Code Generations from Natural Language Specification with Naming Traceability

Janghwan Kim
Software Engineering Lab
Hongik University
South Korea
janghwan.kim@g.hongik.ac.kr

Ye Dong Yoon
Software Engineering Lab
Hongik University
South Korea
yedong@mail.hongik.ac.kr

Chae Yun Seo
Department of Internet
Information Communication
KIUF
Uzbekistan
chaeyun@kiuf.uz

So Young Moon
Dept. of Software and
Communications Engineering
Hongik University
South Korea
whit2@hongik.ac.kr

R. Young Chul Kim
Software Engineering Lab
Hongik University
South Korea
bob@hongik.ac.kr

ABSTRACT

Since the project is separately carried out with requirements and development, it is required to accurately analyze the requirements. In Requirement Engineering, it is very difficult to define requirement Specifications that is based on natural language for a project. We propose a template-mechanism for generating code via UML diagrams from natural language-based requirements by analyzing software requirements adapting with Abbot's textual analysis and Fillmore's semantically linguistic way. We expect this method to help develop software that is reflected by the requirements very completely. In near future, we need to generate smart codes with learning natural language based requirements.

KEYWORDS

Requirement Engineering, Abbot's heuristics, Fillmore's Case-grammar

1 INTRODUCTION

In 2022, Korea government amends Software Promotion Act such that a business with a total business size of 500 million won or more among software businesses pursuant to Article 2, No. 3 of the Software Industry Promotion Act shall be ordered separately[1]. As the separate ordering law for software design and implementation is passed, the importance of SW requirements and SW design is increasing. One important issue is how to implement source codes with satisfying customer requirements more accurately without the redundancy of requirements.

However, the methods of eliciting and analyzing the requirements are still insufficient on the software development lifecycle, which is due to the characteristics of the requirements. In the early stage of software development, many stakeholders present their own expectations and opinions about software [2]. When clients and stakeholders elicit software requirements, they incorporate their thoughts and expectations into the requirements through human language, i.e., natural language.

Therefore, Natural language based requirements may be ambiguous due to the inherent characteristics of natural language. In addition, due to the lack of understanding of the software product, requirements with different sentence types but similar semantically may be generated.

This paper proposes a method to reduce redundancy of requirements and improve ambiguity in the process of deriving requirements made in natural language. Hopefully, this method will reduce the required cost of software development.

We mention this paper as follows. Chapter 2 refers to related works and studies. Chapter 3 remarks about the code generation approach with natural language Requirements. Finally, we remark expectation and limitation with the conclusion.

2 Related Works

2.1 Fillmore's semantic analysis

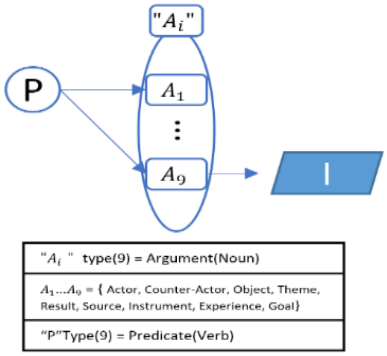

Figure 1. Fillmore's Case Grammar for Use-case Extraction

In 1968, Fillmore's initial paper proposed six cases. Fillmore defines the semantic role of words, which are elements of a sentence, along with structural analysis, and analyzes the sentence through this method [3]. Therefore, the semantic relationship between words is analyzed centering on the predicate (verb) in the sentence to indicate the relationship between the sentence elements.

Park's research proposed a new case mechanism by applying it to the case of software requirements [4]. Figure 1 shows the way that improved Fillmore's case-grammar for use-case extraction. In adapting UML mechanism with the improved case grammars, centering on the main verb is classified together with the arguments (nouns) in a sentence.

2.2 Abbott's Textual Analysis

Part of speech	Model component
Proper noun	Object
Improper noun	Class
Doing verb	Operation
Being verb	Inheritance
Having verb	Aggregation
Modal verb	Constraints
Adjective	Attribute

Figure 2. Abbott's Heuristics for Mapping Part of Speech

Abbott's Heuristics is suitable method for identifying initial objects of the software from the sentences. Figure 2 shows the mapping Part of speech to component of the high-level design model for requirement analysis[5].

3 Code Generation Approach with Natural Language Requirements

3.1 Natural Language Requirement Specification

The smart door lock system consists of parts, such as panel, bolt, database, and controller. There are two input devices, such as Panel and camera (Eye Scanner) to scan iris information. The smart door lock system must have three main cases: lock, unlock, and warn.

① An eye scanner(O) should scan(M) an iris data from the user. The iris data will be sent to the database. The panel(O) also can receive input from the user and then sends(M) the input to the controller. The controller(O) would be able to check(M) input data with the database. The database can check(M) the input from the controller with data in the table. If the input matches up with registered data, the smart door lock device(O) unlock(M) the bolt.

② When intruder start the iris information, controller recognizes that an event happens. And then, the controller checks in info database to check up if the input matches up with data. If the input does not match up with the data, then increase the error number by 1. If the error number hits 3, then it sends the error message back to the controller. Then, the device(O) warns(M) with displaying the error message. And then, the smart door lock device(O) locks(M) the bolt. By the way, the smart door lock system can lock(M) the door by pressing lock button.

Figure 3. Requirement Specification using Natural Language

In chapter 3, we explain our approach to use an example of the smart door lock system as a natural language requirement specification. Figure 3 shows the requirements document for a smart door lock system in natural language. The natural language

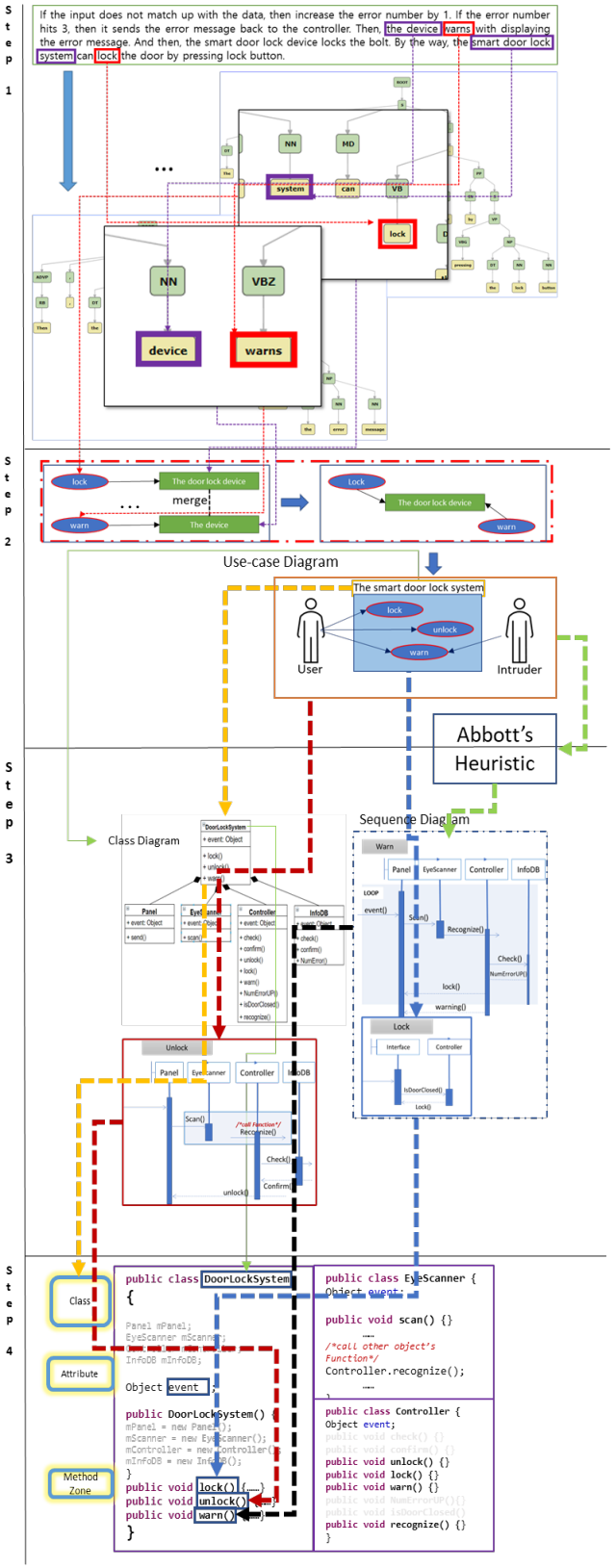

Figure 4. Code Generation Process from Natural Language with Naming Traceability

sentences usually refer to the opinions or needs of stakeholders. However, before analyzing the requirements, the meanings of sentences are ambiguous, and there are semantically similar or conflicting between sentences [5]. Therefore, these requirements documents should be redefined through analysis as requirement sentences necessary for development.

The box ①, ② shows that the case of how to lock up the device by different conditions. We put 'O' for objects and 'M' for method that we can extract from the requirement. These conditions can be used for designing the software to implement.

3.2 Code Generation Process from Natural Language

The nouns acting as subjects or objects, and the main verbs representing the main action of the requirement. In this paper, we show how to apply mechanism in each process by extracting some parts of the requirements in figure 3. We focused on the main elements of the requirement sentence. Figure 4 show the process how to generate code from natural language by tracing names [6].

Step 1: Natural Language Requirement Analysis

The first part of the document mentions the configuration of the system. In this configuration, the smart door lock system speaks the name of this software. Other nouns mentioned are objects of this system and refer to how each object in the system performs its task. We visualize the elements of a sentence using Stanford Parser to analyze the sentence structure of the requirements.

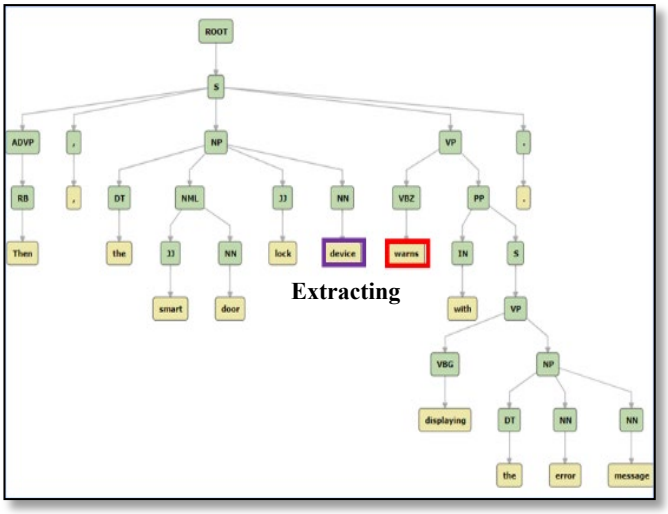

Figure 5. Requirement Structure by Stanford Parser

Figure 5 shows the structure analysis result of requirement by Stanford parser. Identifying right structure of the requirement is key to the code generation process. Stanford parser parses the requirement sentence to extract dependency between the elements of the sentence. In figure 5, warns is the main verb of the requirement sentence. The word "warn" is VBZ. It is a simple verb present tense, indicating the prototype of the verb. Stanford parser found the main verb by extracting the dependency of the sentence structure. The common noun (NN) in front of the main verb is the subject of the corresponding sentence and becomes the subject of the main verb action. This result will be used in step 2 and step 3.

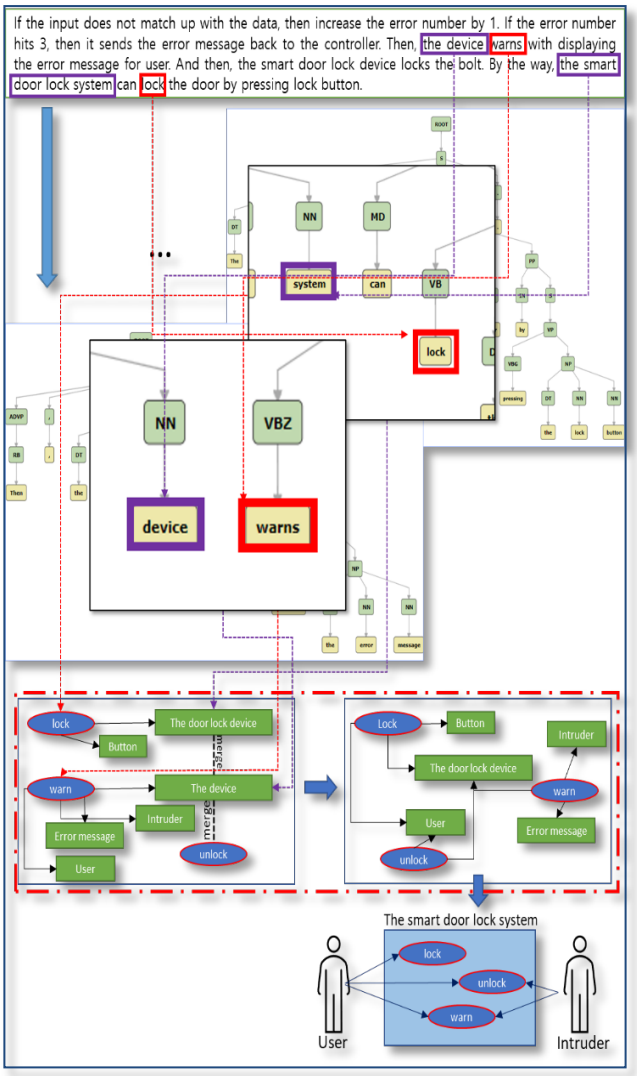

Figure 6. Applying Fillmore's Case grammar Method to Extract Usecase Diagram

Step 2: Generating Use case by Fillmore's case grammar

In Step 2, we extract the main verb and the noun that performs the main verb based on the analyzed requirements. A requirement sentence must have 2 or more nouns. Currently, we apply Fillmore's Case-grammar technique to identify the roles of each sentence element and classify the sentence elements according to the identified role. Then, the subject (noun) accompanying the main verb can be identified, and several nouns centered on the verbs are connected in the form of a node. The connected nodes are represented as in figure 6 with the verb as the center, and nouns of the same form are merged to create a use-case diagram.

Step 3: Generating Sequence Diagram by Abbott's Heuristics

In step 3, we generate a sequence diagram by using Abbott's Heuristics theory. Abbott's research emphasizes the importance of connections between entities. This is because the entity object contains more information about that software. Furthermore, according to Abbott's heuristic, association of objects is identified by examining verbs and verb phrases that indicate states [7].

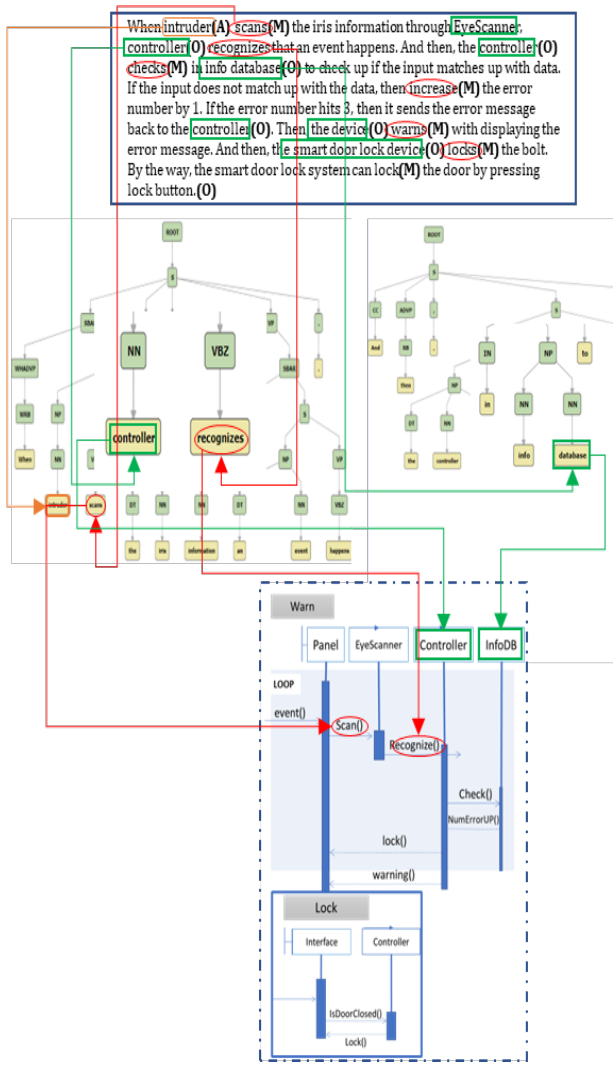

Figure 7. Applying Abbott's Heuristics Method to Extract Sequence Diagram

First, we identify the type of entity objects in the requirement statement. Based on the structural analysis results analyzed in step 1, we analyze each noun as an object and a verb as a method that performs between objects. After identifying the object of the requirement sentence, we identify the verbs between the objects.

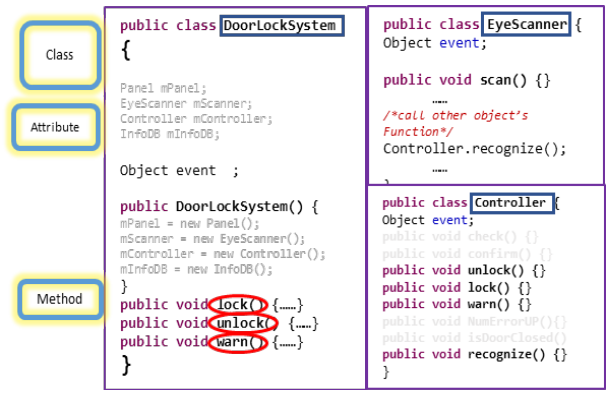

Figure 8. Skeleton Code Generation

At this time, we don't track of how the data or information comes and goes but the functions that is performed by the verbs are expressed by the name of the verb. Then, we put all together in the form of a sequence diagram, it represents as a sequence diagram as shown in the figure at the bottom of figure 7.

Step 4: Generating skeleton codes

In step 4, we create a code template based on the designs created in the previous steps. Each element created in the previous steps is used as a material (class name, function name, variable name, object name) for generating code templates. Figure 8 shows generated code template result. It contains class names, method name from objects and message flows in the sequence diagram.

4 CONCLUSIONS

Even though there is a limit of the Requirement based on natural language, our approach finds to generate high-level design from natural language. In this time, we focus on natural language-based requirements in requirement engineering, which works on how to automatically generate code with natural language based requirements. To do this, we adapt requirement engineering with Abbot's textual analysis and Fillmore's semantically linguistic way.

We propose a template-mechanism for generating code via UML diagrams from natural language-based requirements by analyzing software requirements adapting with Abbot's textual analysis and Fillmore's semantically linguistic way. We expect this method to help develop software that is reflected by the requirements very completely. In near future, we need to generate smart codes with learning natural language-based requirements.

ACKNOWLEDGMENTS

The research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF), funded by the Ministry of Education (No. 2021R1I1A3050407, No. 2021R1I1A1A01044060) and the BK21 FOUR (Fostering Outstanding Universities for Research) funded by the Ministry of Education (MOE, Korea) (No. F21YY8102068).

REFERENCES

[1] J. Jung, "A Study on the Separated Contracting for Package Software in the Public Sector". Journal of Korean Association for Regional Information Society,23(3),23-41,2020

[2] D. Mishra, A. Mishra and A. Yazici, "Successful requirement elicitation by combining requirement engineering techniques," 2008 First International Conference on the Applications of Digital Information and Web Technologies (ICADIWT), 2008, pp. 258-263, doi: 10.1109/ICADIWT.2008.4664355.

[3] C. J Fillmore, "Some problems for case grammar." Monograph series on languages and linguistics 24 (1971): 35-56.

[4] Bo Kyung Park, and R. Young Chul Kim, "Effort estimation approach through extracting use cases via informal requirement specifications." Applied Sciences 10.9 (2020): 3044.

[5] S. T. Demirel and R. Das, "Software requirement analysis: Research challenges and technical approaches," 6th International Symposium on Digital Forensic and Security (ISDFS), 2018, pp. 1-6, doi: 10.1109/ISDFS.2018.8355322.

[6] Chae Yun Seo, Janghwan Kim, and R. Young Chul Kim. "Applied Practices on Codification Through Mapping Design Thinking Mechanism with Software Development Process." KIPS Transactions on Computer and Communication Systems 10.4 (2021): 107-116.

[7] Bruegge, Bernd, and Allen H. Dutoit, "Object–oriented software engineering. using UML, Patterns, and Java." 5.6 (2009): 7.

Code Generations from Natural Language Specification with Naming Traceability