Extracting UML Class Diagrams from Software Requirements in Thai using NLP

Mathawan Jaiwai, Usa Sammapun*
Department of Computer Science, Faculty of Science
Kasetsart University, Bangkok, Thailand
Email: {g5617400606,fsciusa}@ku.ac.th

***Abstract*—In software development, requirements, normally written in natural language, are documents that specify what users want in software products. Software developers then analyze these requirements to create domain models represented in UML diagrams in an attempt to comprehend what users need in the software products. These domain models are usually converted into design models and finally carried over into classes in source code. Thus, domain models have an impact on the final software products. However, creating correct domain models can be difficult when software developers are not skilled. Moreover, even for skilled developers, when requirements are large, wading through all requirements to create domain models can take times and might result in errors. Therefore, researchers have studied various approaches to apply natural language processing techniques to transform requirements written in natural language into UML diagrams. Those researches focus on requirements written in English. This paper proposes an approach to process requirements written in Thai to extract UML class diagrams using natural language processing techniques. The UML class diagram extraction is based on transformation rules that identify classes and attributes from requirements. The results are evaluated with recall and precision using truth values created by humans. Future works include identifying operations and relationships from requirements to complete class diagram extraction. Our research should benefit Thai software developers by reducing time in requirement analysis and also helping novice software developers to create correct domain models represented in UML class diagram.**

***Index Terms*—software engineering; requirement analysis; UML; class diagram; NLP**

I. Introduction

Requirement analysis is usually performed at the early stage of the software development life cycle to understand what users need and expect in a software product. Requirements are often in a textual format written in a natural language since it is easy for both developers and users to have mutual understanding. Thus, one task in requirement analysis is to create visual representations of the requirements so that developers can understand requirements better. Domain models are one type of visual representations created during requirement analysis and usually in the form of Unified Modeling Language (UML) class diagram. Class diagrams represent entities in software comprising of classes, attributes, operations and relationships between classes. The domain models are then passed on to design models, source code, and finally the software product.

* *Corresponding author*

978-1-5090-4834-2/17/$31.00 ©2017 IEEE

However, creating correct and complete domain models, especially class diagrams, depends on experiences of developers and may require additional knowledge in specific problem domains. For example, if software is to be used in banks, financial knowledge may be required. It also takes time since software developers need to read through all requirements to create complete domain models. Moreover, for large requirements, in addition to taking too much time, it can cause developers to produce errors in domain models or miss important information in requirements.

Therefore, various researches have investigated techniques to extract UML class diagrams from requirements automatically or semi-automatically. One technique is to have requirements written in specific patterns such as an IF-THEN pattern or in very simple sentences so that the extraction can be done automatically with high accuracy. However, this approach is not natural to users nor expressive enough to describe requirements. Other approaches allow requirements to be written in natural languages and apply natural language processing (NLP) techniques in various steps to analyze requirements such as part-of-speech tagging. This approach is more convenient for both users and developers since they do not have to change the way they write requirements, but it has a downside since it is not as accurate and may need user's helps to complete some steps.

Existing researches in extracting UML diagrams from requirements are mostly for English language. This paper wants to investigate approaches that can analyze requirements written in Thai and transform them into domain models represented by UML class diagrams, in hope to support Thai software developers in the object-oriented analysis task. The analysis employs natural language processing tools and techniques such as LexTo [1] and NLTK [2] to segment Thai words and tag requirement texts with parts of speech. Words tagged with specific parts of speech such as some types of nouns and verbs are then used to construct class diagrams. Our research studies class construction rules in English, tailors some of the rules to Thai language, and add some rules appropriate for Thai. The work is ongoing where we currently use the NLP tools to automatically process requirement texts and manually apply the rules to extract classes and attributes. We evaluate whether the rules are effective using precision and recall. Our next steps include creating rules to extract behaviors or operations and relationships between classes and making the

extraction process more automatic. Moreover, since creating class diagrams may require knowledge of specific problem domains, we want to develop Thai vocabulary corpus for different application domains such as finance to help increase accuracy in extracting class diagrams.

The remaining of this paper is organized as follows. Section II explains related work. Section III presents theories and techniques used in this paper. Section IV describes details of our proposed approach. Section V discusses the evaluation results and future works. Finally, a conclusion is presented in Section VI.

II. Related Work

There are various researches aiming to analyze software requirements written in English to extract classes, attributes, operations, and relationships to generate UML class diagrams.

Most of the researches [3]–[7] perform similar analysis by first pre-processing requirement texts using NLP techniques such as stemming, part-of-speech tagging, parse tree generation, and morphological analysis. They then consider nouns as candidate classes and apply linguistic rules to select classes, attributes, and relationships from requirement texts tagged with parts of speech.

Some works such as researches by Deeptimahanti and Sanyal [3], [4] also generate other UML diagrams. For their first work [3], UML use case diagrams can also be generated. Their subsequence paper [4] presents a technique to also generate collaboration diagrams from English requirements. In addition, they normalize requirements into simple sentences in the form of "Subject-Predicate" or "Subject-Predicate-Object" before linguistic rules are applied to facilitate UML diagram generation. The two works are semi-automatic.

For the research by Herchi and Abdessalem [5], in addition to employing NLP techniques and building linguistic rules to generate UML class diagrams from requirements, domain ontologies are also used to refine results to help eliminate irrelavant elements. Researches by More and Phalnikar [6] and Shinde et al. [7] also implement tools with UI to generate UML class diagrams.

Elbendak et al. [8] present different works by analyzing use case descriptions instead of normal requirement texts. Their work parses use case descriptions to extract nouns, verbs, adjectives and adverbs and uses them to identify classes, attributes, and relationships. Afterward, class diagrams can be refined manually to improve accuracy.

III. Background

This section describes theories and technologies required to analyze requirement texts written in Thai and extract UML class diagrams. The theories and technologies include word segmentation, part-of-speech tagging, and UML class diagrams.

A. Word Segmentation

Natural language processing of Thai texts has several challenges especially in word, phrase and sentence boundary identification because a Thai sentence is written without spaces between words in contrast to English which has spaces between words. Therefore, processing Thai texts often needs word segmentation to create word boundaries. Several researches have investigated techniques and created tools to segment words such as LexTo [1] and SWATH [9]. Our work chooses LexTo.

B. Part-of-Speech Tagging

Similar to the works that extract class diagrams from English texts, we also consider nouns as candidate classes and verbs as candidate relationships. Therefore, each word in requirement texts needs to be tagged with a part of speech (POS) indicating whether the word is a noun, a verb, an adjective, an adverb, a preposition, a conjunction, an interjection, for instance. There are several tools available to help tag parts of speech, usually requiring a corpus to learn parts of speech before being able to tag words correctly. We have chosen Natural Language Toolkit (NLTK) [2] since it allows adding any corpora from any languages where we use two Thai corpora, ORCHID [10] and NAiST [11].

C. UML Class Diagram

To develop software, UML modeling is often done to help analyze requirements and design software. When using UML class diagrams to analyze requirements, the diagrams are usually used to create conceptual models or domain models representing entities in requirements. The models compartmentalize requirements into visual forms and therefore help developers comprehend tedious details of requirements easier. The domain models in the form of UML class diagrams are then expanded during design and carried over into source code and final products.

When developers create UML class diagrams, they usually follow general rules where class names and attributes are nouns where relationships and operations are verbs. However, since there are many nouns in the requirements, developers must know how to identify which nouns should be class names or attributes and which verbs should be relationships or operations. Experienced developers would be able to extract correct classes whereas novice ones may struggle and make mistakes. Our work wants to facilitate the extraction of classes, attributes, relationships, and operations.

IV. Approach

The ultimate goal of our work is to analyze requirements written in Thai language and extract class diagrams by identifying classes, attributes, relationships, and operations. The current stage of our work is where we employ NLP tools to automatically process requirement texts and manually apply the rules to extract classes and attributes. Our next steps are to apply rules to extract relationships and operations as well as making the process to be as automatic as possible. Heuristic rules may be applied to help increase accuracy as well. We believe our work can benefit Thai software developers by reducing time in generating class diagrams from requirements

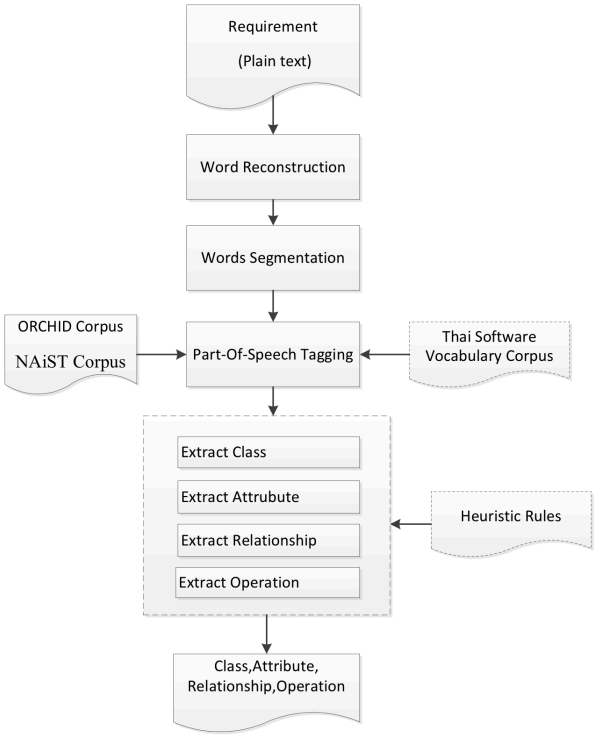

Figure 1. Overall process of the proposed method

written in Thai and also help novice developers learn the relationships between requirements and class diagrams.

Our work consists of four main steps as shown in Figure 1. First, requirements written in Thai in a plain text format is preprocessed and reconstructed. Second, texts are segmented into words. Third, parts of speech are tagged for each word. Lastly, class diagrams are extracted using heuristic rules presented later in this section.

A. Word Reconstruction

In this first step, requirements are preprocessed and reconstructed semi-automatically where texts are splitted into individual sentences and misspelled words are corrected. Since requirements may contain English words, these English words are converted into Thai words. Moreover, some Thai words derived from English words may have different spellings such as the words "แอปพลิเคชัน" and "แอพพลิเคชั่น" where both refer to an application or the words "เบราว์เซอร์" and "บราวเซอร์" where both refer to a web browser. This process chooses a correct spelling and changes all others into this chosen one. After this preprocess and reconstruction step, the word segmentation tool should process more accurate segmentation.

B. Word Segmentation

After preprocessing and reconstructing, the LexTo tool is applied to perform word segmentation of the preprocessed requirement texts. In Table I, the first column displays sample sentences in a requirement, and the second column displays results of word segmentation by LexTo.

Table I
Example of requirements for borrowing a book

Thai Sentence	Words Segmented	Tagged with POS
บรรณารักษ์ใส่เลขหนังสือที่ต้องการยืม (A librarian inputs an ID of a book to be borrowed.)	บรรณารักษ์|ใส่|เลข|หนังสือ|ที่|ต้องการ|ยืม	บรรณารักษ์/NCMN| ใส่/VACT| เลข/NCMN| หนังสือ/NCMN| ที่/PREL| ต้องการ/VACT| ยืม/VACT
ระบบตรวจสอบเงื่อนไขการยืมหนังสือของสมาชิก (A system checks a condition for borrowing a book by a member.)	ระบบ|ตรวจสอบ|เงื่อนไข|การ|ยืม|หนังสือ|ของ|สมาชิก	ระบบ/NCMN| ตรวจสอบ/VACT| เงื่อนไข/NCMN| การ/FIXN| ยืม/VACT| หนังสือ/NCMN| ของ/RPRE| สมาชิก/NCMN
ระบบตรวจสอบเงื่อนไขสถานะของหนังสือ (A system checks a condition of a book status.)	ระบบ|ตรวจสอบ|เงื่อนไข|สถานะ|ของ|หนังสือ	ระบบ/NCMN| ตรวจสอบ/VACT| เงื่อนไข/NCMN| สถานะ/NCMN| ของ/RPRE| หนังสือ/NCMN

C. Parts of Speech (POS) Tagging

To process natural languages, part-of-speech tagging is considered a basic step. This step tags each word in a sentence with different categories. The categorization of parts of speech can vary depending on how precisely one wants to tag. In this paper, parts of speech are divided into seven categories: (1) nouns, (2) pronouns, (3) verbs, (4) adverbs, (5) prepositions, (6) conjunctions, (7) interjections.

Various tools provide part-of-speech tagging where we choose NLTK using NAiST Corpus [11] and ORCHID Corpus [10]. The third column in Table I demonstrates how words in requirement sentences are tagged with parts of speech where 'NCMN' is a common noun, 'VACT' is a verb showing actions, 'PREL' is a pronoun that helps connect two sentences, 'FIXN' is a prefix of a noun, and 'RPRE' is a preposition.

D. Class Extraction

This step uses the output of the POS tagging step and applies the following rules to extract class names and attributes.

1) Rules to extract class names: Eight rules described below are used to extract class names.

- **Class-Rule 1**: Common nouns that appear in requirements are candidate class names where other rules must also be applied to finalize class names. For example, our case studies have following common nouns: หนังสือ (book), รหัส (ID), ชื่อ (name), นักเรียน (student).
- **Class-Rule 2**: A class name can be a word starting with "การ" which makes a verb into a noun, similar to how English adds *-ing* after a verb. For example, a sentence "การยืมหนังสือจะต้องเป็นสมาชิกแล้วเท่านั้น" which means "borrowing a book can only be performed by a member" contains the

word “การยืม” (borrowing). Hence, “การยืม” (borrowing) can be a candidate class.

- **Class-Rule 3**: Some common nouns relating to software environment, design, implementation and logic are not considered as classes. Examples are ฐานข้อมูล (database), ระบบ (system), สารสนเทศ (information), เว็บไซต์ (website), โปรแกรม (program), เงื่อนไข (condition).
- **Class-Rule 4**: A common noun that appears as a subject in a simple sentence is considered a class. For example, a sentence “สินค้าส่งซ่อม” which means “product is sent to be repaired” has “สินค้า” (product) as a subject, and therefore a product can be a candidate class.
- **Class-Rule 5**: A common noun referring to a person or an actor can be a candidate class such as สมาชิก (member), ลูกค้า (customer), บรรณารักษ์ (librarian).
- **Class-Rule 6**: A common noun that represents a material or a tangible object in a business activity is not considered as a class. Examples are ผลิตภัณฑ์ (goods), อุปกรณ์ (tool), เครื่องจักร (engine), เครื่องมือ (equipment), วัสดุ (material).
- **Class-Rule 7**: When two common nouns are written next to each other, the second noun can be considered as a candidate class (whereas the first noun can be considered as an attribute). For example, “ชื่อนักเรียน” (student’s name) has two nouns next to each other where the second noun “นักเรียน” (student) can be a candidate class.
- **Class-Rule 8**: When encountering two common nouns with similar meanings, we choose only one noun by keeping the more descriptive one. For example, the words “เจ้าหน้าที่” (officer) and “พนักงาน” (employee) have similar meanings in a course registration domain. We choose “เจ้าหน้าที่” (officer) instead of “พนักงาน” (employee).

2) Rules to extract attributes: For attribute extraction, the following four rules are applied.

- **Attribute-Rule 1**: The word that appears before an attributive verb can indicate an attribute. For example, in a sentence “สินค้าขนาดใหญ่จะมีค่าใช้จ่ายเพิ่ม” (“a large-sized product will have an additional cost”) has the word “ใหญ่” (large) that is an attributive verb, and therefore the word appearing before it which is “ขนาด” (size) indicates an attribute.
- **Attribute-Rule 2**: Common nouns in the following list can indicate an attribute: รหัส (ID), ลำดับ (sequence), ชนิด (type), หมายเลข (No.), จำนวน (quantity), วันที่ (date), เวลา (time).
- **Attribute-Rule 3**: The word that appears after the keyword “ระบุ” (indicate, specify) can be an attribute. For example, in a sentence “พนักงานระบุสถานะการจอง” (“an employee specifies a reservation status”), the word “สถานะ” (status) appears after the keyword “ระบุ”. Hence, the word “สถานะ” (status) can be a candidate attribute.
- **Attribute-Rule 4**: When two common nouns are written next to each other, the first noun can be considered as an attribute. For example, “ที่ตั้งสินค้า” (product’s location) has the first word as “ที่ตั้ง” (location) and thus “ที่ตั้ง” can be a candidate attribute.

Table II
Classes and attributes extracted from the example requirement

Requirements	Classes	Attributes
A librarian inputs an ID of a book to be borrowed. บรรณารักษ์/NCMN| ใส่/VACT| เลข/NCMN| หนังสือ/NCMN| ที่/PREL| ต้องการ/VACT| ยืม/VACT	บรรณารักษ์ (librarian), หนังสือ (book)	เลข (ID)
A system checks a condition for borrowing a book by a member. ระบบ/NCMN| ตรวจสอบ/VACT| เงื่อนไข/NCMN| การ/FIXN| ยืม/VACT| หนังสือ/NCMN| ของ/RPRE| สมาชิก/NCMN	การยืม (borrowing), หนังสือ (book), สมาชิก (member)	-
A system checks a condition of a book status. ระบบ/NCMN| ตรวจสอบ/VACT| เงื่อนไข/NCMN| สถานะ/NCMN| ของ/RPRE| หนังสือ/NCMN	หนังสือ (book)	สถานะ (status)

Table III
Evaluation results

Extraction	Recall	Precision
Class	0.90	0.85
Attribute	0.78	0.77

Table II shows the results of applying the class and attribute extraction rules to the sentences in Table I.

V. Result

We have applied our approach to 13 case studies where these case studies are taken from (1) published object-oriented analysis and design books that provide requirements and domain models and (2) requirements appeared in undergraduate senior projects from Department of Computer Science, Faculty of Science, Kasetsart University. These case studies are textual requirements in various application domains such as an airline flight system, a library system, and a rental system.

A. Evaluation Results

The class and attribute extraction performed by applying rules in Section IV is then compared with class diagrams in the books when requirements are taken from the books or extracted by one expert when requirements are taken from undergraduate senior projects. Table III shows recall and precision of class extraction and attribute extraction.

B. Discussion

As shown in Table III, recall and precision for class extraction are 0.90 and 0.85 while recall and precision for attribute extraction are lower at 0.78 and 0.77. It can be concluded that class extraction is more accurate than attribute extraction. Based on the number of rules used to extract classes and attributes, the results are not too surprising. A higher number of class extraction rules can weed out nouns that are not classes better. Therefore, to improve accuracy on both class extraction and attribute extraction, more rules can be added. However, adding too many rules can instead eliminate actual classes or attributes leading to lower accuracy. Therefore, rules must be constructed and chosen wisely. It can be done by performing more experiments on more requirements from various application domains and analyzing the results to fine-tune extraction rules.

C. Future Works

As for future works, extraction of relationships and operations have to be done, and the process should be implemented so that the extraction can be more automatic. To make the process automatic, specific dictionaries must be created to collect words relating to software design and materials so that Class-Rule 3 and Class-Rule 6 can be done automatically. An additional dictionary collecting common nouns referring to persons is also needed for automatic processing of Class-Rule 5. Moreover, a WordNet or a synonym resource is also needed to help compare nouns with similar meanings used in Class-Rule 8. Similarly, lexical resources are needed for the attribute rules.

Thus, making the entire process more automatic and more accurate requires several unavailable lexical resources. We would like to create some of these lexical resources ourselves such as specific dictionaries. However, some resources such as a WordNet or a synonym resource require linguists and may not be possible for us to create them ourselves.

In any case, having an approach to help Thai developers generate UML class diagrams should provide benefits by reducing time spent in creating diagrams and minimize errors. Novice developers or even software development students can learn from the class diagram extraction as well.

VI. Conclusion

This paper aims to extract UML class diagrams from requirements written in Thai by employing NLP tools and applying various extraction rules. The process starts by pre-processing requirement texts and segmenting them into words. These words are then tagged with parts of speech where our extraction rules analyze the POS-tagged words to identify class names and attributes. Thirteen case studies have been applied with the extraction rules, and the results are evaluated with recall and precision. Future works include extraction of relationships and operations as well as making the process more automatic and accurate. Our work helps support Thai developers by reducing time and minimizing errors in generating class diagrams from Thai requirements.

References

[1] National Electronics and Computer Technology Center. LexTo : Text lexeme tokenizer. [Online]. Available: http://www.sansarn.com/lexto

[2] NLTK Project. (2015) Natural language toolkit. [Online]. Available: http://www.nltk.org

[3] D. K. Deeptimahanti and R. Sanyal, "Static UML model generator from analysis of requirements (SUGAR)," in *Proceedings of the 2008 Advanced Software Engineering and Its Applications*, Dec. 2008, pp. 77–84.

[4] ——, "Semi-automatic generation of UML models from natural language requirements," in *Proceedings of the 4th India Software Engineering Conference*, 2011, pp. 165–174.

[5] H. Herchi and W. B. Abdessalem, "From user requirements to UML class diagram," in *Proceedings of the International Conference on Computer Related Knowledge*, Nov. 2012.

[6] 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, Apr. 2012.

[7] S. K. Shinde, V. Bhojane, and P. Mahajan, "NLP based object oriented analysis and design from requirement specification," *International Journal of Computer Applications*, vol. 47, no. 21, pp. 30–34, Jun. 2012.

[8] M. Elbendak, P. Vickers, and N. Rossiter, "Parsed use case descriptions as a basis for object-oriented class model generation," *J. Syst. Softw.*, vol. 84, no. 7, pp. 1209–1223, Jul. 2011.

[9] P. Charoenpornsawat. (1999) SWATH (Smart Word Analysis for THai). [Online]. Available: http://www.cs.cmu.edu/~paisarn/software.html

[10] V. Sornlertlamvanich, T. Charoenporn, and H. Isahara, "ORCHID: Thai part-of-speech tagged corpus," National Electronics and Computer Technology Center, Tech. Rep., 1997.

[11] P. Varasai, C. Pechsiri, T. Sukvaree, V. Satayamas, and A. Kawtrakul, "Building an annotated corpus for text summarization and question answering." in *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, 2008.