Natural Language Requirements Specification Analysis Using Part-of-Speech Tagging

Agung Fatwanto
Informatics Department
UIN Sunan Kalijaga
Yogyakarta, Indonesia
agung.fatwanto@uin-suka.ac.id

Abstract—**This paper proposes a new method for analyzing software requirements specified using a natural language. The aim of this study was to transform requirements which are specified using a natural language into a formal model. A grammatical analysis based on Part-of-Speech Tagging technique and english sentence pattern matching were adopted to analyze the software requirements in order to obtain the structure of sentences of requirements specification that will eventually be transformed into formal models.**

Keywords—software requirements specification; requirements analysis; natural language processing; part-of-speech tagging, pattern matching

I. INTRODUCTION

In general, software development projects are started with gathering stakeholders' requirements. In most cases, these requirements are specified using a natural language. Requirements specified using a natural language, unlike those which are specified using formal models, are inherently ambiguous, inconsistent, and to some extent incomplete. Eventhough it has been widely acknowledge that specifying software requirements using natural languages is problematic, these kind of practices in real projects cannot be easily resist. It is due to the fact that natural languages still become the easiest means of communication among project stakeholders.

On the other hand, most practitioners believe that using formal models can provide better requirements specification in terms of lower ambiguity and also higher degree of consistency and completeness. Considering these two concerns, this study aims to solve the particular problem by looking for a method to transform software requirements specified using natural languages to their formal models.

This paper proposes a method to transform software requirements that originally specified using a natural language to more formal specification in order to minimize ambiguity and incompleteness and, at the same time, maintaining the flexibility and simplicity of the old ways of specifying requirements using natural language. This paper is structured as follows: i) Section I discusses the background of this study, ii) Section II presents the other works on this subject, iii) Section III describes the proposed method, and iv) Section IV discusses the applicability of the proposed method.

II. LITERATURE REVIEW

A number of works on utilizing the natural language processing technique for analyzing software requirements specification have been conducted. These studies related to the translation of software requirements specified in natural language to formal specification. These studies mainly deal with the translation of software requirements specified using natural language that does not consider the dynamic aspect of software system. My proposed method, on the other hand, offers a translation of a dynamic requirements specification (written in a scenario-like format) to formal specification (in this case is object-oriented model).

A number of natural language transformation approaches for translating software requirements into formal models has been proposed. Based on their translation approach, these methods can be classified into two category: artifical intelligence (AI) based approach (or rule-based approach) and grammatical analysis based approach (or natural language processing approach).

The rule-based approach employs artificial intelligence theory for analyzing software requirements specification written in free (unconstrained) natural language. The objective of this approach is to derive formal specifications from their informal sources. Specification Acquisition From Expert (SAFE) [3], the Requirements Apprentice (RA) [4], and Specifier [5] are some of the methods applying the rule-based approach. These methods do not employ linguistic analysis in translating informal specifications into their formal models.

The natural language processing approach, on the other hand, employs grammatical knowledge to analyze the software requirements specification written in free (unconstrained) natural language. Similar as the rule-based approach, the objective of natural language processing approach is also to transform informal requirements specifications into their formal models. Conceptual Model Builder (CM-Builder) [6], Linguistic Assistant for Domain Analysis (LIDA) [7], Language Extended Lexicon (LEL) [8], Metamorfosis [9], and Natural Language Processing (NLP) [10] are some of the methods applying the natural language processing approach.

All of these methods, nevertheless, do not consider the dynamic aspects of requirements specification, which become the inherent part of software requirements specification.

978-1-4799-2975-7/13/$31.00 ©2013 IEEE

III. THE REQUIREMENTS SPECIFICATION ANALYSIS

The main purpose of this research was to develop a requirements specification analysis method based on natural language processing approach. The proposed method employs syntactic analysis (or parsing) technique. Specifically, the proposed method uses part-of-speech tagging technique, english corpus, and english sentence pattern matching.

In addition, the proposed method employs the Concern-Aware Requirements Engineering (CARE) framework as the reference for the formal models [11]. This framework specifies a formal model to represent software requirements using a scenario-like format. There are two reasons why the CARE format was chosen as the referenced for deriving formal models. *First*, the CARE format format allows the accommodation of the dynamic aspects of software requirements. *Second*, the CARE format still maintain its intuitiveness (friendly for most users) whilst keeping its formality.

A stream of text from requirements specification will be parsed using the proposed method. The parsing activity is performed to sentences that compose requirements specification. This activity is conducted in order to classify a sentence into its grammatical constructing elements.

Generally, the proposed method is comprised of five steps: i) breaking each sentences (from the requirements specification) into its composing words; ii) defining each type of these words according to the referenced english corpus (part-of-speech tagging); iii) classify the sentence into its grammatical constructing elements (sentence pattern matching); iv) Matching each sentence's composing elements with the associated CARE's sentence pattern; and v) composing the matched sentences into the CARE's scenario-like format.

At the first step, each sentences that specifies software requirements will be broken into its composing words. For example, if there is a sentence "*PDA searches elector based on family name*" hence it will be broken into: PDA | searches | elector | based | on | family | name.

A sentence that has been broken down into its composing words will become the input for the second step of the proposed method. The type of each words will be defined according to the referenced english corpus. For instance, by using the previous sentence example, the second step of the proposed method will produce the following result:

- PDA : proper noun
- searches : verb
- elector : noun
- based : verb
- on : preposition
- family : noun
- name : noun

In some cases, ambiguity in determining the type of word might be occured. According to the referenced english corpus, a word can be categorized into more than one type. For example, the word "company" can be a noun such as in the phrase "car company" and it can also be a verb such as in the sentence "I keep company you all the time". In such condition, the determination of a word's type can be assisted by utilizing statistical analysis or tree parsing based on a context-free grammar. However, the implementation of these techniques are beyond the scope of this paper.

Once each composing words has been defined its type, the process then proceed with the third step: classifying each sentence into its grammatical constructing elements. There are two challenges for conducting the third step. *First*, it is difficult to develop an automatic method to found phrases within a sentence. When this task is conducted manually, human analyst will normally be quite easy to detect which parts of a sentence are categorized as phrase(s) possibly due to the fact that they have the ability to understand the semantic (meaning) of phrases(s). This case is however beyond the capability of machines that have not been installed with such kind of intellegence. Trying to mimic human ability in understanding the semantic of phrase(s) is very difficult. Therefore, one option to develop an automatic method to found phrase(s) is using context-free grammar parsing technique. However, this is beyond the scope of this paper. *Second*, english sentences have very diverse forms. In general, there are four types of english sentences: i) declarative, ii) interrogative, iii) imperative, and iv) conditional sentence. Hence, it will be complex if we want to develop an automatic method to classify sentences into their grammatical constructing elements due to the variety of english sentences. By considering the most common type of sentences that are used for specifying requirements, the proposed method is therefore designed to focus on declarative sentences.

Another complexity which may challenge the development of an automatic parsing method is that, despite of their sentence's types variety, english sentences may also be in the form of simple or compound. Simple sentences are easier to analyze. On the other hand, compound sentences are more difficult to be comprehended. In a nutshell, compound sentences are sentences which consist of two or more simple sentences merged by conjunctions. The general rule is that a compound sentence can be formed by combining two or more simple sentences of similar type. For example, we can join two declarative sentences such as "He is a lawyer" and "He has two daughters" to become a compound sentence "He is a lawyer and he has two daughters".

Another thing that also has to be considered is that simple, declarative english sentences can be in the form of active or passive sentences. Therefore, the general rule for english sentences can be specified as follow:

$$Sentence \leftarrow \{declarative \vee interrogative \vee imperative \vee conditional\}$$

$$Sentence \leftarrow \{simple \vee compound\}$$

$$declarative \leftarrow \{active \vee passive\}$$

$$compound \leftarrow \{\forall s \in simple | compound \leftarrow s_i \cup s_{i+1} \cup \dots \cup s_n \wedge type(s_i) \equiv type(s_{i+1}) \equiv \cdots \equiv type(s_n)\}$$

In order to reduce the complexity, the proposed method was designed to focus on three aspect of sentences: i) simple sentence, ii) declarative sentence, and iii) active sentence. The argument in focusing only on simple sentences is that it would be easier to develop compound sentences splitter into their composing simple sentences and simple sentence analyzer instead of developing an automatic method for classifying the compound sentences into their grammatical constructing elements. The latter work would be effortful. Hence the feasible option is by breaking compound into simple sentences before they are analyzed. Meanwhile, the rationale for focusing on declarative sentences is that most requirements are specified in declarative sentences. Similarly, the reason for focusing on active sentences is that most requirements are also specified in active sentences.

Requirements specified in natural language will be parsed using sentence pattern matching. A (simple-declarative-active) english sentence can be analyzed to obtain its simple structure following the particular schemata:

$$\{simple \wedge declarative \wedge active\} \leftarrow subject \cdot predicate$$

The predicate must contain a verb that may or may not requires other sentence elements in order to complete the predicate. The associated other sentence elements can be in the form of object (either direct, indirect or both), adverbial, nominal or adjectival subject complement, and nominal or adjectival object complement. In cases where other sentence composing elements (such as adverb, adjective (including article), or prepositional phrase) are found, these elements are only act as the descriptor for the main elements.

The schemata can further be explained as:

$$\{simple \wedge declarative \wedge active\} \leftarrow subject \cdot verb \cdot complement *$$

An example for the (simple-declarative-active) sentence is "PDA searches elector" and "PDA is expensive".

By using the schemata, a (simple-declarative-active) sentence can then be classify into its grammatical constructing elements. For example, the sentence "*PDA searches elector based on family name*" whose composing word's types has been defined as | PDA : *proper noun* | searches : *verb* | elector : *noun* | based on : *noun* | family name : *noun*| will be classified as follow:

- PDA : subject
- searches : verb
- elector based on family name : complement

This classification can further be refined based on a set of (simple-declarative-active) english sentence patterns. The proposed method define eight (simple-declarative-active) sentence patterns as follow:

- subject . verb
- subject . verb . direct-object
- subject . verb . indirect-object . direct-object
- subject . verb . adverbial
- subject . verb . nominal-subject-complement
- subject . verb . adjectival-subject-complement
- subject . verb . direct-object . nominal-object-complement
- subject . verb . direct-object . adjectival-object-complement

The way to perform the classification refinement is by selecting the most similar pattern with the supplied sentence. It can be conducted by matching the type of each composing words with the word/phrase type of each elements within the provided sentence patterns. The word/phrase types of each elements in the provided sentence patterns are as follow:

- verb : verb
- direct-object : noun phrase
- indirect-object : noun phrase
- adverbial : adverb
- nominal subject/object compl. : noun phrase
- adjectival subject/object compl : adjective

A classification refinement to the sample sentence "*PDA searches elector based on family name*" based on the provided sentence patterns will yield the following structure:

- PDA : subject
- searches : verb
- elector : direct-object
- (based on) family name : nominal-object-compl.

The fourth step of the proposed method is matching each sentence's composing elements with the associated CARE's sentence pattern. This step can be started once a sentence has been classified into its grammatical constructing elements. The CARE's sentence pattern for specifying requirements is as follow:

$$Requirement \leftarrow agent \cdot activity \cdot target \cdot [way]$$

According to CARE's concept, a requirement represents an action, an activity performed by an agent who affects/changes one state of a target (an agent or a resource). Agent represents the entity who executes the behavior (the action prescribed by the activity). Activity describes the action taken by the agent. Target can be physical or conceptual entity. The target has a number of properties (attributes), which will be affected by the activity. Meanwhile, way defines the way in which an activity will be taken. Way can either defines a manner or a utilized instrument (a means) to take the prescribed activity. In this specification format, the agent, the activity, and the target are required whereas the way is optional.

By matching the CARE's sentence pattern with the set of eight (simple-declarative-active) english sentence patterns yields the following rule:

$$Requirement \leftarrow \{subject \cdot verb \cdot direct\_object \vee subject \cdot verb \cdot [indirect\_object] \cdot direct\_object \vee subject \cdot verb \cdot direct\_object \cdot [nominal\_object\_complement] \vee subject \cdot verb \cdot direct\_object \cdot [adjectival\_object\_complement]\}$$

This rule apparently mentioned that only four out of eight (simple-declarative-active) sentence patterns which can be matched into the CARE's sentence pattern. The proposed method is therefore focusing only in matching these four sentence patterns into the CARE's format. Based on this rule, therefore, the sample sentence "*PDA searches elector based on family name*" will be matched into the following CARE sentence pattern:

- PDA : agent
- searches : activity
- elector : target
- (based on) family name : way

The particular sample sentence shows a complete version of a requirement sentence. In most practices, however, it is common to find incomplete requirement sentences. An advantage offered by the proposed method is thus that it can reduce potentially incomplete requirements specification.

The last step of the proposed method is composing the matched sentences into the CARE's scenario-like format. There are two reasons why the CARE's scenario-like structure was adopted by the proposed method. *First*, it can incorporate the dynamic aspect of software systems specification. It is important since the study aims to accommodate the way to specify the dynamic parts of software systems. *Second*, scenario-like structure is an intuitive and familiar medium for most stakeholders that involved in the development of software systems, even if they are novices.

The proposed method defines that, along with the use of a scenario-like structure, requirements also have to be expressed in the form of event-state-action format. This type of format can be used to specify the dynamic parts of software systems. It was designed following the way to specify scenario. In this case, event is an incident that causes the activation of one or more actions. An action is an activity performed by an agent who affects/changes one state of a target (an agent or a resource). Meanwhile, a state is a condition of a particular target (an agent or a resource). Each state represents a phase in the lifecycle of a target. A state can be categorized as initial and final state. An initial state defines a pre-condition that has to be attained prior to the execution of an action. Meanwhile, a final state defines a post-condition that is attained after the execution of an action. This type of specification format describes that an action is triggered by an event whereas an event causes one or more actions. An action changes the condition of a state.

An example for software requirements specification following this format is provided in Table 1. In this table, an initial scenario for identifying elector during a particular election using PDA is specified. The requirements as specified in Table 1 form a flow of a scenario. This particular scenario specifies that when the current state of the system is "*PDA shows home screen*" and if there is an event that "*PO enters name*" then the action taken is that "*PDA will search elector*". Later, if there is an event that "*PDA found elector*" at the time when the "*PDA searching elector*", then "*[the] (PDA) [system] [will take an action to] show elector*". This flow of scenario can be easily followed from the event-state-action list specification.

TABLE I. SCENARIO SPECIFICATION FOR IDENTIFY ELECTOR

Event	State	Action		
		Agent	***Activity***	***Target***
PO enters name	{PDA} shows |home_screen|	{PDA}	searches	elector
{PDA} found elector	{PDA} searches elector	{PDA}	shows	elector
PO clicks |home_button|	{PDA} shows elector	{PDA}	shows	|home_screen|

The CARE defines a rule that needs to be imposed when using this specification format. *First*, any agent or target which represents the system being develop, such as PDA in this example, must be written between curly bracket ({…}). *Second*, any agent or target that represents the interface, such as home_button and home_screen in this example, has to be written between vertical bar (|…|). The reason why CARE imposing this rule is in order to differentiate the elements of the system from the systemic objects (agents or targets, for example: the system being develop and the interface) which will be crucial during the translation process. The translation method is, however, beyond the scope of this paper.

Requirements as specified in Table 1 is the initial version of requirements specification in the CARE format. According to the CARE method, the initial specification should be: (i) elaborated and refined, (ii) its conflicts need to be identified and resolved, and (iii) its feasibility needs to be studied. An example of the result from these revision processes can be found in [2], [12], and [13]. The revised result of the proposed method will later become more complete, more consistent, and less ambiguous.

IV. DISCUSSION

The study was aimed to develop a method to transform software requirements specified in natural language format into formal specification. Using the part-of-speech tagging technique, english corpus, english sentence pattern matching, and CARE framework as the referenced formal models, requirements specified in natural languages can be transformed in order to obtain a set of formal specifications, in this context is written in a scenario-like format.

The proposed method is an extension of work from [2], [11], [12] and [13] in which these studies have a couple of limitations. The particular limitation that is related to this study is that the method as specified in these papers can only translate

software requirements specified in a specific format (in this context is the CARE format). The capability to translate software requirements specified in any other formats is beyond the scope of these papers. The selection of the CARE format as the referenced formal models, however, is rational because of two reasons. *First*, the CARE format maintain the flexibility of specifying requirements by using a type of "constrained" natural language whilst keeping specification formality at the same time. *Second*, the CARE requirements specification format is potential to be further translated into more formal format such as into object-oriented models. This study therefore proposes a method to transform "natural" requirements specification into a more formal format (in this case is CARE format) that hopefully can be further translated into object-oriented models as specified in [13]. The contribution of this study was the development of a clear procedure on how to map requirements specified in a natural (english) language written in a flexible (loose) format into their associated formal specification. The limitation of this study was it only accommodate simple, declarative, yet active english sentences.

As a recommendation for future work, a study on how to transform requirements specified in a more flexible (loose) style (such as requirements specified in conditional or compound or passive sentences) to their associated formal specification should be conducted.

REFERENCES

[1] S. Mellor and M. Balcer, Executable UML: A Foundation for Model-Driven Architecture. Indianapolis, IN: Addison-Wesley Professional, 2002.

[2] A. Fatwanto, "Specifying Translatable Software Requirements Using Constrained Natural Language," in Proceeding of the 7th International Conference on Computer Science and Education, IEEE Computer Society, 2012, pp. 1047-1052.

[3] R. Balzer, N. Goldman, and D. Wile, "Informality in Program Specification," IEEE Trans. On Software Engineering. US, vol. 4, no. 2, March 1978.

[4] H.B. Reubenstein and R.C. Waters, "The Requirements Apprentice: Automatic Assistance for Requirements Acquisition," IEEE Trans. On Software Engineering. US, vol. 17, no. 3, March 1991.

[5] K. Miriala and M.T. Harandi, "Automatic Derivation of Formal Specification from Informal Description," IEEE Trans. On Software Engineering. US, vol. 17, no. 10, October 1991.

[6] H. Haramain and R. Gaizauskas, "CM-Builder: An Automated NL-Based Case Tool," in Proceeding of the 15th International Conference on Automated Software Engineering, IEEE Computer Society, 2000, pp. 45–53.

[7] S.P. Overmyer, B. Lavoie, and O. Rambow, "Conceptual Modeling Through Linguistic Analysis Using LIDA," in Proceeding of the 23rd International Conference on Software Engineering, USA, IEEE Computer Society, 2001, pp. 401–410.

[8] L.M. Cysneiros and J.C.S. do Praito Leite, "Non-Functional Requirements: From Elicitation to Conceptual Models," IEEE Trans. On Software Engineering. US, vol. 30, no. 5, 2004, pp. 328–350.

[9] I. Diaz, J. Sancez, and O. Pastor, "Metamorfosis: un Marco Para en Analisis de Requisitos Functionales," in Anais do Workshop en Engenharia de Requisitos, pp. 233–244, 2005.

[10] A. Montes, H. Pacheco, H. Estrada, and O. Pastor, "Conceptual Model Generation from Requirements Model: A Natural Language Processing Approach," in Proceeding of the 13th International Conference on Application of Natural Language to Information Systems, UK, Lecture Notes in Computer Science, Vol. 5039, 2008, pp. 325–326.

[11] A. Fatwanto, A Concern-Aware Requirements Engineering Framework, Canberra, AUS Ph.D Thesis: The Australian National University, 2011.

[12] A. Fatwanto, "Translating Software Requirements from Natural Language to Formal Specification," in Proceeding of the International Conference on Computational Intelligence and Cybernetics, IEEE Computer Society, 2012, pp. 148-152.

[13] A. Fatwanto, "Software Requirements Translation from Natural Language to Object-Oriented Model," in Proceeding of the International Conference on Control, System, and Industrial Informatics, IEEE Computer Society, 2012, pp.191-195.