Journal of Korea Multimedia Society Vol. 22, No. 3, March 2019(pp. 386-394)
https://doi.org/10.9717/kmms.2019.22.3.386

An Automatic Construction for Class Diagram from Problem Statement using Natural Language Processing

Ahmad Zulfiana Utama†, Duk-Sung Jang††

ABSTRACT

This research will describe algorithm for class diagram extraction from problem statements. Class diagram notation consist of class name, attributes, and operations. Class diagram can be extracted from the problem statement automatically by using Natural Language Processing (NLP). The extraction results heavily depends on the algorithm and preprocessing stage. The algorithm obtained from various sources with additional rules that are obtained in the implementation phase. The evaluation features using five problem statement with different domains. The application will capture the problem statement and draw the class diagram automatically by using Windows Presentation Foundation(WPF). The classification accuracy of 100% was achieved. The final algorithm achieved 92 % of average precision score.

Key words: Natural Language Processing, Problem Statement, Class Diagram

1. INTRODUCTION

UML diagrams are the primary requirement in software design and analysis development. Class diagrams are the main component of UML[1]. Class diagrams produced from the results of analysis of existing problems. The problems are usually stated in a problem statement written in natural language. Automation of making class diagrams will help and speed up the time of analysts in designing class diagrams. Natural Language Processing(NLP) is a computer science that can analyze a free text. Many researchers developed NLP modules are to solve a specific problem domain[2].

This research will utilize the features of NLP to extract class diagrams from a problem statement. We will use library spaCy. The function of this library is to get semantic and syntactic information from problem statements. spaCY showed better in analyzing semantic and syntactic information[3-4].

We will explain the algorithm for extracting class diagrams automatically based on the design rules collected from previous research and implementation rules based on the implementation phases. The final results will be visualized by using WPF.

2. RELATED WORKS

Previous studies have explained the automation extracting a conceptual domain from a problem statement. We only represent the latest related work that is done by Elbendak[5] and Sagar[6]. Sagar's researches focus on making grammar rules to extract conceptual models from problem state-

※ Corresponding Author : Duk-Sung Jang, Address: (42601) Sungseo Campus of Keimyung University, Dalgubul-daero 1095, Dalseo-gu, Daegu, Korea, TEL : +82-53-580-5267, FAX : +82-53-580-5165, E-mail : dsjang@kmu.ac.kr
Receipt date : Jan. 5, 2019 , Revision date : Mar. 5, 2019
Approval date : Mar. 7, 2019

† (studied at) Dept. of Computer Eng., Keimyung University
(working at) Indonesian National Institute of Aeronautic and Space
(E-mail : ahmad.zulfiana@lapan.go.id)
†† Dept. of Computer Eng., School of Engineering, Keimyung University

ments. Sagar makes the criteria for writing a problem statement that would be executed in their application. These criteria are important because to clarify and minimize user mistakes in writing a problem statement. They explained class diagram extraction stages include preprocessing process, extraction of grammatical occurrences, extraction design elements (subject-object transformation rules, identifying classes/entities, identifying attributes, and relationships). The conclusion from their research is necessary to add semantic information in a conceptual model. We will do this in this study.

Elbendak[5] develop an identification of classes tool called Class-Gen. The Class-gen functions to identify objects/classes from a parsed use case description(PUCD) automatically. They are using a memory-based shallow parser(MBSP) for the preprocessing stage, and also developing design rules based on grammatical language.

The authors conclude that it is necessary to make semantic information to improve the results of accuracy. Additionally, we calculated frequency of words to define candidate of class diagram. The identification of the frequency of words related to Jiyoung research[7]. They were seeking music recommendations by analyzing the number of words from music lyrics. We make the algorithm logic from their research with our improvement such as how to determine the word phrase in a programmatical approach.

2.1 Problem Statement

A Problem statement is written in free text to specify the domain problem of the system[3]. A problem statement does not have a standard format so that the application will receive any plain text file as input containing the problem statement in software requirement. We are using a problem statement in the library management system, and online shop domain. We have been collecting problem statement from books, scientific paper, and website. The web provided the data as unannotated text.

2.2 spaCy

spaCy is Natural Language library in python. Spacy features new neural models for tagging, parsing and entity recognition. Two peer-reviewed papers in 2015 confirm this library offers the fastest syntactic parser in the world[4]. The accuracy results show the spacy version 2.x is the highest compared with CoreNLP, ClearNLP, MATE, and Turbo. Also, the spacy is the fastest for the processing time compared with other NLP libraries. The features of spaCy almost cover Natural Language models. spaCy has dependency parsing model to determine syntactic dependency relation for each word. The dependency parsing model own by NLTK library. NLTK library has been used by Sagar[6] to extract class diagram automatically. With additional features owned by spaCy will affect the results of class diagram extraction. The comparison of processing time between spaCy and NLTK library is shown in Fig. 1, in which word tokenization, sentence tokenization, and part-of-speech(POS) tagging had been evaluated.

2.3 Preprocessing

The primary goal of preprocessing part is nor-

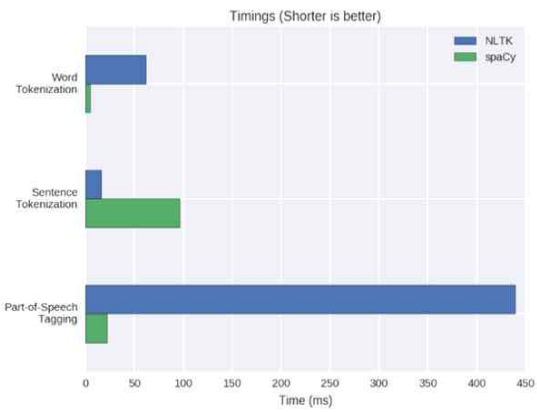

Fig. 1. Comparison between spaCy and NLTK. (https://spacy.io/).

malizing the text by performing several functions. Text normalization refers to a sequence of related functions to transform the text, so it is possible to advance processing. Preprocessing performs several tasks, such as Text Normalizing, Tokenizer, Semantic Analysis, Stemming and Lemmatization, Chunking Process, and Collection of Words.

The final results from preprocessing are a collection of words contain as tuple or dictionary format **LP**(List Preprocessing).

LP := {W, [Tag, DTag, Dep, Stm]}

Each word **W** has semantic information like POS tagger type **Tag**, description part-of-speech **DTag**, syntactic dependency relation **Dep**, and a base form of word **Stm**. This information will be used for further processing.

2.3.1 Text Normalization

Problem statements stored in a file text document and the program will load with encoding "utf-8" as string type. There are several methods in text normalized. Conceptual diagram domain has adjusted to this normalization, so this method cannot be used for all domains in NLP problems. The purpose of making this method is to improve the results of accuracy.

1) Convert text into the lower case by using lower() function. There are some for several words such as *"id"*. This word will be converted into uppercase, based on our observations from various sources the writer of the problem statement usually writes the *"id"* in lowercase. The Spacy library cannot detect *"id"* as identities when written in the lowercase form. So, we need to convert as *"ID"*.

2) Removes any type of determiner from text. The determiner will affect the spaCy when defining a word phrase in chunking process. Determiner removes by regex function. The regex function is proven to produce good results with fast processing time. The functions will remove the words in ***ignoreDeterminer*** list.

ignoreDeterminer=["the", "an", "a", "each", ...]

2.3.2 Tokenizer

The problem statement is separate into several parts of the sentences known as sentence tokenizer. Furthermore, the sentence is segmenting into words and punctuations marks. Spacy use Sentence Boundary Detection(SBD) to finding and segmenting individual sentence.

2.3.3 Lemmatization

The lemmatization is assigning words into the base form[3]. For example, the lemma of *"was"* is *"be"*, and the lemma of *"cars"* is *"car"*. In this research, the main purpose of using lemmatization is to change the plural forms into singulars. Extraction algorithms will observe a class *"Books"* and *"Book"* as same words as *"Book"*.

We have tried using regex to identify the word with the suffix -s if the word has -s suffix then remove the -s alphabet. However, English words have several words with base form suffix -s like "is", "was", "this", "plus", "less", "gas", etc. Therefore, we need a lemmatization process to changing words into base form/singular form.

2.4 Semantic Analysis

2.4.1 Part-of-Speech(POS) Tagging

Part of Speech is one of the terms in Natural Language Processing which functions to provide word type labelling or also called grammatical tagging[8]. Tagging has eight main types such as noun, verb, pronoun, adjective, adverb, preposition, conjunction, and interjection. Also, POS tagging can determine word types based on singular or plural form. An example of POS description is shown in Table 1. The complete guide for POS tag we can see in https://spacy.io/api/annotation

Spacy provides POS tagging functions by using doc object. The object will analyze semantic information automatically, one of which information

Table 1. POS tagging type

NN	Noun, singular or mass
NNS	Noun, plural
VB	Verb, base form
VBZ	Verb, past participle
VBG	Verb, gerund or present participle

Table 2. Universal Dependency Corpora

nsubj	Nominal subject
csubj	Clausal subject
obj	object
iobj	Indirect object
det	Determiner

is POS tagging. This process is the first step in understanding each word of the sentences.

2.4.2 Dependency Relation

Dependencies relation is a short description of grammatical relationships in a sentence to ease understanding and increase effectiveness[9]. It represents all sentence relationships uniformly as typed dependency relations. The dependency analyzes triples of words to produce syntactic relations[10]. The dependency analyzes triples of words to produce syntactic relations. Table 2 is represented the universal dependency corpora.

The spaCy dependency parser feature is one of the fastest and most accurate features in NLP library. spaCy uses the terms head and child to describe each word in a single arc in a dependency tree.

A dependency pair is a triple pair of grammatical related words: the main verbs in two connected clauses, a verb and each of its arguments, a noun and each of its modifiers [11].

Fig. 2 represents the dependency related words and the syntactic relation between them. Dependency relations will be used on the extraction algorithm. The *nsubj* dependency type are defined as class candidate.

2.4.3 Chunking Process

Chunking is a process of extracting short phrases from unstructured text, many possibilities can form a word phrase[12]. Word phrase is usually a noun consisting of one syllable. The syllable can consist of two or more words. The word type of each syllable can be different or same. The word type most commonly found in a phrase are noun followed by another noun word (noun phrase), for example, “marketing staff”, “phone number”, “transaction id”, etc. Knowing the noun phrase in a sentence is a crucial issue in a conceptual model. The program must be able to sort out word phrase automatically; if the program considers “credit card” as a separate word, then the system will incorrectly to identify conceptual name or attribute.

Previous researches[2-3] did not explain the design rule of detailed chunking processes. They only show the results of extraction in the form of a word phrase. We have tried number of trials (implementation) and created design rules to select word phrases.

Spacy has a function to specify word phrase

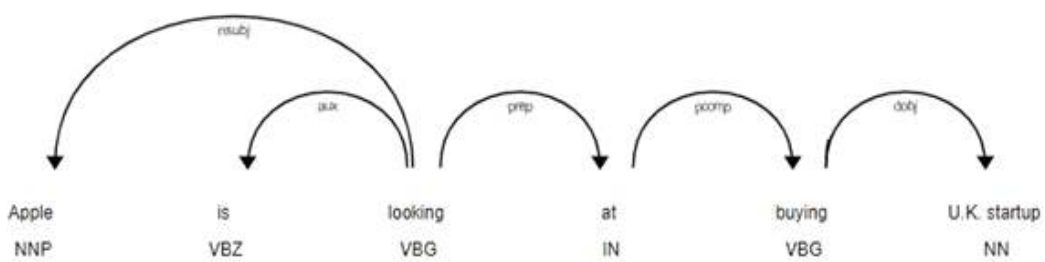

Fig. 2. The dependency visualizer (https://spacy.io/).

through a chunk object *doc.chunk*. The spacy results are not optimal if applied to the conceptual model problem. The *doc.chunk* function does not label the resulting word phrase as POS tag. This problem will complicate the extraction algorithm in determining the conceptual model. We improved the chunking process according to the conceptual model by making three criteria for word phrases for example noun phrases(NP), operation verb phrases(OVP), and verb phrases(VP). The following number is design rules to determine the three criteria.

1. Noun Phrase(**NP**): **if** all words have a noun type.
2. Verb Phrase(**VP**): **if** one of words has type verb (**V**).
3. Operation Verb Phrase(**OVP**): **OVP** is formed from two words, where the first word has a verb type(**V**), and this verb exist in ***operationWords*** list. The operationWords is formed from a collection of verbs commonly used as operations in the conceptual model. This list obtained from various sources.

operationWords = ["update", "add", "delete", "view", "create", "increase", decrease", "pay", "details", "create", "enter", "clears", "accepts", "dispenses", "prints", "print", "dispense", "processes", "display", "search", "issue", "calculate", "borrow", "remove", "return", "update", "request", "provide", ...]

Type **OVP** is very useful to determine operations in the conceptual model.

3. SUGGESTED EXTRACTING ALGORITHM

Preprocessing is an important process in text analysis. The final results of the preprocessing process(***LP***) described in section 3.3 are as follows.

LP = [('member class', ['NP', 'NOUN', 'nsubj', 'member class']),
('contains', ['VBZ', 'VERB', 'ROOT', 'contain']),
('attributes', ['NNS', 'NOUN', 'dobj', 'attribute']),
('such', ['JJ', 'ADJ', 'amod', 'such']),
('as', ['IN', 'ADP', 'prep', 'as']),
('member ID', ['NP', 'NOUN', 'pobj', 'member ID']),
('type', ['NN', 'NOUN', 'conj', 'type']),
('date', ['NN', 'NOUN', 'conj', 'date'])]

We have obtained semantic and syntactic information from sentences. The next process is to extract information based on ListWords. We want to get conceptual information which contains candidate class, attribute, and operations. This research is limited to extracting all this information.

The extracting conceptual diagrams algorithm obtained from various literature reviews (design rule) and implementation design. Implementation design is a concept that was discovered by the author when applying the design rule. Implementation design is a new algorithm found by the author.

3.1 Class Extraction

Table 3 represents grammatical rules to define the class candidate. When we implemented all rules, the results were not satisfying several class names candidate. Therefore, we add new rules called implementation rules. The following are the implementation rules which obtained from the implementation phase.

1) We made a list of words that separate class and attribute names which defined in ***CA*** *(ControlAttribute)*. The following is ***CA*** list:

CA = ["consist", "contain", "include", "has", "have", "such as", "require", "determine", "contains attributes", "based on", "including", "Based on", "described", ...]

2) If the sentence contains *"class"* word *CW*, and word before *"class"* has a ***N*** type, then will be defined as candidate class name *C* = word [*iCW* - 1].

3) If the word has POS type ***NP***, then it is verified as an entity. If the word has an ***OVP*** then it is identified as an operation.

4) Make a list of words that are ignored in ***IgW***:

Table 3. Grammatical Rules to define class candidate

No	Rules
1	All nouns are marked as class candidates[5-6].
2	Nouns followed by verb are indicated as entity types. The terms of verb are a verb formed from "to be verb"[5].
3	Nouns that have a dependency type as a subject always identify as classes[6].
4	Ignore words, if it exists in the Named Entity Recognition such as location name, person's name, and organization name[13].
5	If a sentence consists of only subjects and predicates, then the subject is identified as a class[5].
6	If the class name is related to design elements such as: "*application*, *system*, *data*, *system*, *computer*" then ignore it[3].
7	If nouns are formed as a noun phrase and the second word exists in attribute concept, then the first word is identified as the class name[13].
8	If nouns are identified, and are not existed in the above provisions, they are identified as a class candidate[5].

Table 4. Design rules to define attribute candidate

No	Rules
1	If the words exist in ***LA***(list Attribute)[1-2]. ***LA*** = ["number", "no", "code", "date", "type","volume", "birth", "id", "address", "name", ...]
2	The adjective word can be indicated as an attribute[2-3].
3	A possessive word signifies an attribute[3].
4	The word "of" is a separator between attributes and class names. Words which has a position before "of" can be indicated as attributes[3].
5	Nouns followed by the word "by"[2].

IgW = ['class', 'information', 'data', 'attribute', 'function', 'database', 'record', 'system', 'information', 'organization', 'version', 'analysis' , 'model', 'approach', 'instance', 'design', 'arrangement', 'solution', 'code', 'system code', 'use', 'scope', 'pattern', 'section', 'type', 'functions', 'display', 'management', 'period', 'time', 'entry']

3.2 Attributes Extraction

The attribute algorithm will be performing after finding a class candidate. The design rule to determine the attribute are as Table 4.

The author also enriched the criteria of design rules. The following are the implementation rules for determining candidate attributes:

1) Add semantic information for attribute categories.

2) Nouns after ***CA*** are attributes.

3) Generalize for the adjective word.

3.3 Operation Extraction

Operations are defined as verbs that linking between objects. One of these objects is a class candidate. Based on the literature study, only a few discussed the terms of operation. Table 5 represents the design rules to determine operations in a class.

The implementation rules for the operation can-

Table 5. Grammatical rules for operation candidates

No	Rules
1	Each verb is indicated as an operation[2].
2	Descriptive verb phrase implies an operation[3].

didates are words that have **OVP** type classify the operation candidates. The **OVP** type created when preprocessing in the Chunking Process.

The complete algorithm for extracting class, attribute, and operation is stated below. The description variable are **CA** Control Attribute, **iCA** index Control Attribute, **N** noun type, **NP** noun phrase type, **IgW** IgnoreWords, **C** class candidate, **LC** list class diagram, **A** attribute candidate, **LoA** list of adjective.

4. IMPLEMENTATION

The implementation section used ATM problem statements[13]. The final results visualized by using the Windows Presentation Foundation(WPF). The panel windows in Fig. 3 divided into 5 layouts. The first panel for input the problem statement and the second panel display the result of problem statement with marked colors for class name and attribute. The rest of panels are to display the re-

Function : ExtractClassDiagram
Input: preprocessed sentences
1) **If** word exist in the ***CA*** list, **then** get the index ***iCA***.
2) **If** the ***iCA*** is not empty, **then**:
   a) **Get** the word before ***iCA*** position which has POS type **N** or **NP** and the word is not including in ***IgW***.
      i. **If** the word is **NP**, **then** check every single word in NP is contain *"class"* word.
      ii. **If NP** has *"class"* word, **then** word before the *"class"* word is a class candidate **C** = word[***iCA***-1].
      iii. **If** it is not **NP**, **then** the word is ***C***.
      iv. Repeat step 2-a) until first index (decrement iteration).
   b) **Input** ***C*** into ***listClassDiagram***. **If** the ***C*** already exists in ***listClassDiagram*** **then** ignore it.
3) **If** ***iCA*** is empty, **then**:
   a) **If** word has the POS type **N** or **NP**, and not existing in ***IgW***, and has a *subject* or *object* dependency, then word identified as ***C***.
   b) **If** ***C*** exists in ***LC[C]***, **then** ignore it. **If** ***C*** not exist in ***LC[C]*** then *insert* into listClassDiagram.

Function: FindAttribute
Input : ***iCA, sentence***
1) **If** the word ***iCA*** is not empty, **then**:
   a) **Find** the word that has the type **N** or **NP**.
   b) **If** the type of POS is NP, **then**:
      i. **If** noun phrase contains the *"class"* word **then** the word is input to ***listClassDiagram[C]***
   c) **If** the type of POS is **N**, **then**:
      ii. **If** the word is **not** existing in **IgW**, and ***LC[A]*** then mark as attribute candidate. Input a candidate attribute into ***LC*** as a tuple. The **A** will be grouped based on **C**.
2) **If** ***iCA*** is empty, then:
   a) **If** a noun phrase contains *adjective* word, **then** adjust the adjective word to the semantic information ***LoA***.
      iii. **If exits in** ***LoA***, the noun phrase explodes into several words. The first word is a **C**, **if** it is **not** included in ***IgW*** and ***LC***. The adjective word is returned to the parent word contained in ***LoA*** and marked as a **A** if it is not included in ***IgW*** and ***LC***

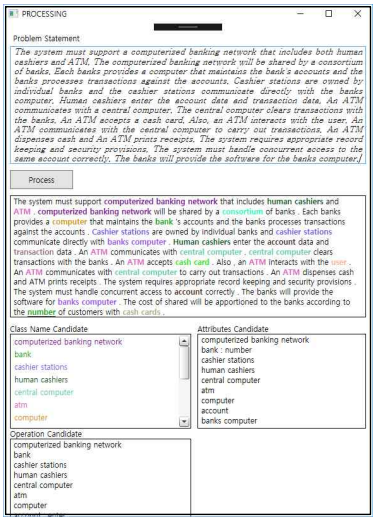

Fig. 3. Candidates of classes, attributes, and operations.

sults of the class candidate, attribute and operations. Fig. 4 shows the automatically constructed class diagram which are drawn automatically in canvas. Users can add new class diagrams, delete or update an existing diagram manually.

5. RESULTS AND EVALUATION

The evaluation matrices commonly used for machine learning field, namely Recall and Precision. We used five problem statement, and performed our algorithm. Table 6 shows recall score of 100% for all problem statements. While for precision percentages not all problem statements reach 100%, the average precision score of 92%. This is caused the algorithm detects a new class diagram known as over-specification.

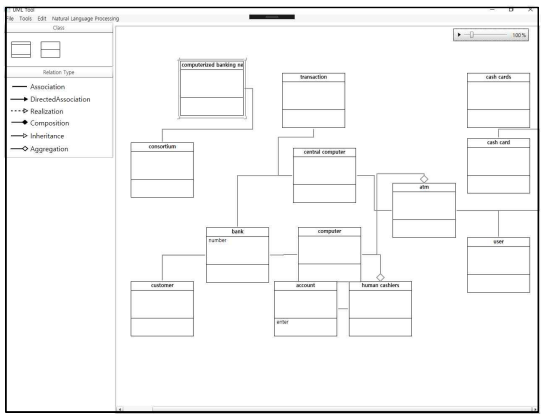

Fig. 4. Automatic constructed class diagram.

Table 6. Recall and precision

Case Study	Recall(%)	Precision(%)
ATM[14]	100	93
LMS[15]	100	100
LMS[16]	100	100
OS[17]	100	90

6. CONCLUSION

The probability of writing a problem statement as a free text is very high. Therefore it is necessary increase semantic information in a conceptual diagram to produce better results of precision. The latest NLP library(spaCy) and new algorithm developed by authors can improve recall and precision score. Determining the word phrase is significant, further research about word phrase extraction is needed.

REFERENCE

[1] D. Berardi, D. Calvanese, and G.D. Giacomo, "Reasoning on UML Class Diagrams," *Artificial Intelligence,* Vol. 168, No. 1-2, pp. 70-118, 2015.

[2] A. Purwarianti, A. Andhika, A.F. Wicaksono, I. Afif, and F. Ferdian, "InaNLP: Indonesia Natural Language Processing Toolkit, Case Study: Complaint Tweet Classification," *Proceeding of International Conference on Advanced Informatics Concepts, Theory, and Application,* pp. 1-5, 2016.

[3] R. Jiang, R.E. Banchs, and H. Li, "Evaluating and Combining Name Entity Recognition Systems," *Proceeding of the Sixth Named Entity Workshop*, pp. 21-27, 2016.

[4] Explosion AI Team, Fact and Figures, https://

spacy.io/usage/facts-figures (accessed Dec., 1, 2018).

[5] M. Elbendak, P. Vickers, and N. Rossiter, "Parsed Use Case Descriptions as a Basis for Object-oriented Class Model Generation," *Journal of Systems and Software,* Vol. 84, No. 7, pp. 1209-1223, 2011.

[6] V.B. Sagar and S. Abirami, "Conceptual Modeling of Natural Language Functional Requirements," *Journal of Systems and Software*, No. 88, pp. 25-41, 2014.

[7] S. Jiyoung and S. Yongtae, "Music Lyrics Summarization Method Using TextRank Algorithm," *Journal of Korea Multimedia Society*, Vol. 21, No. 1, pp. 45-50, 2018.

[8] H. Schmid, "Probabilistic Part-of-speech Tagging Using Decision Trees," *Proceedings of International Conference on New Methods in Language Processing,* pp. 44-49, 1996.

[9] S. Riezler, T.H. King, R.M. Kaplan, R. Crouch, J.T. Maxwell III, and M. Johnson, "Parsing the Wall Street Journal Using a Lexical-functional Grammar and Discriminative Estimation Techniques," *Proceeding of Annual Meeting on Association for Computational Linguistic,* pp. 271-278, 2002.

[10] K. Lindén, "A Probabilistic Model for Guessing Base Forms of New Words by Analogy," *Proceeding of International Conference on Intellegent Text Processing and Computational Linguistic,* pp. 106-116, 2008.

[11] Stanford Typed Dependencies Manual, https://nlp.standford.edu/software/dependencies_manual.pdf (accessed Dec., 15, 2018).

[12] S. Federici, S. Montemagni, and V. Pirrelli, "Shallow Parsing and Text Chunking: a View on Underspecification in Syntax," *Proceedings of ESSLLI'96 Workshop on Robust Parsing,* pp. 35-44, 1996.

[13] M.Z. Alksasbeh, T.A. Alramadin, and K.A. Alemerien, "An Automated Use Case Diagrams Generator from Natural Language Requirements," *Journal of Theoretical and Applied Information Technology,* Vol. 95, No. 5, pp. 1182-1190, 2017.

[14] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W.E. Lorensen, *Object Oriented Modeling and Design*, Prentice Hall, New Jersey, 1991.

[15] E.S. Btoush and M.M. Hammad, "Generating ER Diagrams from Requirement Specifications Based on Natural Language Processing," *International Journal of Database Theory and Application,* Vol. 8, No. 2, pp. 61-70, 2015.

[16] UML Activity Diagram For Library Management System, http://www.javaengineeringprograms.com/umldiagrams-for-library- management-system/ (accessed Dec., 15, 2018).

[17] Online Shopping, https://www.uml-diagrams.org/examples/online-shopping-domain-uml-diagram-example.html (accessed Dec., 15, 2018).

Ahmad Zulfiana Utama

Ahmad Zulfiana is a junior researcher in space science center at Indonesian National Institute of Aeronautics and Space. He is candidate master degree in Keimyung University. His areas of interest include System Information Development, Database, Visualization and Natural Language Processing.

Duk-Sung Jang

He received the BS degree in computer engineering from Kyung pook National Univ., in 1979. He also obtained the MS and PhD degrees in computer engineering from Seoul National Univ., in 1981 and 1988 respectively. He is currently a professor with the Dept. of Computer Eng. at Keimyung Univ. His research interests include Compiler and Natural Language Processing.