SVM Machine Learning Classifier to Automate the Extraction of SRS Elements

Ayad Tareq Imam[1], Aysh Alhroob[2]
Faculty of Information Technology
Isra University, Amman, Jordan

Wael Jumah Alzyadat[3]
Faculty of Science and Information Technology
Al-Zaytoonah University, Amman, Jordan

***Abstract*—The process of extraction of software entities such as system, use case, and actor from an English natural language description of a user's software requirements is a linguistic and semantic process of a natural language processing application. Entity extraction is known to be a complicated and challenging problem by researchers in the fields of linguistics or computation, due to the ambiguities in natural languages. This paper presents a named entity recognition method called SyAcUcNER (System Actor Use-Case Named Entity Recognizer), for extracting the system, actor, and use case entities from unstructured English descriptions of user requirements for the software. SyAcUcNER uses one of the Machine Learning (ML) approaches, that is, the Support Vector Machine (SVM) as an effective classifier. Also, SyAcUcNER uses a semantic role labeling process to tag the words in the text of user software requirements. SyAcUcNER is the first work that defines the structure of a requirements engineering specialized NER, the first work that uses a specialized NER model as an approach for extracting actor and use case entities from English language requirements description, and the first time an SVM has been used to specify the semantic meanings of words in a certain domain of discourse; that is the Software Requirements Specification (SRS). The performance of SyAcUcNER, which utilizes WEKA's SVM, is evaluated using a binomial technique, and the results gained from running SyAcUcNER on text corpora from assorted sources give weighted averages of 76.2% for precision, 76% for recall, and 72.1% for the F-measure.**

Keywords*—Information extraction; named entity recognition; machine learning; support vector machine; software requirement specification; WEKA; I-CASE

I. Introduction

The system, use case, and actor are the main entities of the Software Requirements Specification (SRS), which is an unformatted Natural Language (NL) text description of a system. The extracting of these entities is considered the first step in the development of desired information system, as the actors are the individuals that use the system like humans, external software, etc., in which each actor has certain roles, and the use cases are used to (1) identify the functional requirements of the developed system that would be used by actors, (2) design the system's architecture, (3) control the implementation of the system, (4) verify and validate the developed system via generating test cases [1].

Based on the above, the extracting of the system, actor, and use case entities from SRS has been recognized as a key step in analyzing software user requirements and it is achieved by using systematic definitions of these entities [2]. Usually, a manual approach, which was described algorithmically by [3] and [4], is used to achieve the process of extracting SRS elements.

Due to the unstructured style of the written SRS, certain problems exist that impose a careful linguistic analysis by a human to be accomplished properly. As a consequence, the manual approach can be error-prone and time-consuming [5] [6]. To facilitate and speed up the performing of extracting the SRS elements from an unstructured and natural language-formed user requirement text, a set of solutions have been proposed to automate this process.

The previously proposed solutions for automating the extraction of SRS fall into two approaches. The first one is the production rules approach, which is, in general, has shortcomings like vagueness, inefficiency (time-consuming execution, intelligent interpreter, and difficulty to follow the execution control), absence of learning ability, and the resolution's conflict [7] [8] [9]. The second approach is the connectionist approach – or the Artificial Neural Networks (ANN), which is (in addition to be computationally expensive) has some problems like the poor ability to predict, the excessive training required for developing a solution, the long time that is required to develop a network, and the unexplainable answer (Blackbox) [10] [11]. ANN approach is a Machine Learning (ML) method, which has other methods. Because of the problems that ANN has, this paper proposes the use of another ML method that is the Support Vector Machine (SVM) method to automatically extract system, actor, and use case entities from unstructured NL requirements text documents in English.

An SVM is used to create a learning model based on a supervised learning approach that uses pre-labeled training data to train the model to classify these data [12]. SVM is a non-probabilistic binary classifier that categorizes data into several classes. The binary classification SVM achieves classification by mapping input data to classes (hyperplanes) in an N-dimensional space based on a maximal margin, where N is the number of features of a data point [12]. SVM is a very useful classifier of undistributed data and irregularly distributed data, which can be of different types like text, images, audio, and other types. This is seen in the different and many real-world applications where SVM is used such as sentiment analysis, handwriting recognition, a cancer diagnosis.

Although there are many classification algorithms in machine learning, yet, SVM has been shown to achieve significantly better and more robust classification than other -supervised learned- classification algorithms due to the following outstanding properties [12] [13]:

1) SVM is distinguished in learning by:

- SVM has no overfitting problem.
- SVM can apply to semi-supervised learning models also.
- SVM works stably and it generalizes well to data not included in the training data set or that data that its features would be changed. This is because the SVM classification approach is principally reliant on a subset of points only in its work to maximize the gap (margin) between nearby points of classes. It means that only an inliers subset of points is helpful and no need to consider outliers points.
- SVM is a fast-learning algorithm as the kernel function of SVM is performed for the classification per training sample. Worthy to note that the Polynomial kernel was proved as a better factor in SVM.
- SVM is robust, which is shown by its ability to produce a unique solution.

2) SVM is more efficient in high n-dimensional space, in cases where the number of samples is less than the number of dimensions and is relatively memory efficient.

3) SVM delivers accurate results due to the following:

- The generated hyperplane creates a clear margin to separate classes, and as the large margin is as a more corrected classification of data would be gained. The soft margin is used with non - linearly separable data and the hard margin is used with linearly separable data.
- The convex optimization nature of SVM makes the answer a global minimum rather than a local minimum, which in turn yields more optimality confidence in the results.

4) SVM can be adapted to work with different data types. This is because SVM has a built-in kernel function, which is a technology that provides the ability to solve any complex problem. Note that Kernel is a non-parametric (linear or nonlinear) identifiable function that comes with different forms depending on the data it operates on.

5) Generally, the SVM classifier has better computational complexity than the other classifiers. SVM has a very little execution time than the Artificial Neuron Network (ANN). SVM has a faster prediction with better classification accuracy than the Naive Bayes classifier. SVM has less time complexity than the logistic regression classifier. SVM is more robust than the logistic regression, just as there is some bias in the training data set.

6) The availability of library SVM classifiers in many programming languages and packages such as MATLAB, Weka, and Python makes the work with SVM so easy.

As shown, the advantages of SVM make it an attractive method that can be used instead of ANN. Worth mentioning that SVM will underperform and being unsuitable when the data sets are large has more noise (overlapped classes) and has no clear probabilistic justification to have classification [12]. To achieve the goal of automating the extraction of SRS elements, our work should answer the following research questions:

- How can SVM be used to extract certain entities from an unstructured text, and.
- What is the performance of a system that utilizes SVM for extracting SRS elements?

Section 2 of this paper illustrates background theoretical issues and Section 3 examines related works and approaches. Section 4 describes the proposed approach, which is followed by a discussion and evaluation of the experimental results in Section 5, and finally, the conclusions, findings, and recommendations are presented in Section 6.

II. Background

This paper is about using the SVM machine learning classifier as the main part of a Named Entity Recognition (NER) system to automate the extraction of SRS elements.

NER is an ML-based process that is used to find and classify names in unstructured or semi-structured texts. These goals are achieved by annotating the words in the text words with the names of categorized entities in the real world, such as locations, places, organizations, companies, persons, individuals, etc. Stanford CoreNLP [14] and the Apache OpenNLP [15] are two well-known examples of NER that extract real-world entities from a text. Also, there are NER models for extracting beneficial information from biomedical texts, such as mentions of proteins and genes and the relationships between them [26]. There are two types of NER methods: the first is an ontology-based NER, which strongly relies on updates of knowledge to successfully distinguish known terms and concepts in unstructured or semi-structured texts [16], and the second is a deep learning NER, which aims (in addition to recognizing terms and concepts) to cluster words by using a word embedding technique that attempts to understand both the semantics of a word and the syntactic relationship between words [17]. As NER is a central subfunction that extracts and classifies certain information (names in a text) from either semi-structured or unstructured text, it is considered an important sub-task of open information extraction (OIE).

OIE is a process that creates a structured representation of information in an unstructured or semi-structured text. The resulting structured representation is usually in the form of n-ary propositions [18]. OIE aims to extract all types of relations that may exist in a text, whether these are pre-known relations or under discovery. Based on this approach, OIE supports the independent extraction of relations from small, large, and heterogeneous corpora within a specific domain. Automation of the OIE process needs to be efficient, to rely on unsupervised extraction strategies, and to consider corpus heterogeneity [19]. The OIE process is achieved by using several types of NL processing approaches at the semantic level, all of which function to infer the semantics of a word from its particular linguistic attributes. These attributes are linguistic annotations of a word and are used by a processing technique to recognize a word's semantics (or connotation) within a specific domain. Annotation, for example with part-of-speech (POS) tags, is accomplished by using natural language processing (NLP) tools such as parse trees, syntactic, and dependency parsers [20] [9].

NLP is of great importance in creating human-machine interfaces, and accordingly has become an attractive research field, aiming to find and define algorithms, methods, and approaches to give computers the ability to communicate with a human via natural language [9]. NLP is dedicated to allowing a computer system to perform analysis and comprehension, and to specify the meanings of words or even statements that are written in NL. NLP is a difficult issue in computer science; this is due to the nature of NL, as it naturally suffers from the issues of ambiguity and expressiveness, which easily lead to problems with misunderstanding [21]. In general, working with NLP has moved towards viewing the analysis processing of language as being disintegrated into different sub-processes, illustrating the theoretical linguistic singularity for each of the lexical, syntactic, semantic, and pragmatic levels of NLP [20]. The essential view is that the statements are first investigated according to their syntax; this gives a structure that is increasingly amenable to examination regarding semantics. The next stage, which is a pragmatic analysis, aims to specify the true meaning of the text or speech in the context. The three core subprocesses are syntax, semantic, and pragmatic, all of which serve as a starting point in the processing of texts formed using NL [20]. The standard analysis stages in NLP are [20] [9]:

1) Tokenization: The process of breaking up a sentence into elements called tokens.

2) Lexical analysis: A process that aims to check whether a word belongs to a language and to find the part of speech (POS) for the word, or to reveal the class of a word (i.e., verb, noun, or preposition). The lexical analysis also includes the morphological processing of a word, which aims to isolate the stem of the word and its affixes.

3) Syntactic analysis: This applies the grammar of the language (using a parsing algorithm) to identify the legal structure of the input statement.

4) Semantic analysis: This is the process of extracting the exact meaning from the text.

5) Pragmatic analysis: This aims to infer the purpose of the use of the word/text in situations and requires knowledge about the domain of discourse. It is achieved by reinterpreting the text as it really implies.

In short, the linguistic and semantic analysis of a text is carried out either as a semantic analysis of the whole text as a single unit or as a semantic analysis of individual words in a text. The first approach is used to recognize the intention or sentiment of a speaker, and the second is used to extract specific information from a text, or in other words to convert semi-structured and unstructured text to a structured form. Here, NER has a key role in the semantic stage of NLP in terms of extracting the meaning of words and sentences in addition to their relationships.

III. Related Works and Approaches

Automation of the manual approach to extracting actors and use cases from software requirements statements shows that several types of NLP tools and approaches have been used for extracting certain semantics from software functional requirements described in natural language.

The first approach described here is the use of the production rules that govern linguistic properties to extract the elements of the software requirements that are required to develop each use case diagram and class diagram. This approach was utilized by the UMGAR system [22]. A similar technique known as a rule-based approach was proposed by [23] for automatically extracting use cases and goal models from unformatted, NL, and textual documents of requirements. This approach combines a number of methods to detect goals and the entities of use cases along with their relationships from the textual document. The semantic parameterization of textual specifications is used to guide the detection process of the rules. Worth to report here that the Genetic Algorithm (GA) can be utilized as a supporting step – for optimization purposes- to select the best set of production rules that should be manually created earlier. The approach can be seen in the work of [24] to discover the best classification rules for the Car, Zoo, and Mushroom classes, and the work of [25] that used GA (with treebank) to develop a syntactic analyzer to enhance the Parseval score of seed grammar rules.

Production rules may be supported by an NLP tool to facilitate the development of more precise recognition rules. A hybrid NLP tool that combines production rules with predefined types of use cases and actors is used by [1]. Also, a combined NLP and domain ontology approach was used in RAPID, a scheme proposed by [26] that takes textual requirements in NL form and extracts the primary concepts and their relationships to create unified modeling language (UML) diagrams.

An approach using a set of semantic heuristics rules to generate the patterns used to extract the use case model, based on a general NLP tool, was proposed by [27]. The software requirements processed in this work are Arabic natural language texts, and the generated patterns depend on the sentence structure. The Stanford parser is the NLP tool utilized in this approach. This scheme follows the work of [21], which uses an open NLP tool called Semantic Business Vocabulary and Rules to extract object-oriented models from user software requirement specifications (SRS).

Conversion of the description of requirements from a natural language form to structured natural language as a prior step in utilizing other NLP analyzing processes is the approach used by [28]. The conversion process is facilitated by an elicitation process, both of which form part of an expert system that elicits requirements from different stakeholders and maintains a knowledge base that supports the future extraction of certain elements from similar requirement descriptions. A very similar approach is used by [29], who proposes an approach that takes requirements in NL form and converts them to an intermediate structured representation using grammatical knowledge patterns and the dependency analyzing of the requirements statements. This intermediate representation is used to create a class diagram.

An approach using POS, pre-processing, and parsing to extract certain UML models, called GUEST, is proposed by [30]. This is a semi-automated rule-based approach that aims to specify models of the goal and use case from unformatted textual requirements documents. In this scheme, a number of different techniques are utilized to discover and classify the goals, use cases, and their relationships from a text, and semantic parameterization of the textual specifications is carried out. In two selected case studies, GUEST is used to process software user requirements described in NL text, and producing activity and sequence diagrams. A Recursive Object Model (ROM) diagram is utilized to extract semantic information from requirements by [31]. This extracted semantic information then forms the elements required by system modeling language (SysML), which is similar to UML, to produce different system models.

Without denying the achieved results gained by supporting the rule-based approach by GA, NLP tools, heuristic style, and the modeling approach, the shortcomings of the rule-based approach still exist. The general shortcomings of the rule-based approach have been reported in the introduction. The best-reported achievement of the rule-based approach is the one that comes from the work of Marinos et al [1], which was 96% of precision.

The alternative approach to the rule-based approach is the connectionist approach that uses Artificial Neural Networks (ANN) to elicit the SRS's elements. This is a Machine Learning (ML) approach that had been agreed as a good solution to the problems accompanied by the rule-based approach. ANN, together with Semantic Role Labeling (SRL), was suggested by Al-Hroob et al [32] to extract the use case and actor SRS entities from NL statements of user requirements, as this work is the best-reported achievement that is 47.2% of precision.

IV. Proposed SyAcUcNER Approach

Examination of the related works and approaches described above inspired us to seek a new approach to extract SRS semantics, namely the system, actor, and use case. We aimed to find an approach that relies on the linguistic (lexical and semantic) attributes of a word to discover its true SRS semantics.

In this work, we view NER as a process of extracting a structured form (that is, a system, an actor, and a use case form) from semi-structured or unstructured text (i.e. a user requirements text). Here, NER is applied to the specific domain of the user requirements of the software, rather than a real-world domain. NER is accomplished as a mapping process of certain nouns into a predefined system or actor classes of the software requirement domain and certain verbs into a predefined use case class of the software requirements domain. In fact, NER has previously been used in a specific domain by [33], who developed a rule-based NER model for knowledge extraction of evidence-based dietary recommendations (in the biomedical domain).

In our suggested SyAcUcNER approach, NER is an SVM-based model that uses certain linguistic attributes of a word to recognize the entities of the system, the use case, and the actor from a textual description of software requirements. As illustrated in Fig. 1, SyAcUcNER is created during the training phase and is used for extraction during the testing phase. A subprocess involving the linguistic annotation of a statement's tokens is performed in both phases to prepare the data that will be used for recognition by the SVM data mining model in the training phase and SyAcUcNER in the testing phase.

A. *NL Functional Requirements*

The data set used to train SVM contains 66 English language statements with different structures, representing software requirements. We collected these statements from various sources, such as books and examples in the literature, and also from actual software analysis tasks. In each statement, the tokens representing a system, an actor, and a use case of SRS are manually defined, and their linguistic attributes (lexical category, SRL, and dependency relations) are automatically extracted and exported to an Excel spreadsheet. Due to the exceptional importance of the data set in creating an effective classification model, certain properties are considered when selecting these 66 NL statements of functional requirements. These properties are the numbers and types of system, use cases, and actors that exist in an NL functional requirements statement.

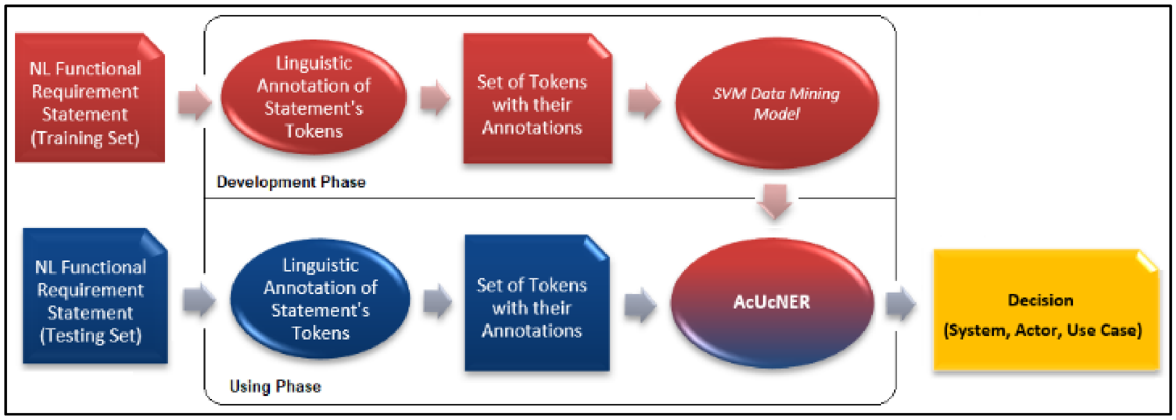

NL Functional Requirement Statement (Training Set)
Linguistic Annotation of Statement's Tokens
Set of Tokens with their Annotations
SVM Data Mining Model
Development Phase
NL Functional Requirement Statement (Testing Set)
Linguistic Annotation of Statement's Tokens
Set of Tokens with their Annotations
AcUcNER
Decision (System, Actor, Use Case)
Using Phase

Fig. 1. The Proposed SyAcUcNER Approach.

B. *Annotation of the Tokens of the Sentence*

The SRSs are tokenized into words, and each word is given linguistic attributes that are used to distinguish the word. The tokenization of a statement aims to isolate the words within it, as a first step in eliciting the system, actors, and use cases. In this paper, we use the following linguistic attributes of a word to distinguish its true semantics in an NL functional requirement statement (system, actor, and use case):

- Lexical attribute: This is the linguistic type of a word in a language. In English, these include a noun, verb, adjective, adverb, conjunction, particle, and adposition [20] [9]. This attribute is usually given with the word in a dictionary. The realization of the lexical category of a word is automatically achieved using computerized NLP systems [9].
- SRL attribute: This is the thematic role property of a noun within a statement (rather than the lexical semantics of a word).

The annotation of semantic roles is an approach in which the arguments (nouns) of a predicate (usually a verb) are classified based on a predefined set of participant types (annotations). These participant types are either the semantic relationships between the arguments of the verb or the circumstance that is described by the verb. The participant types (i.e., the annotations), which are known as semantic or thematic roles, are defined by linguistics [34] [35] [36], as illustrated by Table I, which lists the known thematic roles of a noun. SRL is the process of automatically assigning a semantic role to a noun [9]. In SRL, the verb is considered the predicate, and the semantic role labels or annotations that label a verb's arguments (nouns) are used to specify the true meaning of the verb (predicate) itself. The author in [37] gives an example to illustrate the use of the SRL approach to realize the semantics of a verb by explaining how to differentiate between break and hit verbs: a hit verb has the argument (Agent, Instrument, Place) and the verb break has the arguments (Agent, Instrument, Object). In practice, the semantics of verbs have been used in a number of studies where the verb is the core element of a linguistic process, for example, the development of an approach for converting pseudocode to C# [38].

- Dependency (clausal argument) relations attribute: Dependency relations are a set of directed binary grammatical relationships that exist among the words of a text. These relations are used to encode significant hidden information that results from the analysis of a complex phrase structure. Dependency grammars are the formalisms that use clausal argument relation annotations to tag binary grammatical relationships between the syntactic words (or lemmas) in a sentence. This type of grammar and its parsing scheme is of key importance in dealing with morphologically rich languages that have a relatively free order of words [39] [9]. Fig. 2 illustrates an example of a method based on dependency grammars.

Clausal argument (dependency) relations are defined (among other types of dependency) in a universal dependency (UD) set, and this annotation method uses a dependency parsing process to achieve this task [9]. UD dependency banks are available for more than 50 languages. This is due to the fact that each language has its own set of dependencies and may or may not share these with other languages; in addition, some languages have no UDs. This means that a balance must be found between universality and meaningful dependencies, and with other requirements such as parsing efficiency, ease of human annotation, etc. Another challenge is presented by the vagueness that limits the identification of all UD classes [8].

Although there are continuing efforts to define a cross-linguistically and computationally useful set of dependency relations, it is worth mentioning here a linguistically motivated study of UDs that is handled by [41]. Table II shows a subset of the clausal argument relations in UD (others are found at https://universaldependencies.org/u/dep/).

TABLE I. LIST OF THEMATIC ROLES

Thematic Roles	**Definition**
Agent	An action's doer/instigator, denoted by the predicate
Patient	An action's 'undergoer', denoted by the predicate
Theme	An action's moved entity, denoted by the predicate
Experiencer	An action's living-entity practitioner, denoted by the predicate
Goal (direction)	An object's destination, indicated by a transfer event
Beneficiary	The entity that gets the benefit denoted by the predicate
Source (origin)	The location from which something moves
Instrument	The medium used to act, denoted by the predicate
Locative	The situation/location in which the action occurred
Stimulus	Accidental sensory trigger
Force or natural cause	The entity that does the action
Recipient	The entity that denotes a change in ownership, possession
Time	The time of occurrence of an action
Manner	How an action is accomplished
Purpose	The reason for performing an action
Cause	The reason for the action occurring

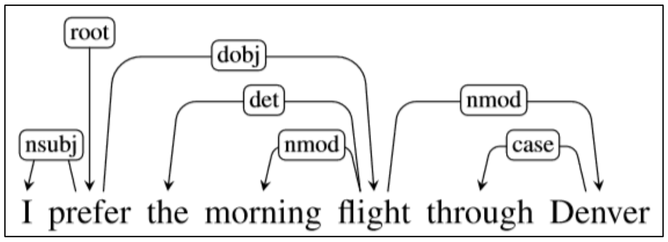


Fig. 2. Results of Dependency Parsing of a Sentence [9].

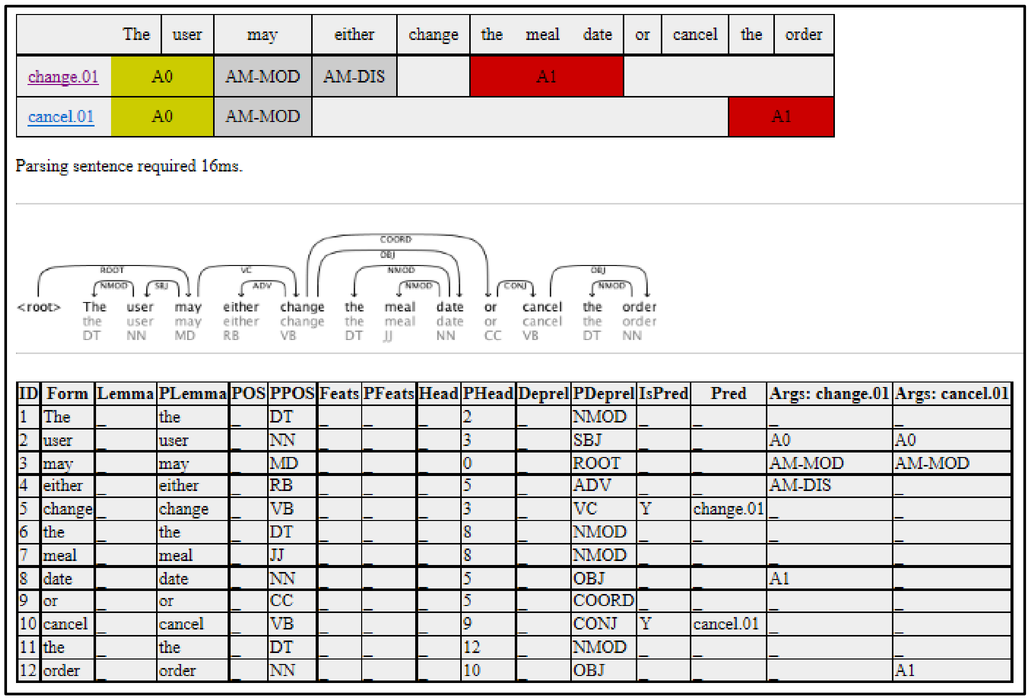
TABLE II. SELECTED DEPENDENCY RELATIONS FROM THE UD SET [40]

Clausal argument relations	**Description**
advmod	adverbial modifier
amod	adjectival modifiers
aux	auxiliary
cc	coordinating conjunction
ccomp	clausal complement
conj	conjunct
dep	dependent
det	determiner
dobj	direct object
iobj	indirect object

To facilitate the process of extracting the SRL and dependency relationships between the words in an SRS, we used an NLP software tool that can provide these linguistic attributes for SRS tokens in English. The LTH (Lunds Tekniska Högskola) System for Frame-Semantic Structure Extraction (or SRL) software tool is used in this work, as it allows for dependency parsing and SRL in addition to other NLP processes such as tokenization, POS-tagging, lemmatizing, morphological tagging, and graph visualization [42]. Fig. 3 illustrates the semantic parsing results yielded by the LTH system for an SRS. The LTH system provides a table of annotation data for tokens (the second table of parsing results) based on a CoNLL-2009 shared task.

A CoNLL-2008 shared task is used to define the format of the data provided in a CoNLL-2009 shared task, with some modifications related to enhancing the performance of the CoNLL-2009 shared task over the CoNLL-2008 shared task. Although they are similar for all-natural languages, they may vary in terms of content [43]. The lexical attribute of a token is obtained from the predicted part of the speech (PPOS) field (coded as NN for the name, VB for the verb, etc.). The dependency relation attribute is obtained from the PDEPREL field. The semantic roles of the arguments of a predicate are obtained by following the hyperlink of the predicate (verb) that appears in the parsing table (the first table in Fig. 3). For example, the arguments of the predicate (verb) change.01 shown in Fig. 4, are 'the user' (coded as A0, i.e. an Agent semantic role), and ‘the meal date’ (coded as A1, i.e. a Patient semantic role).

It is important to note the differences between the standard values of the lexical, SRL, and dependency relations and those of CoNLL-2009 (the core of the LHT system used here). The latter aims to perform and evaluate SRL using a dependency-based representation to predict syntactic and semantic relations [44]. CoNLL-2009 [45] provides a more complicated model of syntactic dependencies, based on a belief that more syntactic dependencies lead to more effective semantic processing, especially in applications such as IE.

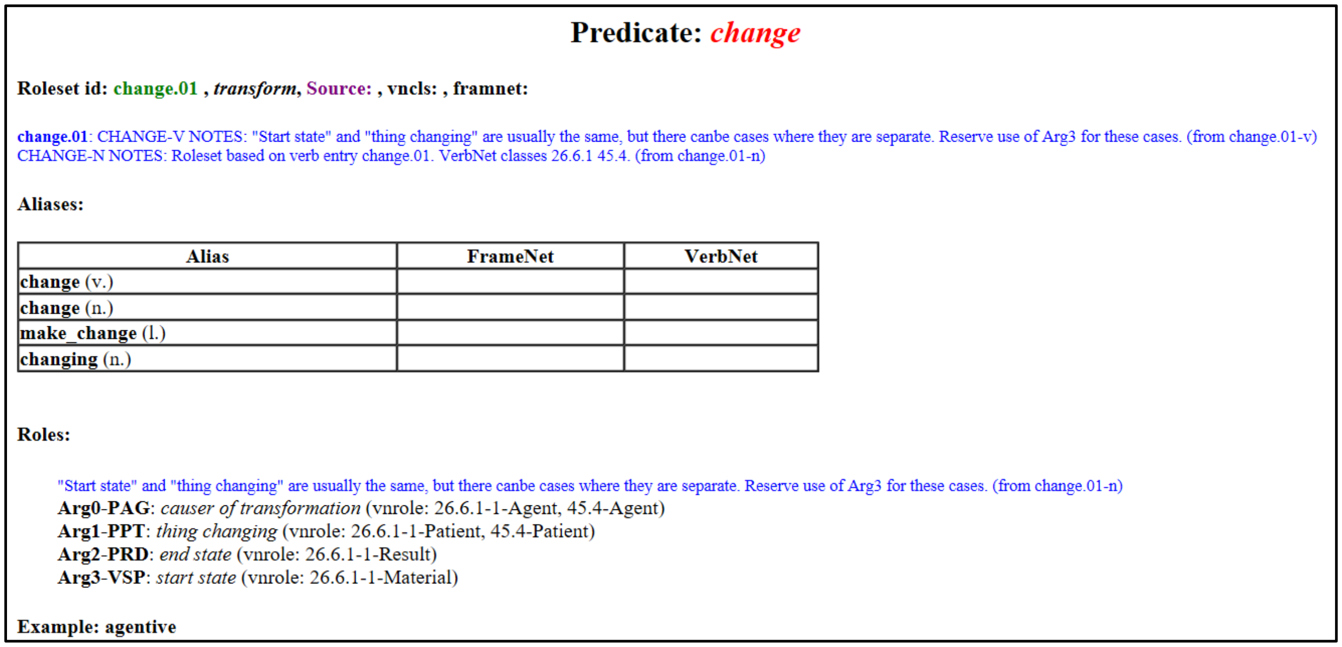


	The	user	may	either	change	the	meal	date	or	cancel	the	order
change.01	A0		AM-MOD	AM-DIS		A1						
cancel.01	A0		AM-MOD								A1	

Parsing sentence required 16ms.

ID	Form	Lemma	PLemma	POS	PPOS	Feats	PFeats	Head	PHead	Deprel	PDeprel	IsPred	Pred	Args: change.01	Args: cancel.01
1	The	_	the	_	DT	_	_	_	2	_	NMOD	_	_	_	_
2	user	_	user	_	NN	_	_	_	3	_	SBJ	_	_	A0	A0
3	may	_	may	_	MD	_	_	_	0	_	ROOT	_	_	AM-MOD	AM-MOD
4	either	_	either	_	RB	_	_	_	5	_	ADV	_	_	AM-DIS	_
5	change	_	change	_	VB	_	_	_	3	_	VC	Y	change.01	_	_
6	the	_	the	_	DT	_	_	_	8	_	NMOD	_	_	_	_
7	meal	_	meal	_	JJ	_	_	_	8	_	NMOD	_	_	_	_
8	date	_	date	_	NN	_	_	_	5	_	OBJ	_	_	A1	_
9	or	_	or	_	CC	_	_	_	5	_	COORD	_	_	_	_
10	cancel	_	cancel	_	VB	_	_	_	9	_	CONJ	Y	cancel.01	_	_
11	the	_	the	_	DT	_	_	_	12	_	NMOD	_	_	_	_
12	order	_	order	_	NN	_	_	_	10	_	OBJ	_	_	_	A1

Fig. 3. Semantic Parsing of a Software Requirement Statement using LTH [42].



Predicate: change

Roleset id: change.01 , *transform*, Source: , vncls: , framnet:

change.01: CHANGE-V NOTES: "Start state" and "thing changing" are usually the same, but there canbe cases where they are separate. Reserve use of Arg3 for these cases. (from change.01-v)
CHANGE-N NOTES: Roleset based on verb entry change.01. VerbNet classes 26.6.1 45.4. (from change.01-n)

Aliases:

Alias	FrameNet	VerbNet
change (v.)		
change (n.)		
make_change (l.)		
changing (n.)		

Roles:

"Start state" and "thing changing" are usually the same, but there canbe cases where they are separate. Reserve use of Arg3 for these cases. (from change.01-n)
Arg0-PAG: *causer of transformation* (vnrole: 26.6.1-1-Agent, 45.4-Agent)
Arg1-PPT: *thing changing* (vnrole: 26.6.1-1-Patient, 45.4-Patient)
Arg2-PRD: *end state* (vnrole: 26.6.1-1-Result)
Arg3-VSP: *start state* (vnrole: 26.6.1-1-Material)

Example: agentive

Fig. 4. SRL for Predicate Change.01 [42].

C. Set of Tokens with Annotation

The linguistic analysis information (lexical, dependency relationships, and SRL) resulting from the LTH system software tool were manually assigned to an Excel spreadsheet of tokens (for a given SRS) with their annotations. We considered only tokens that were system, actor, or use case.

As shown in Fig. 1, there are two versions of the set of tokens with annotations. The first, which is used in the training phase, contains the tokens and their linguistic annotations (lexical, SRL, and dependency relations), which take the form of a table with text values. The SRS identity of a word (token) is specified manually, forming a training set of data that can be used to train the SyAcUcNER (SVM-based), model. The textual contents of this table can be converted to a numeric form, allowing them to be handled by the SVM in the next processing step. The second version of the set of tokens and annotations is used in the testing phase and is similar to the first except that the SRS's entity is not manually assigned to each token. Instead, SyAcUcNER is responsible for performing this assignation or other word recognition of the SRS identity of a token. In both versions of the table, the contents are numerically coded and saved as a .csv file, conforming with the format required by Weka software, in which its SVM was used to perform the classification of the tokens. The coding of the word (token) was neglected, and coding only the linguistic features (lexical, SRL, and dependency relation) with their corresponding SRS.

We developed and implemented an algorithm to code and save the table of the set of tokens, as shown.

Create_Coded_Data_File (Table of tokens with their features and annotations)

```
Begin
   For all tokens in the table
  –  Code the lexical attribute field according to the
     LexicalCodeTable
  –  Assign LexicalCodeValue to its column in the
     Coded_Data Table
  –  Code the SRL attribute field according to the
     SRLCodeTable
  –  Assign SRLCodeValue to its column in the SRLData
     Table.
  –  Code the dependency relation attribute field according to
     the DepRelCodeTable
  –  Assign DepRelCodeValue to its column in the
     Coded_Data Table
  –  Code the SRS Field according to the SRSCodeTable
  –  Assign SRSCodeValue (in term of char 'c' and a
     sequence) to its column in the Coded_Data Table
   End For
   Save Coded_Data Table in a .csv file.
End
```

We used Excel software to maintain tables of the software statements, their tokens, linguistic attributes, and codes. We used the VBA function in Excel to implement the Create_Coded_Data_File algorithm. In practice, this function forms a pre-processing step ensuring that the values of the targeted attributes conform to the constraints of Weka, which is used in the next processing step.

D. SVM Data Mining Model

Based on our view of the system, use case, and actor semantics of SRS as classes, we used SVM to generate and optimize combinations of classifications for each of these SRS's semantics.

In the example shown in Fig. 5, where SVM is used for induction purposes, the training data are represented as vectors {X1, ..., Xn} in a domain D, where Xi ∊ D and their labels are represented as {Y1, …, Yn}. The vectors positioned on one side of a hyperplane would be labeled as Yα, and the vectors on the other would be labeled as Yε. The support vectors are the lying instances that closest to the hyperplane that is the decision surface [46].

Since we use SVM in this work, the training data vectors {X1 ... Xn} are required, where Xi is represented as {x1, x2, x3} in English language (El) space X ⊆ El. The labels {Y1, …, Yn} are also needed, where Yi ∈ {1,2,3}, representing {System, Actor, Use Case}. These training data were prepared using the Create_Coded_Data_File function given above. In general, SVM projects data in space (X) to a higher-dimensional feature space (f) using a Mercer kernel operator K. A set of classifiers are formed as follows [46]:

$$f(x) = \left(\sum_{i=1}^{n} \alpha_i \, K(X_i, X) \right) \quad (1)$$

In the case where K satisfies Mercer's condition, K(a,b) we can be rewritten as [47]:

$$K(a,b) = \Phi(a)\cdot\Phi(b) \quad (2)$$

where Φ: X→F, and "·" symbolizes the inner product operation.

Thus, *f* in (1) can be rewritten as:

$$f(x) = w.\ \Phi(x), \text{ where } w = \sum_{i=1}^{n} \alpha_i \, \Phi(X_i) \quad (3)$$

Consequently, the use of K enables us to implicitly project the data into space (f), which usually has higher dimensional features. SVM can then be used to map the αis, which agrees with the maximal margin hyperplane in (f). Changing kernel functions would implicitly project the data from space X into space f, where their hyperplanes agree with the decision boundaries of the more complex features in space X [47] [46]. SVM is a supervised learning method, in which a learning algorithm utilizes pre-labeled training data to develop a classification model that outlines classes and their distinguished data values. The resulting trained classification model can be used to classify new data. SVM has been extended to perform non-linear classification, multi-class classification, and regression analysis [13] [48]; therefore, is recognized as a robust classifier.

Weka (Waikato Environment for Knowledge Analysis) is a machine learning software technology that offers implementation of SVM in addition to other machine learning algorithms [49]. It is free software, licensed under the GNU General Public License, and was developed at the University of Waikato, New Zealand.

The SVM in Weka can handle numerical input data saved in an Excel file with only one worksheet, as a .csv file. The Create_Coded_Data_File VBA macro yields the Coded_Data.csv file, which contains the coded SRS classes and the value of their linguistic attributes as a table. The header of this table is the metadata of its fields, which are a1 (representing the lexical attribute code), a2 (representing the SRL attribute code), a3 (representing the dependency relation attribute), and a4 (representing the SRS class attribute code), where 'a' means 'attribute'. It is important to note that the values a4 are nominally in the form of a char c (meaning 'class') along with numbers such as c1, c2. Weka's SVM is referred to as 'SMO' in its classifier list. This stands for sequential minimal optimization, and it is an efficient optimization training algorithm for SVM [13] [49].

The training data file was loaded via the Open file command button. The attributes of the data set were displayed in the Attribute submenu, in addition to other related information about the dataset. The classes (i.e. system, actor, and use case in this study) appear in different colors in the lower right-hand corner of the Weka Explorer interface.

Weka's SVM is a Java class with certain properties, and these can be displayed by clicking the text box near the Choose command button. In this work, the properties of the SMO were set using trial and error to obtain the most accurate SVM-based NER model.

The SyAcUcNER model produced in this research is a multi-class classification model that maps input data to system, actor, or use case classes. To achieve multi-class classification with Weka's SMO (i.e., an SVM), the classification method was set to Hastie and Tibshirani's pairwise coupling (also known as '1-vs-1'). In order to achieve accurate possibility estimates, an option is used that fits the calibration models to the SVM's outputs [50] [51]. When the properties are set, the SMO is then trained, and this is achieved via the Start command button in the Classify tab of the Explorer window in Weka. The classifier output (analysis of the classification performance) is then displayed in the lower right-hand corner of the Weka Explorer window. Fig. 6 shows one of the training runs.

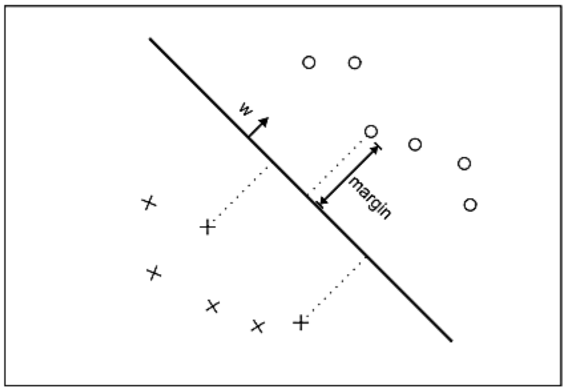
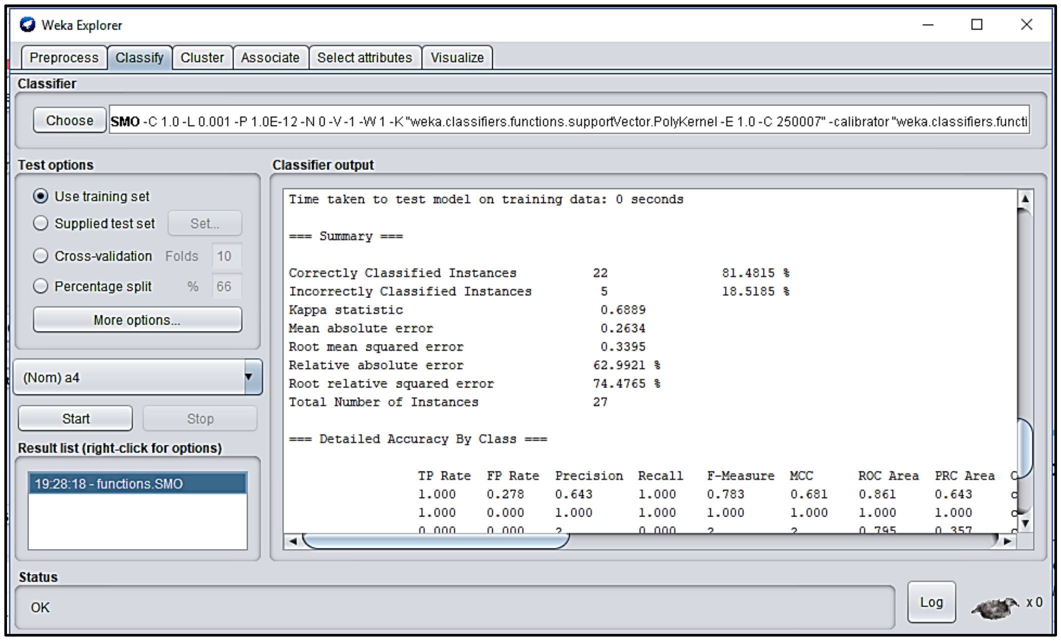


Fig. 5. A Simple SVM for Induction [46].



Classifier: SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.functi

Test options: Use training set; Cross-validation Folds 10; Percentage split % 66

(Nom) a4

Result list (right-click for options): 19:28:18 - functions.SMO

Status: OK

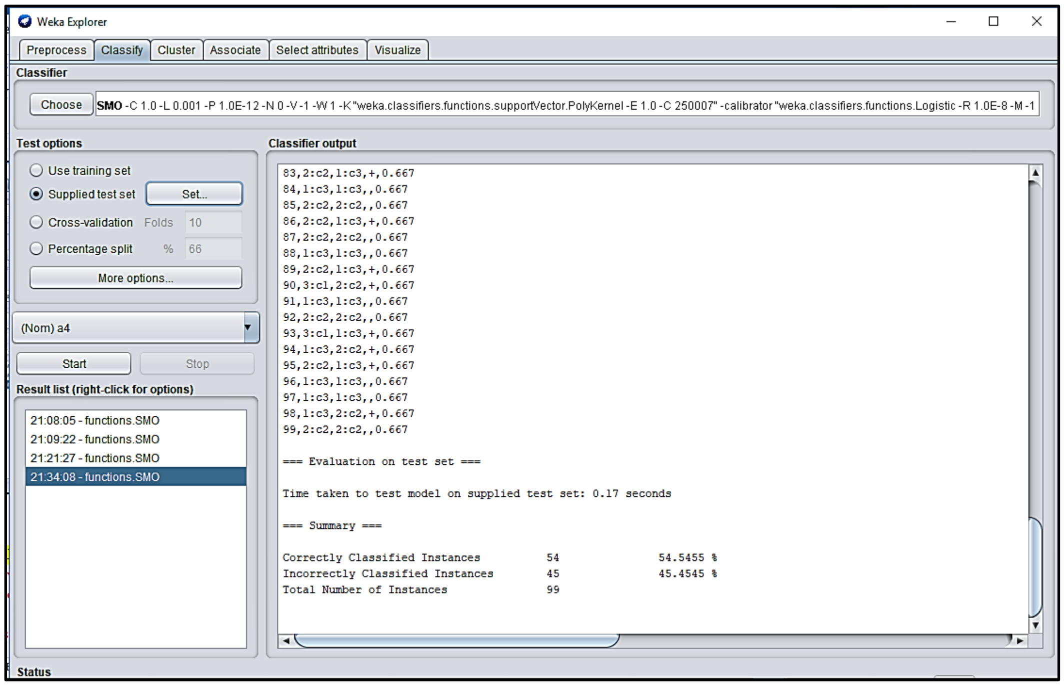
Classifier output

```
Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances          22               81.4815 %
Incorrectly Classified Instances         5               18.5185 %
Kappa statistic                          0.6889
Mean absolute error                      0.2634
Root mean squared error                  0.3395
Relative absolute error                 62.9921 %
Root relative squared error             74.4765 %
Total Number of Instances               27

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  C
                 1.000    0.278    0.643      1.000    0.783      0.681    0.861     0.643     c
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     c
                 0.000    0.000    ?          0.000    ?          ?        0.795     0.357     c
```

Fig. 6. Running (Training) Phase of Weka's SMO Function [52].

The finalized and trained SMO model is then saved to an external file so that it can be loaded later and used to make predictions using the testing data. The SyAcUcNER Model

The final trained SyAcUcNER model is a specialized named entity recognition model for software requirements engineering based on SVM. This model can then perform the classification of testing data (that have been pre-processed) in the same way as for the training data. The testing data represent the actual problems that a software requirement analyzer needs to solve. The saved SyAcUcNER model has first loaded it from its file; this is achieved by following the same steps used to save the trained model but selecting the option Load model instead of the Save model. Predictions are made for the new testing data by loading the test and then selecting the Classify tab, the Test options pane, and the Supplied test set option. The file format of the output predictions is set to .csv, and the evaluation metrics utilize each of the elements of the binomial approach (TP, FP, Recall, Precision, and F-measure) and the number of correct and incorrect predictions. When the Start command button is clicked, the predictions for each test instance are listed in the Classifier output pane. Fig. 7 illustrates a testing run for 99 instances of the testing data.



Classifier: SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8 -M -1

Test options: Supplied test set; Cross-validation Folds 10; Percentage split % 66

(Nom) a4

Result list (right-click for options):
21:08:05 - functions.SMO
21:09:22 - functions.SMO
21:21:27 - functions.SMO
21:34:08 - functions.SMO

Status

Classifier output

```
83,2:c2,1:c3,+,0.667
84,1:c3,1:c3,,0.667
85,2:c2,2:c2,,0.667
86,2:c2,1:c3,+,0.667
87,2:c2,2:c2,,0.667
88,1:c3,1:c3,,0.667
89,2:c2,1:c3,+,0.667
90,3:c1,2:c2,+,0.667
91,1:c3,1:c3,,0.667
92,2:c2,2:c2,,0.667
93,3:c1,1:c3,+,0.667
94,1:c3,2:c2,+,0.667
95,2:c2,1:c3,+,0.667
96,1:c3,1:c3,,0.667
97,1:c3,1:c3,,0.667
98,1:c3,2:c2,+,0.667
99,2:c2,2:c2,,0.667

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0.17 seconds

=== Summary ===

Correctly Classified Instances          54               54.5455 %
Incorrectly Classified Instances        45               45.4545 %
Total Number of Instances               99
```

Fig. 7. Using the Trained SyAcUcNER Model on Testing Data [52].

V. Results and Evaluation

The performance of the SyAcUcNER model and the selection of distinguishing features (lexical, SRL, and dependency relations) were assessed for a given testing data set by selecting the Supplied test set option in the Test option pane of the Classify tab in the Weka Explorer interface.

We scored SyAcUcNER's performance in terms of its accuracy, defined as the quality degree of a class achieved by the proposed model compared with the true quality degree for the same class [22]. Accuracy was quantified by calculating the ratio of the number of correctly classified cases to the total number of classified cases, and was mathematically described using the following formula [53]:

Accuracy = number of correctly classified cases/total number of cases (4)

The use of this naïve definition of accuracy to score the performance of a classification model overlooks both the real threats from the different forms of errors and the ability to be free from error depending on the distribution of the classes in a dataset while calculating the accuracy. A better analysis of the error (in terms of recognizing the types of wrong classification results) can be achieved by using a two-dimensional confusion matrix. Each row of the confusion matrix contains a forecasting class and its recorded incidence number, while each column contains the actual class and its recorded incidence number. An increase in the number of classes in the classification leads to a larger confusion matrix, causing a significant problem; this can be solved by classifying the results as either positive or negative relative to the target class, thus giving four different numbers [54] [55]:

- True positive (TP) value: The number of correct positive classifications.
- True negative (TN) value: The number of correct negative classifications.
- False-positive (FP) value: The number of incorrect positive classifications.
- False-negative (FN) value: The number of incorrect negative classifications.

As illustrated in Table III, these values are used to calculate the set of performance metrics. Finally, we made a number of train courses, each with different settings for the properties in the SMO object (Weka's SVM).

TABLE III. Performance Metrics [54] [55]

Metric	Formula
Percentage of TP value (TP rate)	TP / (TP + FN)
Percentage of FP value (FP rate)	FP / (FP + TN)
Percentage of TN value (TN rate)	TN / (TN + FP)
Percentage of FN value (FN rate)	FN / (FN + TP)
Percentage of TP to all true values (Precision)	TP / (TN + TP)
Percentage of all true results (Accuracy)	TP+TN / (TP+FN+FP+TN)
Precision & recall harmonic mean (F1 Score)	2*(Precision*Recall) / (Precision+ Recall)

We used a common agreement among the users of Weka, which is the trying of a suite of different values of kernels and C parameters could lead to the best achievement. Thus, we got good accuracy in terms of a weighted average of 76.2 percent for precision, 76 percent for recall, and 72.1 percent for the F-measure. Using this configuration, we obtained the highest F1 scores of 21.4 percent for the system entity, 82.5 percent for the use-case entity, and 76.8 percent for the actor entity. The weighted average of F1 was 72.1 percent.

VI. Conclusions, Findings, and Recommendations

In this work, we have proposed a solution to the problem of the automatic extraction of the SRS's entities: the system, the use case, and the actor as a specialized SRS NER that is called SyAcUcNER and uses the SVM to extract SRS elements from an unstructured English language textual document of user requirements. This systematic approach was inspired by the Intelligent Computer Aided Software Engineering (I-CASE) principle [56] and the known NER's function, which is the extraction of certain entities from an unstructured or semi-structured text written in NL.

The SyAcUcNER approach is implemented as software that has embedded other readymade free software tools such as the LTH system (for the extraction of NLP frame-semantic structure) and Weka (that offers SVM). This method facilitates and speeds up the development process and makes the work more robust. The proposed SyAcUcNER has been evaluated using a confusion matrix technique; we believe that this method is a realistic one since it gives the evaluation basing on a comparison with human achievement, rather than a comparison with other systems. The accuracy of SyAcUcNER can be described as good, based on a weighted average of 76.2 percent for precision, 76 percent for recall, and 72.1 percent for the F-measure. A comparison of the results from IT4RE [32], which extracts only the use case and actor, with those of SyAcUcNER, that extracts the system, use case, and actor, gives some interesting results. The best F-measure for IT4RE was 71 percent, while for SyAcUcNER, this was 72.1 percent.

The use of a new suite of linguistic properties, i.e., the lexical, SRL, and dependency relations, demonstrates the effectiveness of SyAcUcNER in reaching such good accuracy. We believe that SyAcUcNER can also be used to recognize more entities, especially if more effective NLP tools are used that can handle the linguistic problems arising from the particular text to be processed, as reported by [32]. The well-structured design of SyAcUcNER also enables it to act as a framework for similar future works. Besides, the use of Weka may allow another data mining machine to be used in this specialized NER rather than SVM. In addition to the achievements in terms of accuracy, the contributions made by this work include a new definition of an SRS-specialized NER and the use of an SVM (i.e., a data mining machine) for NLP-oriented applications at the semantic level. It should be noted that the work of [46] aimed to classify text, rather than engaging in deeper NLP tasks like SyAcUcNER, which performs semantic analysis in the SRS domain. The main contribution of this work is a framework for specialized NER applications, and hence, a general NER structure can be defined and implemented as an object for various discourses.

Last but not the least, we suggest, as future work, to consider the problem of revealing the true meaning of an entity as a complex ambiguity that may be handled by using the Relative-Fuzzy approach, as defined and used by [57].

References

[1] M. G. Georgiades and A. S. Andreou, "Formalizing and Automating Use Case Model Development," The Open Software Engineering Journal, vol. 6, pp. 21-40, 2012.

[2] Q. Stiévenart, J. Nicolay, D. M. Wolfgang, and C. D. Roover, "A general method for rendering static analyses for diverse concurrency models modular," Journal of Systems and Software, vol. 147, pp. 17-45, 2019.

[3] E. M. Jebril, A. T. Imam and M. Al-Fayuomi, "An Algorithmic Approach to Extract Actions and Actors (AAEAA)," in Proceedings of the International Conference on Geoinformatics and Data Analysis, Prague, Czech, 2018.

[4] H. A. Nassar, A. Alhroob and A. T. Imam, "An Algorithmic Approach for Sketching Sequence Diagram (AASSD)," in Proceedings of the International Conference on Advances in Image Processing, Bangkok, Thailand, 2017.

[5] I. Sommerville, Software Engineering, 10th ed., Essex, England: Pearson, 2015.

[6] R. S. Pressman and B. R. Maxim, Software Engineering: A Practitioner's Approach, 8/e, NY, USA: McGraw-Hill Global Education Holdings, LLC, 2015.

[7] G. F. Luger, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6th ed., Pearson, 2011.

[8] A. Copestake, Natural Language Processing: PartII Overview of Natural Language Processing (L90): PartIII/ACS, Cambridge, 2017.

[9] D. Jurafsky and J. H. Martin, Speech and Language Processing, vol. 3, London: Pearson London, 2018.

[10] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Cambridge, MA: MIT Press, 2016.

[11] A. Oleinik, "What are neural networks not good at? On artificial creativity," Big Data & Society, vol. 6, no. 1, pp. 1-13, 2019.

[12] R. Goswami, Selected Topics in Machine Learning, Michigan, USA: Independently published, 2018.

[13] B. Bayat, C. Krauss, A. Merceron and S. Arbanowski, "Supervised Speech Act Classification of Messages in German Online Discussions," in The 29th AAAI International Florida AI Research Society Conference, Florida, USA, 2016.

[14] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford Core NLP Natural Language Processing Toolkit," in The 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, Baltimore, Maryland, USA, 2014.

[15] T. A. S. Foundation, "Welcome to Apache OpenNLP," 2018. [Online]. Available: https://opennlp.apache.org/.

[16] V. Karkaletsis, P. Fragkou, G. Petasis, and E. Iosif, "Ontology Based Information Extraction from Text," in Knowledge-Driven Multimedia Information Extraction and Ontology Evolution, Berlin, Heidelberg, Springer, 2011, pp. 89-109.

[17] J. Li, A. Sun, J. Han and C. Li, "A Survey on Deep Learning for Named Entity Recognition," IEEE Transactions on Knowledge and Data Engineering, p. Early Access Article, 2020.

[18] P. Groth, M. Lauruhn, A. Scerri and R. D. Jr, "Open Information Extraction on Scientific Text: An Evaluation," in The 27th International Conference on Computational Linguistics, Santa Fe, New Mexico, USA, 2018.

[19] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead and O. Etzioni, "Open Information Extraction from the Web," in The 20th international joint conference on Artificial intelligence, Hyderabad, India, 2017.

[20] N. Indurkhya and F. J. Damerau, Handbook of Natural Language Processing, London, U.K: Chapman & Hall, 2010.

[21] M. Mohanan and P. Samuel, "Software Requirement Elicitation Using Natural Language Processing," in Innovations in Bio-Inspired Computing and Applications. Advances in Intelligent Systems and Computing, vol. 424, Cham , Springer, 2016, pp. 197-208.

[22] D. K. Deeptimahanti and M. A. Babar, "An Automated Tool for Generating UML Models from Natural Language Requirements," in International Conference on Automated Software Engineering, Auckland, New Zealand, New Zealand, 2009.

[23] T. H. Nguyen, J. Grundy, and M. Almorsy, "Rule-Based Extraction of Goal-Use Case Models from Text," in 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy, 2015.

[24] R. Robu and S. Holban, "A Genetic Algorithm for Classification," in International Conference on Computers and computing, Canary Islands, Spain, 2011.

[25] M. Junczys-Dowmunt, "A Genetic Programming Experiment in Natural Language Grammar Engineering," in 15th International Conference on Text, Speech and Dialogue, Brno, Czech Republic, 2012.

[26] P. More and R. Phalnikar, "Generating UML Diagrams from Natural Language Specifications," International Journal of Applied Information Systems, vol. 1, no. 8, pp. 19-23, 2012.

[27] N. Arman and S. Jabbarin, "Generating Use Case Models from Arabic User Requirements in a Semiautomated Approach Using a Natural Language Processing Tool," Journal of Intelligent Systems, vol. 24, no. 2, pp. 277-286, 2015.

[28] M. Murtaza, J. H. Shah, A. Azeem, W. Nisar, and M. Masood, "Structured Language Requirement Elicitation Using Case Base Reasoning," Research Journal of Applied Sciences, Engineering and Technology, vol. 6, pp. 4393-4398, 2013.

[29] R. Sharma, P. K. Srivastava, and K. K. Biswas, "From Natural Language Requirements to UMLClass Diagrams," in IEEE Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE), Ottawa, ON, Canada, 2015.

[30] S. Gulia and T. Choudhury, "An Efficient Automated Design to Generate UML Diagram from Natural Language Specifications," in 6th International Conference - Cloud System and Big Data Engineering (Confluence), Noida, India, 2016.

[31] W. Wan, H. Cheong, W. Li, Y. Zeng, and F. Iorio, "Automated Transformation of Design Text ROM Diagram into SysML Models," Advanced Engineering Informatics, vol. 30, no. 3, pp. 585-603, 2016.

[32] A. Al-Hroob, A. T. Imam and R. Al-Heisa, "The Use of Artificial Neural Networks for Extracting Actions and Actors from Requirements Document," Information and Software Technology, vol. 101, pp. 1-15, 2018.

[33] T. Eftimov, B. K. Seljak, and P. Korošec, "A rule-based named-entity recognition method for knowledge extraction of evidence-based dietary recommendations," PLOS ONE, vol. 12, no. 6, pp. 1-32, 2017.

[34] L. M. Berk, English Syntax: From Word to Discourse, NY, USA: Oxford University Press, 1999, p. 315.

[35] T. E. Payne, "Summary of Semantic Roles and Grammatical Relations," 2007. [Online]. Available: https://pages.uoregon.edu/tpayne/EG595/HO-Srs-and-GRs.pdf.

[36] V. Punyakanok, D. Roth and W.-t. Yih, "The importance of syntactic parsing and inference in semantic role labeling," Computational Linguistics, vol. 34, pp. 257--287, 2008.

[37] C. J. Fillmore, "Types of Lexical Information," in Semantics: an interdisciplinary reader in philosophy, linguistics and psychology, London, U.K, Cambridge University Press, 1971, pp. 370 - 392.

[38] A. T. Imam and A. J. Alnsour, "The Use of Natural Language Processing Approach for Converting Pseudo Code to C# Code," Journal of Intelligent Systems, vol. 28, no. 3, p. 362, 2019.

[39] T. Osborne and T. Gross, "Constructions are catenae: Construction Grammar meets Dependency Grammar," Cognitive Linguistics, vol. 23, no. 1, p. 163–214, 2012.

[40] M.-C. d. Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre and C. D. Manning, "Universal Stanford dependencies: A cross-linguistic typology," in Ninth International Conference on Language Resources and Evaluation, Reykjavik, Iceland, 2014.

[41] Universaldependencies.org, "Universal Dependencies," 2017. [Online]. Available: https://universaldependencies.org.

[42] D. o. C. S. Lund University, "Try the semantic role labeler," 2019. [Online]. Available: http://barbar.cs.lth.se:8081/.

[43] J. Hajič, M. Ciaramita, R. Johansson, D. Kawahara, M. A. Martí, L. Màrquez, A. Meyers, J. Nivre, S. Padó, J. Štěpánek, P. Straňák, M. Surdeanu, N. Xue and Y. Zhang, "The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages," in CoNLL '09 Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task, Boulder, Colorado, 2009.

[44] K. Hacioglu, "Semantic Role Labeling Using Dependency Trees," in 20th international conference on Computational Linguistics, Geneva, Switzerland, 2004.

[45] R. Johansson and P. Nugues, "Extended Constituent-to-Dependency Conversion for English," in The 16th Nordic Conference of Computational Linguistics (NODALIDA 2007), Tartu, Estonia, 2007.

[46] S. Tong and D. Koller, "Support Vector Machine Active Learning with Applications to Text Classification," Journal of Machine Learning Research, vol. 2, no. 1, pp. 45-66, 2001.

[47] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," Data Mining and Knowledge Discovery, vol. 2, no. 2, p. 121–167, 1998.

[48] M. Fern, D. Delgado, E. Cernadas, S. Barro and D. Amorim, "Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?," Journal of Machine Learning Research, vol. 15, no. 1, pp. 3133-3181, 2014.

[49] G. Holmes, A. Donkin and I. H. Witten, "Weka: A machine learning workbench," in Second Australia and New Zealand Conference on Intelligent Information Systems, Brisbane, Australia, 1994.

[50] S. Keerthi, S. Shevade, C. Bhattacharyya and K. Murthy, "Improvements to Platt's SMO Algorithm for SVM Classifier Design.," Neural Computation, vol. 13, no. 3, pp. 637-649, 2001.

[51] Nabble, "Explanation of SMO Parameters?," 2019. [Online]. Available: http://weka.8497.n7.nabble.com/Explanation-of-SMO-Parameters-td21768.html.

[52] R. S. R. Boddu and S. Kalyanapu, Waikato Environment for Knowledge Analysis: Data Mining Tool, Mauritius: LAP LAMBERT Academic Publishing, 2019, pp. 87-112.

[53] C. W. Ahn and R. Ramakrishna, "A Genetic Algorithm for Shortest Path Routing Problem and The Sizing of Populations," IEEE Transactions on Evolutionary Computation, vol. 6, pp. 566 - 579, 2002.

[54] Kohavi and Provost, "The Case Against Accuracy Estimation for Comparing Introduction Algorithm," in ICML '98 Proceedings of the Fifteenth International Conference on Machine Learning, 1998.

[55] D. L. Olson and D. Delen, Advanced Data Mining Techniques, 1st ed., Springer, 2008, p. 38.

[56] A. T. Imam, A. J. Al-Nsour and A. Al-Hroob, "The Definition of Intelligent Computer Aided Software Engineering (I-CASE) Tools," Journal of Information Engineering and Applications, vol. 5, no. 1, pp. 47-56, 2015.

[57] A. T. Imam, "Relative-Fuzzy: A Novel Approach for Handling Complex Ambiguity for Software Engineering of Data Mining Models," De Montfort University, Leicester, UK, 2010.