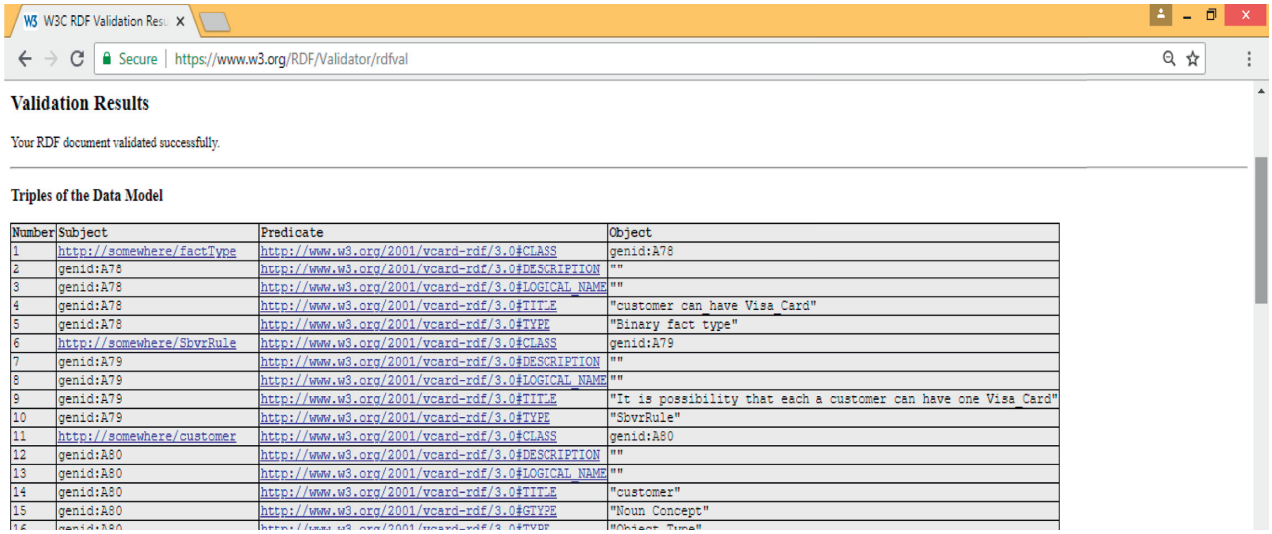

Figure 10: W3C generated triplets verifying system generated RDF/XML output.

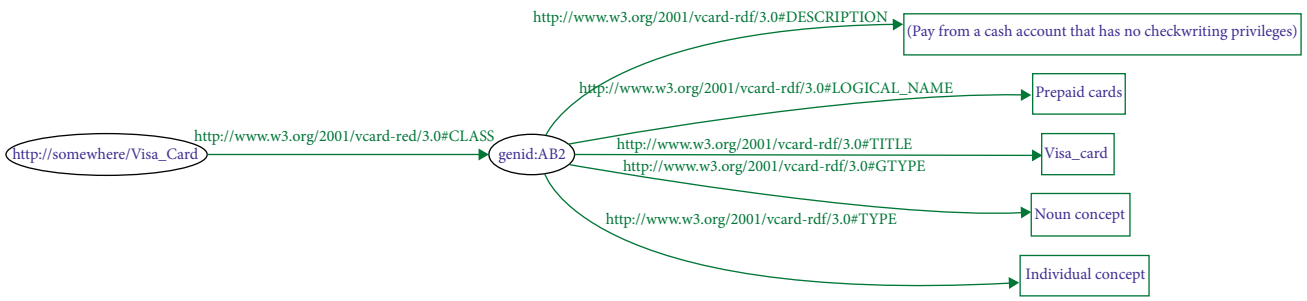

Figure 11: W3C-generated graphs verifying system-generated RDF/XML output.

Table 12: Result of executed NL text software requirements.

Requirements set	Result type	Extracted result
T	SBVR Noun Concept	**company_name**, **Rent-a-Car**, **company**, **drivers**, **hours**, **day**, **cars**, **Ahmad**
	SBVR Verb Concept	**is**, **employed**, **work**, **owns**
	SBVR Fact Type	**company_name** *is* **Rent-a-Car company** *employed* **drivers drivers** *work* **hours company** *owns* **cars Ahmad** *is* **driver**
	SBVR Rules	It is permitted that each **company_name** *is* "Rent-a-Car." each **company** *employed* 5 **driver**.each **driver** *work* 8 **hours** each a **day**.each **company** *owns* 5 **car**. It is permitted that Ahmad *is* each **name** of each a **driver**.

Table 13: Extracted elements by prototype.

Type/Metrics	Requirements set							
	T1	T2	T3	T4	T5	T6	T7	Total
Noun Concept	89	93	92	90	87	92	81	624
Verb Concept	48	48	47	47	50	46	46	332
Fact Type	50	48	49	50	50	49	50	346
SBVR Rules	50′	50	50	50	50	50	50	350

Table 14: Comparison of sample and prototype generated results.

Requirements set	Type/metrics	N_{expert}	N_{total}	$N_{correct}$	$N_{incorrect}$	$N_{missing}$
$T1$ (50 requirements)	Noun Concept	94	89	86	3	5
	Verb Concept	50	48	46	2	2
	Fact Type	50	50	48	2	0
	Rules	50	50	50	0	0
$T2$ (50 requirements)	Noun Concept	105	93	93	0	12
	Verb Concept	50	48	45	3	2
	Fact Type	50	48	47	1	2
	Rules	50	50	50	0	0
$T3$ (50 requirements)	Noun Concept	100	92	86	6	8
	Verb Concept	50	47	44	3	3
	Fact Type	50	49	49	0	1
	Rules	50	50	50	0	0
$T4$ (50 requirements)	Noun Concept	99	90	86	4	9
	Verb Concept	50	47	45	2	3
	Fact Type	50	50	47	3	0
	Rules	50	50	50	0	0
$T5$ (50 requirements)	Noun Concept	100	87	82	5	13
	Verb Concept	50	50	49	1	0
	Fact Type	50	50	49	1	0
	Rules	50	50	50	0	0
$T6$ (50 requirements)	Noun Concept	102	92	87	5	10
	Verb Concept	52	46	42	4	6
	Fact Type	50	49	45	4	1
	Rules	50	50	50	0	0
$T7$ (50 requirements)	Noun Concept	100	81	75	6	19
	Verb Concept	50	46	44	2	4
	Fact Type	50	50	50	0	0
	Rules	50	50	50	0	0
Total Identified Elements		1752	1652	1595	57	100

where $N_{correct}$ is the number of correct results, and $N_{incorrect}$ is the number of incorrect results generated by the developed system.

F-measure is the measure of the prototype's accuracy.

$$F = \frac{2(P)(R)}{P + R}, \tag{3}$$

where P is the precision value and R is the recall value.

Table 15 describes the calculated recall, precision, and F-value of the prototype for each NL requirements set ($T1$, $T2$, ..., $T7$). The calculation is based on the abovementioned Equations (1), (2), and (3). Recall, Precision, and F-value are separately calculated for each set of requirements ($T1$, $T2$, ..., $T7$). After that, an average value is calculated. $T1$ has the highest Recall (0.97) and F-value (0.97), while $T2$ has the highest Precision (0.98). $T7$ has the lowest Recall (0.91) value, while $T6$ has the lowest Precision (0.95). The reason for such low values is a weak dataset. The average recall for the SBVR SRS document is calculated as 0.94, while the average precision is calculated as 0.97. The average F-value is computed as 0.95. The results of this initial performance evaluation are very encouraging. The results support both the used approach and the potential of this tool in general. Figure 12 shows the graphical representation of the tool's evaluation.

Figure 13 displays the N_{sample}, $N_{correct}$, $N_{incorrect}$, and $N_{missing}$ evaluation results of the proposed tool for the seven case studies, $T1$–$T7$. The blue column shows results for N_{sample}, the red column shows the result for $N_{correct}$, the green column shows results for $N_{incorrect}$, while the purple column shows the results for $N_{missing}$. $T2$ has the highest value of $N_{correct}$ elements, while $T6$ has the lowest value of $N_{correct}$ elements. $T1$ has the lowest value of $N_{missing}$ elements, while $T7$ has the highest value of $N_{missing}$ elements.

The abovementioned Figure 13 graphically represents the results of Recall, precision, and F-value generated by the prototype while processing seven different case studies ($T1$, $T2$, ..., $T7$). According to our calculated results, $T1$ has high Recall, $T2$ has high precision, and $T1$ has high F-Value. In contrast, $T7$ has the lowest Recall value, $T6$ has the lowest Precision, and $T7$ has the lowest F-measure value. The reason for such low values is a weak dataset.

The resultant output is stored in RDF/XML format. To verify XML schema, the XML file is parsed by an online validation service named W3C RDF Validation Service [47]. This web service successfully replicates the output into triples and graphs. A sample section of web-generated output is shown in Figure 10.

The results presented above show that it is convenient and time-saving to formulate a semantically formal and controlled illustration using our proposed approach. Figure 10 shows the validation results and Figure 11 shows the W3C-generated graphs verifying system-generated RDF/XML output.

Table 15: Evaluation results of prototype.

Input	N_{expert}	$N_{correct}$	$N_{incorrect}$	$N_{missing}$	Recall	Precision	F-value
T1	244	230	7	7	0.97	0.97	0.97
T2	255	235	4	16	0.94	0.98	0.96
T3	250	229	9	12	0.95	0.96	0.96
T4	249	228	9	12	0.95	0.96	0.96
T5	250	230	7	13	0.95	0.97	0.96
T6	254	224	13	17	0.93	0.95	0.94
T7	250	219	8	23	0.91	0.96	0.94
Total	1752	1595	57	100			
Average					0.94	0.97	0.95

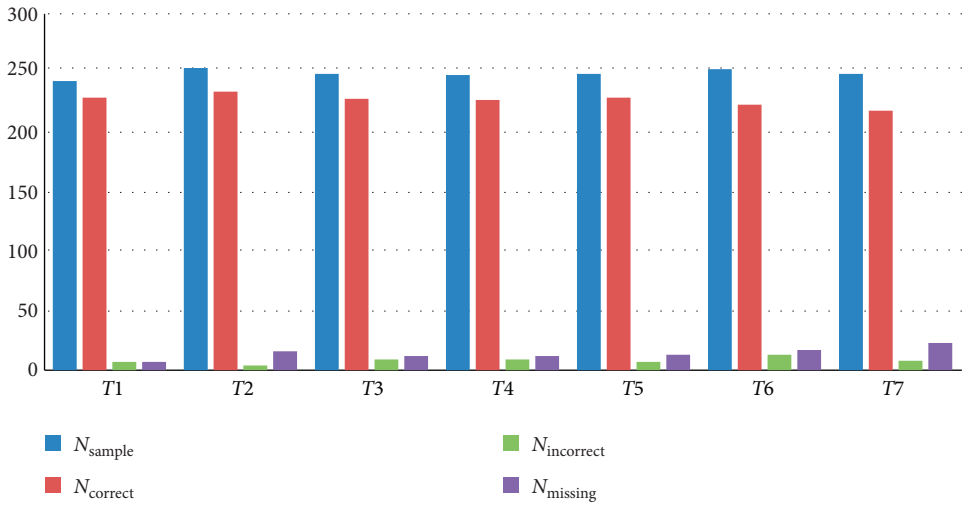

Figure 12: Evaluation result of the proposed tool.

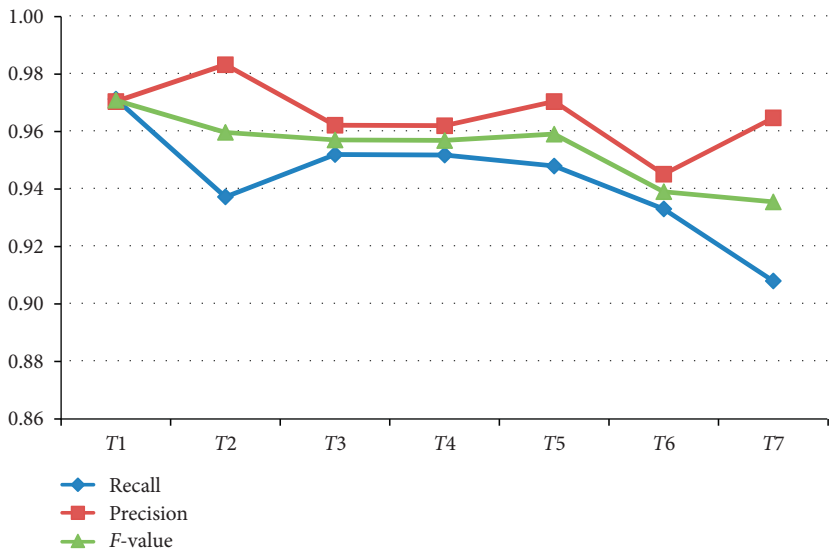

Figure 13: Recall, precision, and F-values of the evaluation result.

8. Scope and Limitations

We recognize a set of limits of this investigative study, which would permit the researchers to present a more suitable design for the succeeding research phase. Our study used only one expert to create sample data, which is not a representative set of the population of experts. Furthermore, the experience level of the expert can also put limitations on the credibility of the sample data.

It is assumed that the given English text to the prototype as the input is grammatically correct. The dataset used for experimental evaluation contains complete sentences, i.e., subject–verb–object. Results may be different for incomplete or incorrect sentences.

9. Conclusion and Future Work

The major goal of this research work was to automate the practice of software requirement elicitation and requirement specification while tackling the ambiguous temperament of NL such as English. At the same time, we aim to generate a controlled representation of such requirements, which is domain-independent and acceptable for the business industry. To tackle the challenge, we have proposed an NL-based approach. We developed a prototype based on the proposed approach. The prototype parse English written software requirement and generate the SBVR-based controlled representation. The prototype further extracts the SBVR vocabulary to generate SBVR rules. The proposed prototype performs the lexical, syntactic, and semantic analyzes concerning SBVR rules. While extracting the SBVR vocabulary, the assignment of meanings to each vocabulary element removes the chances of ambiguity. The prototype generates rules written in the SBVR-based controlled language. Such rules are clear and unambiguous among the business community. The output of the whole process is stored in XML format, which is portable and platform-independent. Also, the generated output document is easily convertable into any other format for further processing. Such documents will be easy for the machine to process.

We have, successfully, evaluated the proposed approach on seven case studies with the aid of a developed prototype to support our proposed approach. Our proposed prototype can be used for automated Object-oriented analysis and design (OOA&D) of NL-software requirements. In addition, the prototype offers a higher accuracy as compared to other available NL-based tools. As shown in the Results section, the recall value of 0.94 and precision value of 0.97 results obtained from seven case studies for software requirements by using our prototype are very encouraging. Likewise, the resultant F-value of 0.95 is also satisfactory. Hence, the results of our assessment show the encouraging performance of our developed tool in terms of usability, time, and accuracy.

Beforehand, research has been carried out in large amount for the automation of SRS document using NLP-based approaches, but comparatively, little effort has been done on the approaches based on CNL representing requirement specification. For that reason, countless aspects need to be investigated while using the SBVR-based controlled representation of requirements specification. The future work is to validate the extracted vocabulary automatically using UML and ontology models. Automated validation of such data can be helpful in automated conceptual modeling of the NL SRS document.

Data Availability

Data are available upon request to the corresponding author.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] A. O. J. A. Sabriye and W. M. N. W. Zainon, "A framework for detecting ambiguity in software requirement specification," in *Proceedings of the 2017 8th International Conference on Information Technology (ICIT)*, pp. 209–213, IEEE, Amman, Jordan, May 2017.

[2] T. Hovorushchenko and O. Pomorova, "Methodology of evaluating the sufficiency of information on quality in the software requirements specifications," in *Proceedings of the 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, pp. 370–374, IEEE, Kyiv, Ukraine, May 2018.

[3] M. Luisa, F. Mariangela, and N. I. Pierluigi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering*, vol. 9, no. 1, pp. 40–56, 2004.

[4] E. Kamsties and B. Peach, "Taming ambiguity in natural language requirements," in *Proceedings of the Thirteenth International Conference on Software and Systems Engineering and Applications*, Paris, France, December 2000.

[5] M. P. S. Bhatia, A. Kumar, and R. Beniwal, "Ontology based framework for detecting ambiguities in software requirements specification," in *Proceedings of the 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 3572–3575, IEEE, New Delhi, India, March 2016.

[6] M. Bano, "Addressing the challenges of requirements ambiguity: a review of empirical literature," in *Proceedings of the 2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, pp. 21–24, IEEE, Ottawa, Canada, August 2015.

[7] D. M. Berry, E. Kamsties, and M. M. Krieger, *From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity*, University of Waterloo, Waterloo, Canada, 2003, https://cs.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf.

[8] V. Basili, G. Caldiera, F. Lanubile, and F. Shull, "Studies on reading techniques," in *Proceedings of the Twenty-First Annual Software Engineering Workshop*, vol. 96, p. 002, Greenbelt, MD, USA, December 1996.

[9] E. Kamsties, "Understanding ambiguity in requirements engineering," in *Engineering and Managing Software Requirements*, pp. 245–266, Springer, Berlin, Germany, 2005.

[10] S. McConnell, *Code Complete*, Pearson Education, London, UK, 2004.

[11] A. Umber and I. S. Bajwa, "Minimizing ambiguity in natural language software requirements specification," in *Proceedings of the IEEE Sixth International Conference on Digital Information Management (ICDIM 2011)*, pp. 102–107, Melbourne, Australia, 2011.

[12] A. Takoshima and M. Aoyama, "Assessing the quality of software requirements specifications for automotive software

systems," in *Proceedings of the 2015 Asia-Pacific Software Engineering Conference (APSEC)*, pp. 393–400, IEEE, New Delhi, India, December 2015.

[13] F. Zait and N. Zarour, "Addressing lexical and semantic ambiguity in natural language requirements," in *Proceedings of the 2018 Fifth International Symposium on Innovation in Information and Communication Technology (ISIICT)*, pp. 1–7, IEEE, Amman, Jordan, November 2018.

[14] A. Chikh and H. Alajmi, "Towards a dynamic software requirements specification," in *Proceedings of the 2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, pp. 1–7, IEEE, Hammamet, Tunisia, January 2014.

[15] R. Schwitter, "Controlled natural languages for knowledge representation," in *Proceedings of the Coling 2010: Posters*, pp. 1113–1121, Beijing, China, August 2010.

[16] R. Denaux, V. Dimitrova, A. G. Cohn, C. Dolbear, and G. Hart, "Rabbit to OWL: ontology authoring with a CNL-based tool," in *Proceedings of the International Workshop on Controlled Natural Language*, pp. 246–264, Springer, Marettimo, Italy, June 2009.

[17] C. White and R. Schwitter, "An update on PENG light," in *Proceedings of the Australasian Language Technology Association Workshop 2009*, pp. 80–88, Sydney, Australia, December 2009.

[18] G. Sandhu and S. Sikka, "State-of-art practices to detect inconsistencies and ambiguities from software requirements," in *Proceedings of the International Conference on Computing, Communication & Automation*, pp. 812–817, IEEE, Greater Noida, India, May 2015.

[19] P. Martin, "Knowledge representation in CGLF, CGIF, KIF, frame-CG and formalized-English," in *Proceedings of the International Conference on Conceptual Structures*, pp. 77–91, Springer, Borovets, Bulgaria, July 2002.

[20] U. S. Shah and D. C. Jinwala, "Resolving ambiguities in natural language software requirements: a comprehensive survey," *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 5, pp. 1–7, 2015.

[21] F. Friedrich, J. Mendling, and F. Puhlmann, "Process model generation from natural language text," in *Proceedings of the International Conference on Advanced Information Systems Engineering*, pp. 482–496, Springer, London, UK, June 2011.

[22] H. Afreen and I. S. Bajwa, "Generating UML class models from SBVR software requirements specifications," in *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*, pp. 23–32, Ghent, Belgium, 2011.

[23] S. J. Korner and T. Brumm, "RESI-a natural language specification improver," in *Proceedings of the 2009 IEEE International Conference on Semantic Computing*, pp. 1–8, IEEE, Berkeley, CA, USA, September 2009.

[24] L. Mich and R. Garigliano, "Ambiguity measures in requirement engineering," in *Proceedings of the International Conference on Software Theory and Practice*, ICS, Beijing, China, August 2000.

[25] S. F. Tjong, "Avoiding ambiguity in requirements specifications," Ph.D. thesis, https://cs.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/TjongThesis.pdf, University of Nottingham, Nottingham, UK, 2008.

[26] H. Yang, A. De Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, "Analysing anaphoric ambiguity in natural language requirements," *Requirements Engineering*, vol. 16, no. 3, p. 163, 2011.

[27] E. Kamsties, D. M. Berry, B. Paech, E. Kamsties, D. M. Berry, and B. Paech, "Detecting ambiguities in requirements documents using inspections," in *Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01)*, pp. 68–80, Paris, France, July 2001.

[28] A. Ferrari, G. Lipari, S. Gnesi, and G. O. Spagnolo, "Pragmatic ambiguity detection in natural language requirements," in *Proceedings of the 2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, pp. 1–8, IEEE, Karlskrona, Sweden, August 2014.

[29] H. Kaiya and M. Saeki, "Ontology based requirements analysis: lightweight semantic processing approach," in *Proceedings of the Fifth International Conference on Quality Software (QSIC'05)*, pp. 223–230, IEEE, Melbourne, Australia, September 2005.

[30] B. Gleich, O. Creighton, and L. Kof, "Ambiguity detection: towards a tool explaining ambiguity sources," in *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality*, pp. 218–232, Springer, Essen, Germany, June 2010.

[31] O. Al-Harbi, S. Jusoh, and N. Norwawi, "Handling ambiguity problems of natural language interface for question answering," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 3, p. 17, 2012.

[32] R. P. Verma and M. R. Beg, "Representation of knowledge from software requirements expressed in Natural Language," in *Proceedings of the 2013 6th International Conference on Emerging Trends in Engineering and Technology*, pp. 154–158, IEEE, Nagpur, India, December 2013.

[33] K. D. Gill, A. Raza, A. M. Zaidi, and M. M. Kiani, "Semi-automation for ambiguity resolution in open source software requirements," in *Proceedings of the 2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–6, IEEE, Toronto, Canada, May 2014.

[34] A. K. Massey, R. L. Rutledge, A. I. Antón, and P. P. Swire, "Identifying and classifying ambiguity for regulatory requirements," in *Proceedings of the 2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pp. 83–92, IEEE, Karlskrona, Sweden, August 2014.

[35] S. W. Ali, Q. A. Ahmed, and I. Shafi, "Process to enhance the quality of software requirement specification document," in *Proceedings of the 2018 International Conference on Engineering and Emerging Technologies (ICEET)*, pp. 1–7, IEEE, Lahore, Pakistan, February 2018.

[36] D. Popescu, S. Rugaber, N. Medvidovic, and D. M. Berry, "Reducing ambiguities in requirements specifications via automatically created object-oriented models," in *Proceedings of the Monterey Workshop*, pp. 103–124, Springer, Monterey, CA, USA, September 2007.

[37] N. E. Fuchs, K. Kaljurand, and T. Kuhn, "Attempto controlled English for knowledge representation," in *Reasoning Web*, pp. 104–124, Springer, Berlin, Heidelberg, 2008.

[38] http://www.omg.org/spec/SBVR/1.5/PDF.

[39] I. S. Bajwa, M. G. Lee, and B. Bordbar, "SBVR business rules generation from natural language specification," in *Proceedings of the 2011 AAAI Spring Symposium Series*, San Francisco, CA, USA, March 2011.

[40] S. Ramzan, I. S. Bajwa, I. U. Haq, and M. A. Naeem, "A model transformation from NL to SBVR," in *Proceedings of the Ninth International Conference on Digital Information Management (ICDIM 2014)*, pp. 220–225, IEEE, Phitsanulok, Thailand, September 2014.

[41] P. Danenas, T. Skersys, and R. Butleris, "Natural language processing-enhanced extraction of SBVR business vocabularies and business rules from UML use case diagrams," *Data & Knowledge Engineering*, vol. 128, Article ID 101822, 2020.

[42] P. B. F. Njonko and W. El Abed, "From natural language business requirements to executable models via SBVR," in *Proceedings of the 2012 International Conference on Systems and Informatics (ICSAI2012)*, pp. 2453–2457, IEEE, Yantai, China, May 2012.

[43] F. L. Siqueira, T. C. de Sousa, and P. S. M. Silva, "Using BDD and SBVR to refine business goals into an Event-B model: a research idea," in *Proceedings of the 2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE)*, pp. 31–36, IEEE, Buenos Aires, Argentina, May 2017.

[44] P. K. Chittimalli and K. Anand, "Domain-independent method of detecting inconsistencies in sbvr-based business rules," in *Proceedings of the International Workshop on Formal Methods for Analysis of Business Systems*, pp. 9–16, Singapore, September 2016.

[45] P. Arnold and E. Rahm, "Automatic extraction of semantic relations from wikipedia," *International Journal on Artificial Intelligence Tools*, vol. 24, no. 2, Article ID 1540010, 2015.

[46] L. Hirschman and H. S. Thompson, "Chapter 13 evaluation: overview of evaluation in speech and natural language processing," in *Survey of the State of the Art in Human Language Technology*, R. A. Cole, H. Mariani, J. Uszkoreit et al., Eds., pp. 114–126, Cambridge University Press, Cambridge, UK, 1995, http://cslu.cse.ogi.edu/HLTsurvey/HLTsurvey.html.

[47] https://www.w3.org/RDF/Validator/rdfval, 2020.

[48] K. Toutanova and C. Manning, "Enriching the knowledge sources used in a maximum entropy part-of-speech tagger," in *Proceedings of the 2000 Joint SIGDAT Conference EMNLP/VLC*, vol. 63–71, Hong Kong, China, 2000.

[49] I. S. Bajwa, A. Samad, and S. Mumtaz, "Object oriented software modeling using NLP based knowledge extraction," *European Journal of Scientific Research*, vol. 35, no. 1, pp. 22–33, 2009.