F. DEMİR

ATILIM UNIVERSITY 2021

GENERATING CLASS DIAGRAMS FROM SOFTWARE REQUIREMENTS IN TURKISH USING NATURAL LANGUAGE PROCESSING

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ATILIM UNIVERSITY

FATİH DEMİR

A MASTER OF SCIENCE THESIS
IN
THE DEPARTMENT OF SOFTWARE ENGINEERING

JANUARY 2021

GENERATING CLASS DIAGRAMS FROM SOFTWARE REQUIREMENTS IN TURKISH USING NATURAL LANGUAGE PROCESSING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ATILIM UNIVERSITY

BY

FATİH DEMİR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
THE DEPARTMENT OF SOFTWARE ENGINEERING

JANUARY 2021

Approval of the Graduate School of Natural and Applied Sciences, Atilim University.

_____

Prof. Dr. Ali KARA
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of **Master of Science in Software Engineering, Atilim University**.

_____

Prof. Dr. Ali YAZICI
Head of Department

This is to certify that we have read the thesis "Generating Class Diagrams from Software Requirements in Turkish Using Natural Language Processing" submitted by Fatih Demir and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____	_____
Prof. Dr. Ali YAZICI Co-Supervisor	Asst. Prof. Dr. Çiğdem TURHAN Supervisor

Prof. Dr. Ali Yazıcı
Software Eng. Department, Atilim University _____

Asst. Prof. Dr. Çiğdem TURHAN
Software Eng. Department, Atilim University _____

Assoc. Prof. Dr. Ahmet Murat Özbayoğlu
Artificial Intelligence Eng. Department, TOBB ETÜ _____

Asst. Prof. Dr. Güzin TİRKEŞ
Computer Eng. Department, Atilim University _____

Prof. Dr. Murat KOYUNCU
Information Sys. Eng. Department, Atilim University _____

Date: *08.02.2021*

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Fatih, Demir

Signature :

ABSTRACT

GENERATING CLASS DIAGRAMS FROM SOFTWARE REQUIREMENTS IN TURKISH USING NATURAL LANGUAGE PROCESSING

Demir, Fatih
M.S., Department of Software Engineering
Supervisor : Asst. Prof. Dr. Çiğdem TURHAN
Co-Supervisor : Prof. Dr. Ali YAZICI

January 2021, 36 pages

In software engineering, designing software that meets its requirements is a challenging task. In order to assist software engineers in this difficult task, many systems have been proposed in the literature that automatically generate class diagrams from software requirements. However, most of those studies are for software requirements in English, and such studies are very few for software requirements in Turkish. In this study, a system that automatically generates UML class diagram from software requirements in Turkish is proposed. The system first analyzes software requirements using natural language processing tools, particularly the dependency parser tool. Then, this parsing result is processed by a rule-based system and the classes, attributes and methods of the classes as well as the relations between classes are extracted. The extracted items are transformed into a class diagram with the visualization tool. Promising results were obtained when the performance of the system was evaluated. Therefore, it is concluded that class diagrams can be generated using natural language processing tools, especially the dependency parser, from software requirements in Turkish.

Keywords: software requirements, class diagram, natural language processing

ÖZ

DOĞAL DİL İŞLEME KULLANARAK TÜRKÇE YAZILIM GEREKSİNİMLERİNDEN SINIF DİYAGRAMLARI OLUŞTURMA

Demir, Fatih
Yüksek Lisans, Yazılım Mühendisliği Bölümü
Tez Yöneticisi : Dr. Öğr. Üyesi Çiğdem TURHAN
Ortak Tez Yöneticisi : Prof. Dr. Ali YAZICI

Ocak 2021, 36 sayfa

Yazılım mühendisliğinde, gereksinimlerini karşılayacak bir yazılım tasarlamak zorlu bir görevdir. Bu zor görevde, yazılım mühendislerine yardımcı olması için, literatürde yazılım gereksinimlerinden otomatik olarak sınıf diyagramı üreten çokça sistem önerilmiştir. Fakat bu çalışmaların çoğu İngilizce yazılım gereksinimleri içindir ve Türkçe yazılım gereksinimleri için bu tür çalışmalar çok azdır. Bu çalışmada, Türkçe yazılım gereksinimlerinden otomatik olarak UML sınıf diyagramı üreten bir sistem önerilmiştir. Bu sistem, ilk olarak, doğal dil işleme araçlarını, özellikle de bağımlılık çözümleyici aracını kullanarak, yazılım gereksinimlerini çözümler. Sonra, bu çözümleme sonucu kural bazlı bir sistem tarafından işlenir ve sınıflar, sınıfların nitelik ve metotları ve sınıflar arası ilişkiler ayıklanır. Ayıklanan öğeler görselleştirme aracı ile sınıf diyagramına dönüştürülür. Sistemin başarım değerlendirmesi yapıldığında umut vadeden sonuçlar alınmıştır. Bundan dolayı da Türkçe yazılım gereksinimlerinden doğal dil işleme araçları, özellikle de bağımlılık çözümleyici, kullanılarak sınıf diyagramlarının üretilebileceği sonucuna varılmıştır.

Anahtar Kelimeler: yazılım gereksinimleri, sınıf diyagramı, doğal dil işleme

To software engineering

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor Asst. Prof. Dr. Çiğdem Turhan for supporting me throughout my research, for correcting my mistakes without getting tired and motivating me in my bad times. Also, I would like to thank my co-supervisor Prof. Dr. Ali Yazıcı who cares for me and supports me in every aspect of life.

I shall also thank to jury members Assoc. Prof. Ahmet Murat Özbayoğlu, Asst. Prof. Dr. Güzin Tirkeş and Prof. Dr. Murat Koyuncu for reviewing my thesis and suggesting valuable remarks to make it better.

Finally, I am grateful to my father Hasan Demir, my mother Aysun Demir, my elder sister Duygu Demir Kızılırmak and her family for making me who I am today.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

In a world where various types of software are used to accomplish different types of tasks, one can, and should, expect them to help us build software also. It can be claimed that they do help us to build software, especially in modeling, implementation, testing, etc. There are indeed a lot of Computer-Aided Software Engineering (CASE) tools that make the life of software engineer easier. But we may want to keep our expectations high considering the fact that people use various “intelligent” software solutions to make a wide variety of things autonomous. Even some “non-intelligent” software solutions result in automated if not autonomous implementations. So, why can’t we make, at least try to make, software production automated, by using yet another software? That is the main question standing behind the motivation of this study.

As it is said, the question above is the main one which is a very comprehensive job and requires much more study than a single thesis study. We can narrow it to one level considering the traditional software engineering process model and ask “which phase can we make automated?” For this study, the “design” phase is chosen. The question “Which part of the design phase can we make automated?” needs to be answered to determine the scope of this study.

In most of the engineering branches, designing a system is a challenging task. It requires creativity and experience to design a system that fully satisfies its requirements. Satisfying requirements is important because that is what determines the quality of the system. Designing a system from scratch is prone to not meeting its requirements. Furthermore, sometimes, engineers struggle so hard to start designing that all they need would be a draft of design from where they could continue. So, in summary, designing is a difficult task and providing assistance in the initial design is another motivating reason for this study.

To automate a part of the design phase, this study aims to automatically draw the class diagrams from the requirements given in the Turkish language. The scope of this research is limited to the following constraints:

- Class diagrams will be generated only;
- Requirements will be accepted in the Turkish language;
- Existent Natural Language Processing (NLP) tools and techniques will be used;
- Existent data sets will be used;
- Input requirements are assumed to be free of ambiguities and contradictions.

As will be explained in the next chapter, there are several publications for the generation of class diagrams from requirements. But, most of them are for the requirements written in the English language. Normally, for any other research, it is expected that the language should not make any difference. But when the topic is related to natural language, it does. Those studies could indeed contribute to future research, but they cannot be applied as-is to studies conducted for natural languages other than English.

As will also be explained in the next chapter, there are only a few studies on the generation of class diagrams from requirements written in the Turkish language which have used only a small amount of the capabilities of NLP tools. That is the reason why this study was deemed necessary.

The main NLP tool that has not been used in the previous studies is the dependency parser. Using the dependency parser, the role of every word in a sentence can be determined such as predicate, subject, object, etc. The dependencies between the words are also determined, thus an extra effort is not required to extract phrases or multi-word expressions. This study utilizes the dependency parser because of these advantages to generate a better analysis of the sentences. Moreover, in this study, not only the class diagram is generated, but also, it is visualized automatically.

1.1. Organization

The organization of the thesis is as follows.

Chapter 2 presents some background information and related work from the literature.

Chapter 3 presents the proposed system by explaining each of its modules.

Chapter 4 presents a discussion on the results of this study and suggest future work.

Chapter 5 presents the evaluation of the system and comments about the evaluation results.

Chapter 6 summarizes this study and presents limitations and ideas for future work.

CHAPTER 2

BACKGROUND INFORMATION

Some prior knowledge about software engineering, natural language processing and the Turkish language is required to fully understand this study which will be explained in this chapter.

2.1. Software Engineering

Software engineering, in its most general definition, is the application of engineering discipline to software production [1]. The traditional software engineering process model consists of the following phases:

1. Requirement Analysis
2. Design
3. Implementation
4. Testing
5. Maintenance

Although there are various types of software engineering process models, most of them have the requirements gathering and design phases. As this study only concerns the requirements and design phases, they will be explained in detail in the following subsections.

2.1.1. Requirement Analysis

A requirement is a natural language expression of what the software is expected to do. There are two types of requirements: functional and non-functional requirements. Functional requirements deal with the functionalities of the software whereas non-functional requirements deal with the quality attributes of the software. In this study, our interest will be on functional requirements.

The main source of requirements is the customer. But, every stakeholder of the software may also affect the requirements.

2.1.2. Design

After requirements are gathered, the software is designed to satisfy its requirements. The design of the software is visualized by a standard notation called Unified Modelling Language (UML). There are various types of diagrams in UML to visualize various aspects of the design of the software.

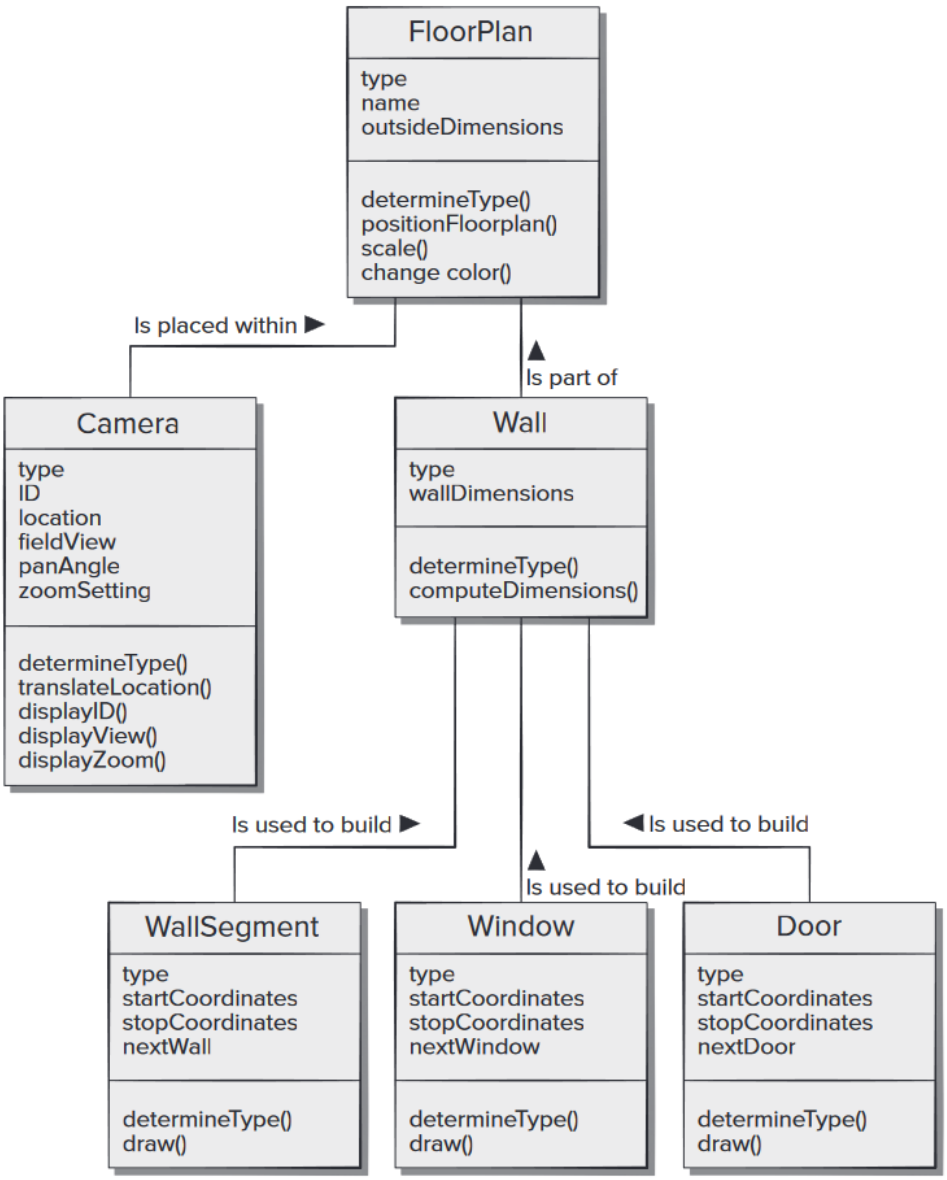

Figure 2.1 An example analysis level class diagram [1]

The class diagram is a type of UML diagram in which the classes, their attributes, their methods, and relationships between them are shown. Details in class diagrams can be at two levels. In the first level, class diagrams contain fewer details and are somewhat high-level. The classes in these types of class diagrams are called analysis classes. An example of this type of diagram can be seen in Figure 2.1.

In the second level, class diagrams contain more detail and are almost ready to be implemented. The classes in these types of class diagrams are called design classes. An example of this type of diagram can be seen in Figure 2.2.

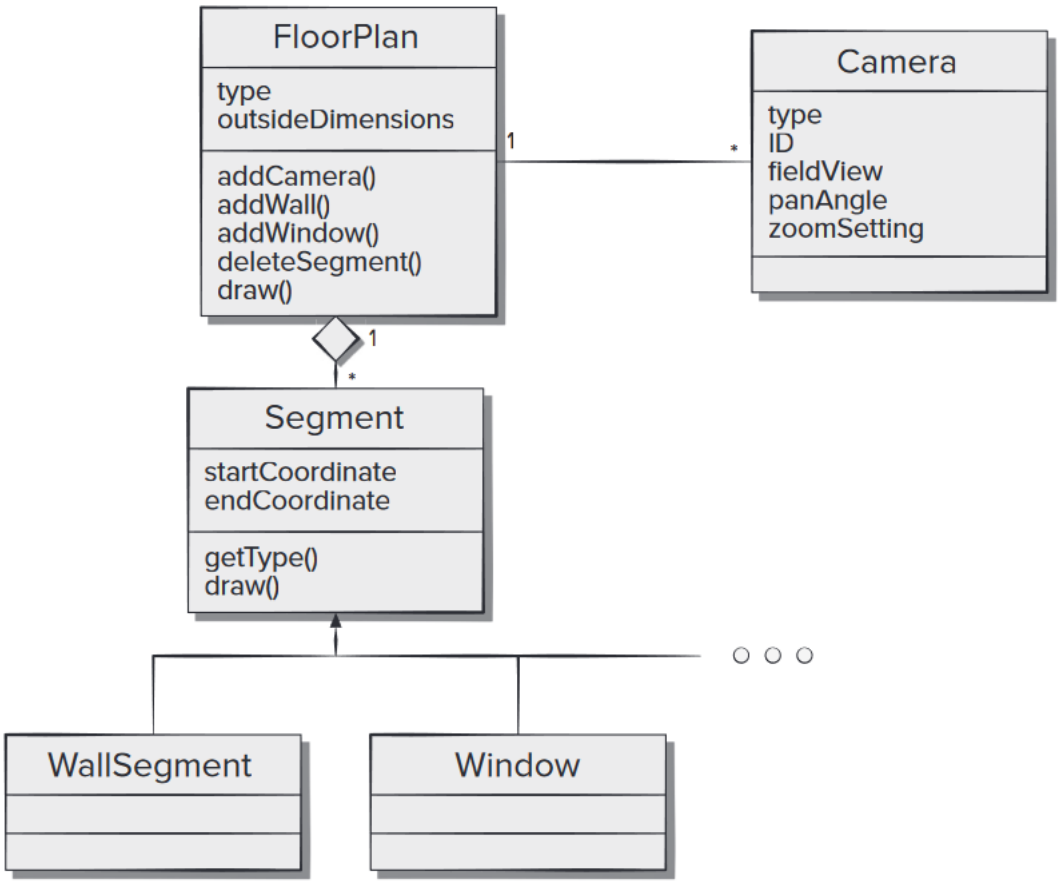

Figure 2.2 An example design level class diagram [1]

In this study, the generated class diagrams will consist of analysis classes because they will be generated directly from the requirements and will be at a high level.

2.2. Natural Language Processing (NLP)

NLP, as its name suggests, is a field of artificial intelligence that deals with the processing of natural languages to enable computers to understand and generate natural language speech or text. NLP allows humans to interact with computers using their natural languages. Also, it allows computers to analyze human language and act accordingly. Some capabilities of the modern NLP tools are:

- Tokenization;
- Stemming;
- Part-of-speech (POS) tagging;
- Normalizing;
- Dependency/Grammatical parsing.

Tokenization is the process of dividing the text into processable elements, i.e., tokens. Stemming is the process of finding the stems of words. POS (Part-of-Speech) tagging is the process of tagging the words' POSs which show the role of a word in a text such as a noun, verb, adjective, etc. Normalization means trying to correct errors in the text if there are any. Examples of such errors are misspelled words, non-alphanumeric characters, whitespace errors, etc. Dependency parsing is the process of finding out the relations (i.e., dependencies) between words in a text.

2.3. Turkish Language

The Turkish language is the official language of Turkey and Northern Cyprus. It is also spoken in the countries between Europe and Central Asia by a small part of the population. It is one of the 35 Turkic languages, which are regarded as one of the subdivisions of Altaic languages. It is written in the Latin alphabet. It is an agglutinative language which means new words can be derived by appending suffixes to the end of the stem of a word. A popular example of how long can a word become by the suffixes is the word "muvaffakiyetsizleştiremediklerimizdenmişsinizcesine" which means "as if you were one of those we could not fail". The suffixes in this word are shown below.

muvaffakiyet-siz-leş-tir-e-medik-ler-imiz-den-miş-siniz-ce-si-ne

Sentences usually start with a subject and end with a predicate. This order can be changed without changing the meaning of the sentence, but it is usually done in literary works, not in formal texts such as software requirements specification.

2.4. Related Work

There are some studies for the generation of class diagrams from requirements, but most of them are for the English language. On the contrary, for the Turkish language, only a few studies exist. There are also studies for other languages (Japanese [2], Thai [3] and French [4]) but they are not in our area of interest.

2.4.1. Studies for Requirements in English

The first study on extracting classes from requirements using NLP dates back to 1996. A project called NL-OOPS (Natural Language – Object-Oriented Production System) by Mich [5] can extract classes from given natural language requirements using LOLITA (Large-scale Object-based Linguistic Interactor Translator Analyser) as the NLP tool. In another study by Ambriola and Gervasi [6], a tool called Circe is developed that checks requirements and builds models from them. CM-Builder (Class Model Builder) tool developed by Harmain and Gaizauskas[7] can generate class diagrams either automatically or interactively.

In 2004, a study by Song et al. [8] lays out the rules to extract classes from requirements. From that date, more than 15 papers were published related to class diagram generation where most of them referenced that study.

The remaining studies related to English Language can be divided into two groups differentiated by the use of the domain ontology technique.

Domain ontology represents the concepts specific to a field containing terms related to the field and their definitions. In some studies on generation of class diagrams from requirements in English Language [9]–[16], the extraction of classes is supported by domain ontology in addition to NLP to increase the accuracy of the extraction. Since there is no such purpose of using domain ontology technique in this study, those studies will not be explained here.

The studies that do not use the domain ontology technique is in our area of interest. Each of them will be explained next.

In a study by Alkhader et al. [17], a class diagram can be extracted from requirements. The format of the class diagram is in XML but it can be represented visually or textually.

Static UML Model Generator from Analysis of Requirements (SUGAR) is the name of the tool proposed by Kumar et al. [18] that can generate both use case model and class diagram.

UML Model Generator from Analysis of Requirements (UMGAR) is another tool by Deeptimahanti et al. [19], [20] that can generate UML diagrams in XMI format.

In a project by Bajwa et al. [21] a system is designed that can generate a class diagram and its code in the popular object-oriented programming languages.

Natural language Processing for Class (NLPC) is the tool proposed by Kothari [22] that can only extract classes, attributes and methods by some simple rules.

In a comprehensive study by Sagar et al. [23], a system that can generate a conceptual model (in other words, analysis class diagram) is proposed. One distinct feature of that study from the previous ones is that the dependency parser is used to extract classes, in addition to POS tagger. The average recall and precision values for that study are 100% and 89% respectively.

A system proposed by Tripathy et al. [24] can generate class diagrams using POS tagger. The output format is XML and XSD.

In addition, four studies mainly use the POS tagger in the extraction of classes [25]–[28]. Finally, three studies utilize the dependency parser in addition to the POS tagger in class extraction [29]–[31]. The recall values of those studies are 89%, 94%, 100% and the precision values are 92%, 82%, 75%, respectively.

2.4.2. Studies for Requirements in Turkish

For extracting classes from requirements in Turkish, there is only one paper published by Bozyiğit et al. [32] and only one doctorate thesis also by Bozyiğit [33]. Those studies are very similar to each other and both of them are explained below.

In those studies, a rule-based model is proposed by the author. There are 5 categories of rules:

1. General rule
2. Class rule
3. Attribute rule
4. Method rule
5. Relationship rule

In the general rules:

- Nouns except proper nouns are selected as candidate classes.
- Noun phrases are detected.
- Verbs are collected as method names.
- Verb phrases are detected.

Class rules deal with selecting classes among class candidates. It selects classes based on:

- Frequency of nouns
- Frequency of the second noun in noun phrases
- Verbs in the sentence

Attribute rules deal with selecting the attributes of classes. Attributes are selected according to the following rules.

- Adjective types can be attributes such as color, number, shape, etc.
- The second noun of a noun phrase is selected as an attribute if the first noun takes possessive of place suffixes.
- Non-frequent nouns are selected as attributes.
- If the verb is one of the pre-determined words (such as “sahip olmak”, “içermek”, “bulundurmak”) all the nouns except the class are selected as an attribute.

Method rules deal with extracting methods of classes. Methods are selected according to the following rules.

- Each verb is a candidate method except the words listed by the authors.
- The verb in a sentence is selected as a method belonging to the class of the sentence.
- A method can belong to more than one class in a sentence.

Relationship rules extract the relationships between classes. Aggregation, composition and generalization types of relationships can be detected. There are rules defined by the authors to detect aggregation. To detect composition and generalization, there are linguistic patterns instead of rules defined by the authors.

There are three main NLP tools used in these studies, the main one being the POS tagger. The others are stemming and tokenization. Most of the rules depend on the POS tags of the words.

The main difference of our study from the studies explained above is the usage of a dependency parser, along with POS tagger. That is, although the POS tags are still important for the extraction of classes, the results of the dependency parser are more valuable.

CHAPTER 3

PROPOSED SYSTEM

To show that class diagrams can be generated automatically from the requirements provided in natural language, which is the main objective of this study, an application is designed and implemented.

The data flow diagram of the application is shown in Figure 3.1.

The application consists of 4 main modules as listed below:

1. NLP Pipeline API Client
2. Analyzer
3. Extractor
4. Visualizer

Briefly, the application takes a list of natural language requirement sentences, processes them, and outputs an image of the generated UML class diagram. It is expected that the generated UML class diagram should be similar to the ones drawn by a human designer.

The modules of the system are explained in the following sections.

3.1. NLP Pipeline API Client Module

The main job of this module is to process the given sentence using the ITU NLP Tool [34]. ITU NLP Tool can be used in two ways: by entering text to the web site or by sending HTTP requests to its API. The API method is chosen to make it easier to process sentences programmatically.

To be able to use ITU NLP API, an account must be created by its administrators. After an account is created, an API token is assigned to the account. By using this token, it becomes possible to send HTTP requests to the API.

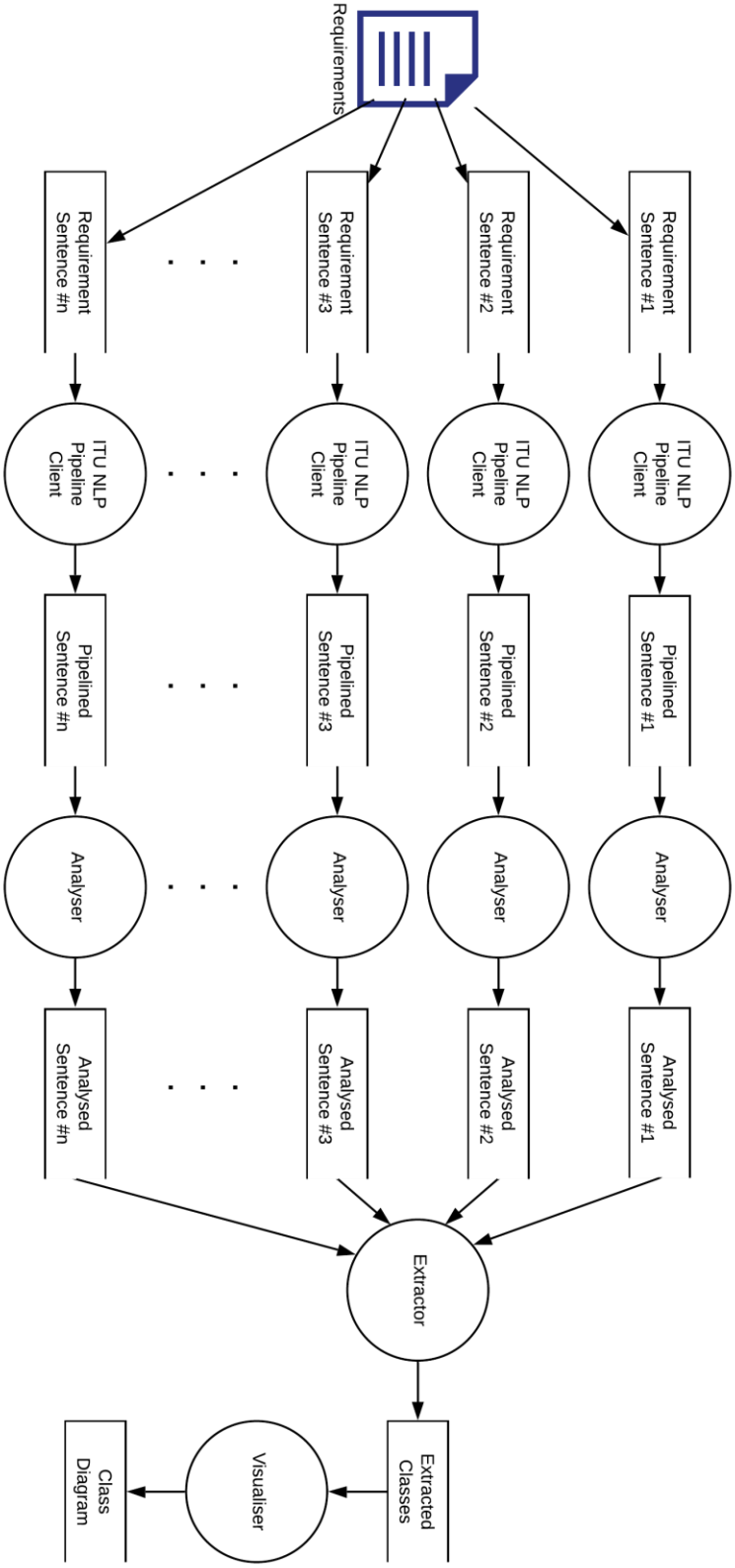

Figure 3.1 Data flow diagram of the application

An example URL with the query parameters is shown below. Note that the example URL is not encoded to make it more readable.

```
http://tools.nlp.itu.edu.tr/SimpleApi?tool=pipelineFormal&input=Müzik
grubuna yeni bir müzisyen katılabilir.&token=XXXXXXXXXXXXXXXXXXXX
```

If the request succeeds, the sentence is run through the ITU NLP Pipeline, and the following result is returned.

```
1 Müzik       müzik     Noun Noun A3sg|Pnon|Nom          2 POSSESSOR
2 grubuna     grup      Noun Noun A3sg|P3sg|Dat          6 MODIFIER
3 yeni        yeni      Adj  Adj  _                      5 MODIFIER
4 bir         bir       Adj  Num  _                      5 DETERMINER
5 müzisyen    müzisyen  Noun Noun A3sg|Pnon|Nom          6 SUBJECT
6 katılabilir kat       Verb Verb Pass|Able|Pos|Aor|A3sg 0 PREDICATE
7 .           .         Punc Punc _                      6 PUNCTUATION
```

The result shows each element in the sentence and their dependencies on the other elements. Each line of the result consists of tab-separated columns. These columns represent the tools used in the pipeline and contain the result of the tool. The structure of a line is shown below.

Id	Token	Normalized Form	POS	POS	Word Analysis	Related Element	Relation

To be able to use the NLP pipeline result in the application, each of its lines is converted to an object where the Class of this object is called Element. A list of element objects is the output of this module.

So, as a summary, this module takes a natural language sentence as an input and outputs a list of Element objects. The list of Element objects is processed by the Analyzer module, which is explained in the following section.

3.2. Analyzer Module

The main goal of this module is to pre-process the given pipelined sentence to make it ready for class extraction.

The value in the “Related Element” column of the pipeline result shows the identifier of the element that it is related to where the relation is shown in the “Relation” column.

As it can be understood from this syntax, the textual representation of the pipeline result can be converted to a tree by connecting each element to the element that it is related to. Showing it as a tree is helpful to see and understand the pipeline result. The tree representation of the pipeline result, mentioned in the previous section, would be the one shown in Figure 3.2.

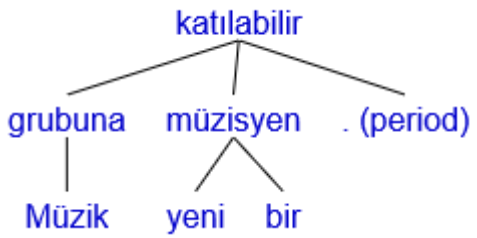

Figure 3.2 Example tree representation of pipeline result for the sentence “Müzik grubuna yeni bir müzisyen katılabilir.”

The tree data structure is used throughout the application to represent the NLP pipeline result. The very first job of this module is to build a tree from the incoming list of Element objects. The predicate element becomes the root of the tree because it is related to the element with the identifier 0, i.e., a non-existent element. After assigning it as root, elements that are related to the predicate are added as leaves. This process is repeated recursively for each element. After the list is exhausted, the tree becomes ready to be used.

After the tree is built, the following rules are executed on it to prepare the tree, and the sentence it represents, for class extraction. Note that the examples are specific to each step. Also, for every example, it is assumed that the previous rules are already executed.

1. Remove the punctuation that ends the sentence.

Rationale: Requirement sentences almost always ends with a period. Besides, the ending punctuation will not affect the extraction.

Example: For the sentence “Kullanıcı sisteme giriş yapabilir.” whose tree is shown in Figure 3.3, the period is removed. The resulting tree is shown in Figure 3.4.

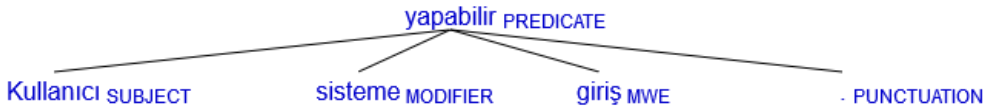

Figure 3.3 Before applying Rule 1

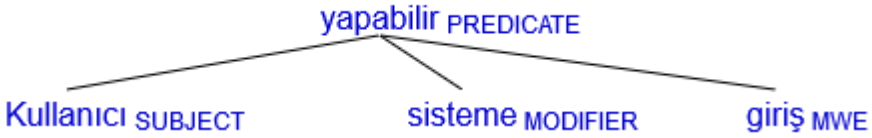

Figure 3.4 After applying Rule 1

2. Merge two or more elements that represent a derived word.

Rationale: NLP tool represents the stem and the derived word as two separate elements. Reducing them to a single element makes the extraction easy.

Example: For the sentence “Sistem kimlik doğrulama modülünü kullanır.” whose tree is shown in Figure 3.5, the word “doğrulama” is derived from “doğrula” and they are merged into a single element. The resulting tree is shown in Figure 3.6.

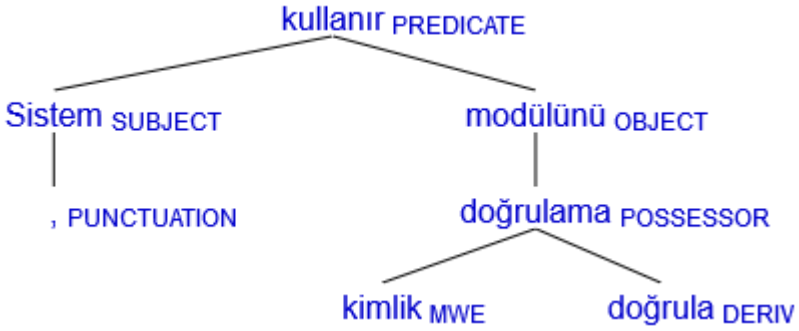

Figure 3.5 Before applying rule 2

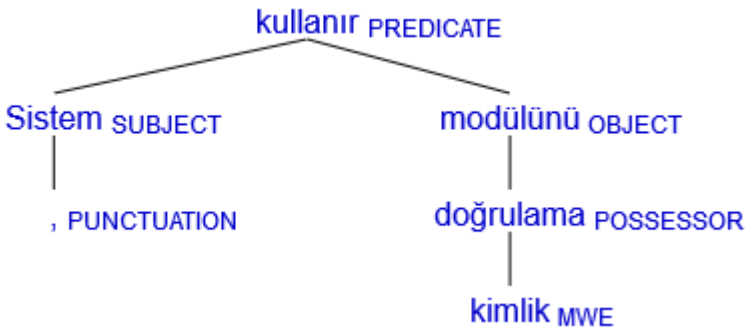

Figure 3.6 After applying rule 2

3. Remove the conjunction word *"ve"* and bring the conjunct word to the same level in the tree.

Rationale: The word "ve" is irrelevant to extraction but the elements that it conjuncts are.

Example: For the sentence "Kullanıcılar isim ve parola ile giriş yapar." whose tree is shown in Figure 3.7, the word "ve" is removed and the word "isim" is brought to the same level as the word "parola". The resulting tree is shown in Figure 3.8.

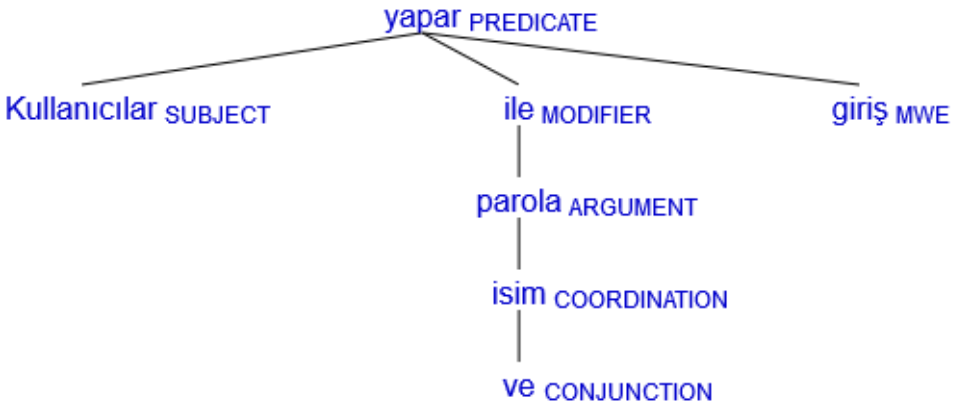

Figure 3.7 Before applying rule 3

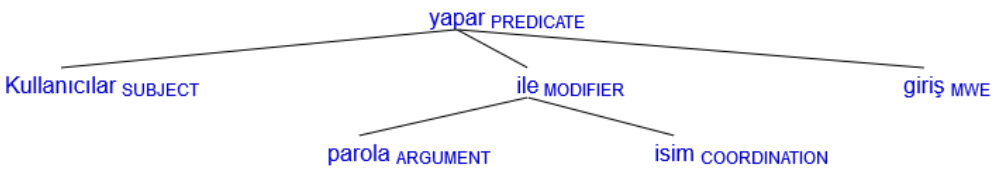

Figure 3.8 After applying rule 3

4. Remove the element if its type is DETERMINER and its normalized form is *"bir"* or *"her"*.

Rationale: These words are irrelevant to extraction.

Example: For the sentence "Her kullanıcı hesabının bir rolü vardır." whose tree is shown in Figure 3.9, the word "her" and "bir" is removed. The resulting tree is shown in Figure 3.10.

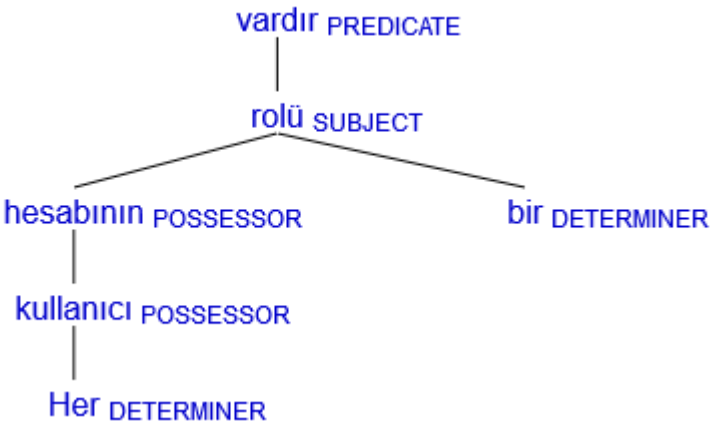

Figure 3.9 Before applying rule 4

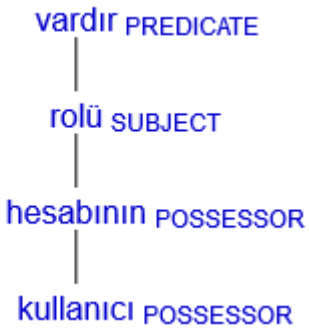

Figure 3.10 After applying rule 4

5. Remove the elements that have type MODIFIER and whose part of speech is adjective.

Rationale: Adjectives are currently irrelevant to extraction.

Example: For the sentence “Yönetici yeni kullanıcı ekleyebilir.” whose tree is shown in Figure 3.11, the word “yeni” is removed. The resulting tree is shown in Figure 3.12.

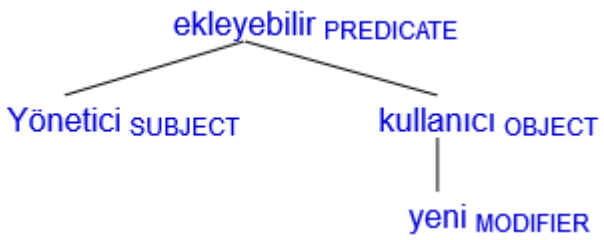

Figure 3.11 Before applying rule 5

Figure 3.12 After applying rule 5

6. If the element’s type is POSSESSOR, merge it to the element that it possesses. Rationale: Merging these words creates a phrase that can be treated as a single element.

Example: For the sentence “Kullanıcı, mesajlaşma grubundan ayrılabilir.” whose tree is shown in Figure 3.13, the word “mesajlaşma” is the possessor of the word “grubundan” and they are merged into a single element. The resulting tree is shown in Figure 3.14.

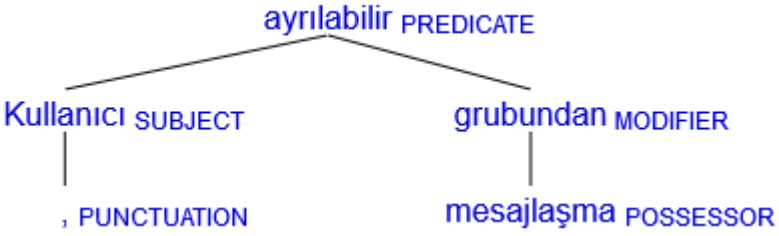

Figure 3.13 Before applying rule 6

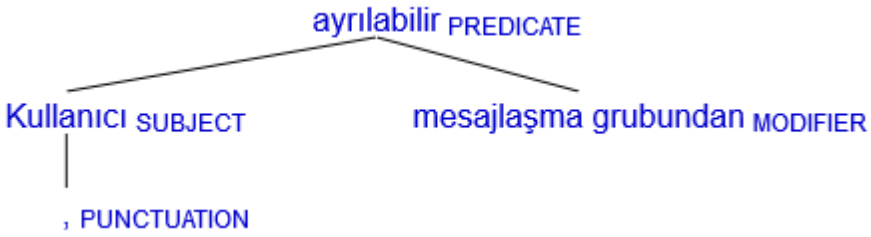

Figure 3.14 After applying rule 6

7. Remove commas and bring the related element to the upper level

Rationale: Comma is irrelevant for the extraction but the words that it connects are.

Example: For the sentence “Öneri listesi müzik, film ve kitaplar içerir.” whose tree is shown in Figure 3.15, the comma is removed and the word “müzik” is brought to the same level as the word “film”. The resulting tree is shown in Figure 3.16.

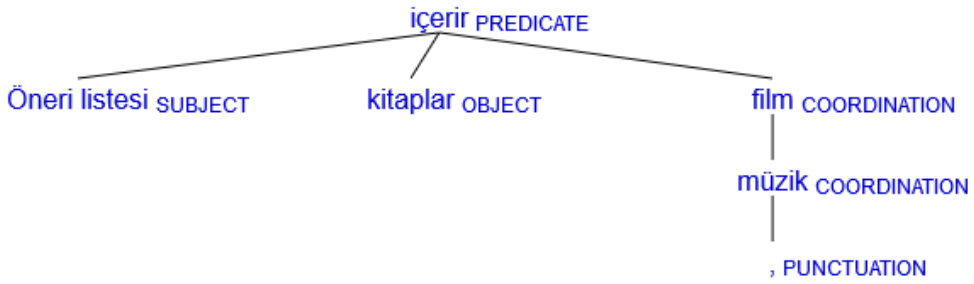

Figure 3.15 Before applying rule 7

Figure 3.16 After applying rule 7

8. Transform the tree to prepare it for the extraction module. As it can be recalled, each element of the tree has the following properties:
    - Id
    - Token
    - Normalized Form
    - POS

- Word Analysis
- Related Element
- Relation

Id and related element properties are used to build the tree and since the tree is already built, they are not needed by the extractor module so they are removed.

3.3. Extractor Module

This module's main goal is to extract classes, their attributes, methods and their relationships. While the analyzer module works on a single tree, the extractor module works on a list of trees.

Classes and their attributes are selected among the nouns. To determine whether a noun is a class or an attribute, every noun is at first assumed to be a class candidate. Class candidates are given points according to how they are selected. The rules of giving points are as follows:

- When the subject of the sentence is selected as candidate class, it is initialized with 2 points.
    - Example: For the sentence given in Figure 3.17, the subject "öğrenci" is selected as a class candidate and is given 2 points.

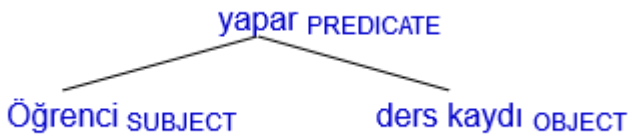

Figure 3.17 Analysis result of the sentence "Öğrenci ders kaydı yapar."

- As an exception to the previous rule, if the predicate of the sentence is in the passive form, the candidate class is initialized with 1 point. This rule is needed because the NLP tool is not yet fully capable of determining the subject in sentences that have a passive predicate.
    - Example: For the sentence given in Figure 3.18, the NLP tool wrongly determines "ders kaydı" as the subject where the real subject is

"öğrenci". To compensate for this error "ders kaydı" is still selected as a class candidate but initialized with 1 point.

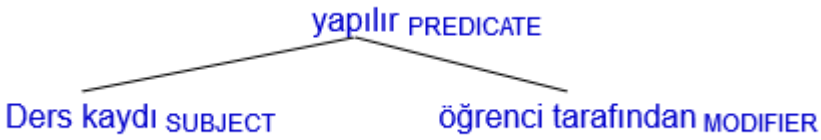

Figure 3.18 Analysis result of the sentence "Ders kaydı öğrenci tarafından yapılır."

- If a noun in a sentence (other than subject and predicate) is already selected as a candidate class, its point is incremented by 1. Otherwise, it is initialized with 1 point.
    - Example: The noun "ders" in the sentence shown in Figure 3.19 is selected as a class candidate with a point 1. Then, it is also seen in the sentence shown in Figure 3.20 and its point is incremented by 1. Also, in Figure 3.20, the noun "sınıfları" ("sınıf" in the normalized form) is selected as a class candidate with point 1.

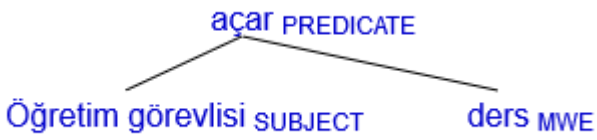

Figure 3.19 Analysis result of the sentence "Öğretim görevlisi ders açar."

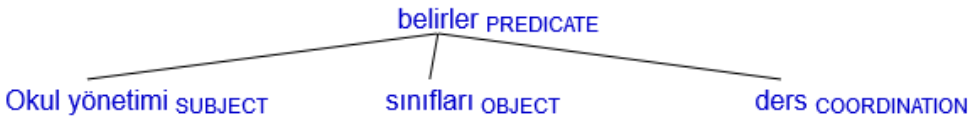

Figure 3.20 Analysis result of the sentence "Okul yönetimi ders ve sınıfları belirler."

These rules are deemed necessary to identify which nouns are more likely to become classes. Usually, in the sentences of requirements, the subjects are the nouns that are very important to the system, which means they are more likely to be selected as classes. That is the reason for initializing subjects with 2 points. The reason for giving 1 point to other nouns is that they are still important but they are more likely to become attributes, not classes. The nouns that have more than 1 point are treated as classes

because they are important enough to be mentioned in more than one sentence of the requirements. There is a shortcoming of this rule: There still can be attributes with the same name which are mentioned in more than one sentence in the requirements. In this case, they would have more than 1 point which causes them to be selected as classes by mistake when they should have been selected as attributes. The solution to this problem is expected to be done in future studies.

By using the point rules explained above, the extractor module executes the following rules to extract classes.

1. Find the predicate and the subject in every sentence and select the subjects as classes and predicates as their methods.
2. For the sentences whose predicate is a noun and is selected as a class in rule 1, select the other nouns in the sentence as classes and set them to be the subclass of the predicate class. This rule detects the inheritance relationships between classes.
3. Select all the nouns in a sentence as candidate classes that are related to the subject class. Set those classes' points to 1. If those nouns are already selected as a class, increment their points. This rule detects the composition relationships between classes.
4. Convert candidate classes that have only 1 point to attributes.

The output of this module is the list of extracted classes with their attributes, methods and their relations to other classes.

3.4. Visualizer Module

This module's main goal is to show the extracted classes as a class diagram. The application named PlantUML[1] is chosen to draw class diagrams. The main advantage of PlantUML is that it can convert the textual representation of class diagrams to an image file. It has its own syntax for representing class diagrams textually. By submitting the text to it via the web site, the class diagram image can be obtained.

[1] http://www.plantuml.com

To use PlantUML to draw class diagrams, this module, firstly, generates a textual representation of the given class list according to PlantUML syntax. Then, it sends the generated text to the PlantUML server by an HTTP request to obtain a class diagram image.

3.4.1. PlantUML Syntax for Class Diagrams[2]

For a text to be interpreted by PlantUML, it must start with the @startuml tag and end with the @enduml tag. All the class, attribute, method and relation declarations are stated between these tags.

The syntax of declaring classes is very flexible which means there are many ways to declare classes. Only one of them is chosen and is described.

A class can be declared as follows:

```
class <class_name>
```

An attribute of a class can be declared as follows:

```
<class_name> : <attribute_name>
```

A method of a class can be declared as follows:

```
<class_name : <method_name>()
```

An inheritance relationship between classes is declared as follows:

```
<super_class> <|-- <subclass>
```

A composition relationship between classes is declared as follows:

```
<class> *-- <part_class>
```

For example, the following text produces the class diagram shown in Figure 3.21.

[2] https://plantuml.com/class-diagram

```
@startuml
class Player
class Media
class Audio
class Video
Media : duration
Media : seek()
Video : width
Video : height
Player : play()
Player : stop()
Media <|-- Audio
Media <|-- Video
Player *-- Media
@enduml
```

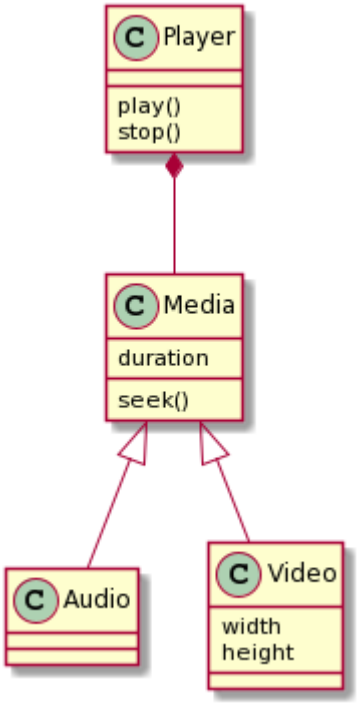

Figure 3.21 Example class diagram visualized by PlantUML

CHAPTER 4

EVALUATION

The evaluation of the proposed system is done based on the evaluation method proposed by Bozyiğit et al. [32]. The data set for the evaluation is also retrieved from that study. 12 requirement documents selected for the data set with 10-15 sentences in each of them.

Table 4.1 shows the performance of the system of finding classes for each requirements document.

Table 4.1 Class extraction performance of the system

Requirements	Correctly Found Classes	Incorrectly Found Classes	Missing Classes	Precision	Recall
Restaurant	11	1	0	0.92	1.00
Company	7	2	2	0.78	0.78
Library	6	1	3	0.86	0.67
Game	7	2	0	0.78	1.00
Music band	5	1	2	0.83	0.71
File manager	5	3	2	0.62	0.71
Car gallery	4	2	1	0.67	0.80
Enrollment	6	4	0	0.60	1.00
ATM	6	2	2	0.75	0.75
Video rental	5	3	0	0.62	1.00
Cinema	4	3	0	0.57	1.00
Pressure	5	2	1	0.71	0.83

Precision is calculated according to the formula (1) and recall is calculated according to the formula (2).

$$precision = \frac{correct\ class\ count}{correct\ class\ count + incorrect\ class\ count} \quad (1)$$

$$recall = \frac{correct\ class\ count}{correct\ class\ count + missing\ class\ count} \quad (2)$$

As can be understood from the formulas, the precision value is inversely proportional to the incorrect class count. That is, it decreases as the system wrongly selects words as classes. The recall value is inversely proportional to the missing class count. That is, it decreases as the system fails to select words as classes.

As a result, the average precision value of finding classes is 0.73 and the average recall value is 0.85. The average precision and recall values for the referenced study was 0.94 and 0.94, respectively. The difference of precision is 0.21 and the difference of recall is 0.09. These results are summarized in Table 4.2.

Table 4.2 Class extraction performance summary of the system

	Average Precision	Average Recall
This study	0.73	0.85
Reference study [32]	0.94	0.94
Difference	-0.21	-0.09

The reason for the low precision value of the system is the greediness of the class extraction rules. In other words, the class extraction rules focus more on finding classes than finding correct classes. Another reason for the low precision value is that some attributes are wrongly selected as classes.

The recall value is somewhat satisfactory but still lower than the referenced study. Thus, some improvement in rules is still needed.

It can be argued that the recall value is more important than the precision value. In other words, it is more preferable to have incorrect classes rather than missing classes. The reason for this is that the incorrect classes can be eliminated by the designer easily, however, finding the missing classes is a more difficult task.

The graph in Figure 4.1 shows the relation between the performance of the system and the class counts for each requirements document. As can be seen from the graph, it can be said that the precision value increases as the class count increases. But there is no apparent relation between the recall value and the class count.

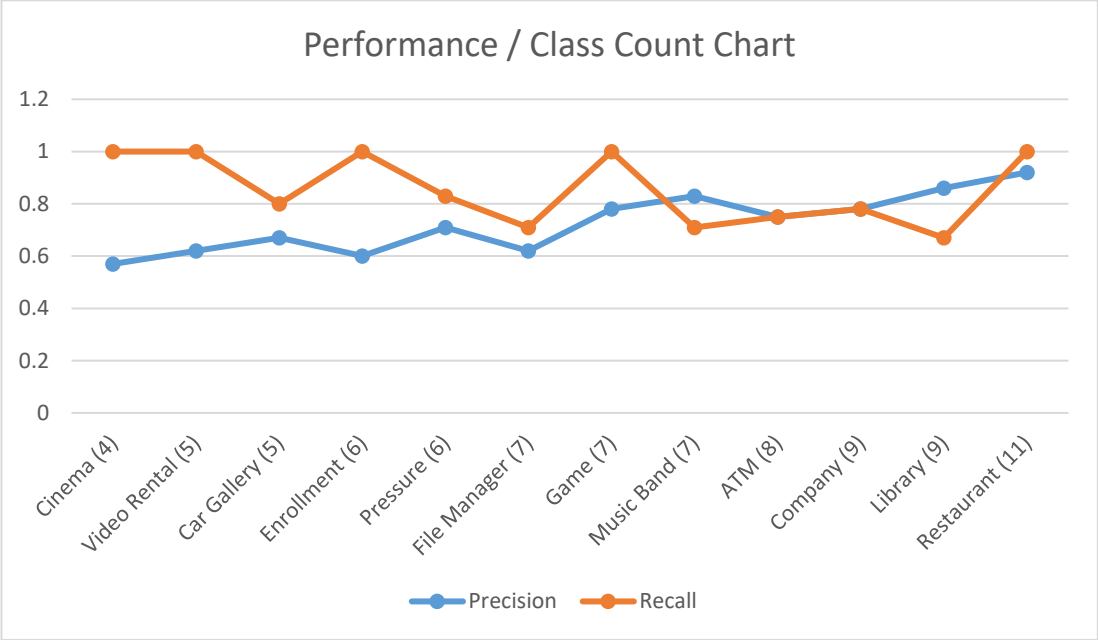

Figure 4.1 Relation between performance of the system and the class count contained in requirements

Table 4.3 shows the performance of the system of finding attributes of classes for each requirements document. Table 4.4 summarizes the results.

The system's performance of extracting attributes is low which indicates some changes or additions must be made to extraction rules. There could be more rules defined specifically to attribute or method extraction.

Table 4.3 Attribute extraction performance of the system

Requirements	Correctly found attributes	Incorrectly found attributes	Missing attributes	Precision	Recall
Restaurant	5	5	5	0.5	0.50
Company	7	6	1	0.54	1.00
Library	2	18	1	0.10	0.67
Game	7	12	0	0.27	1.00
Music band	6	2	0	0.75	1.00
File manager	1	2	4	0.33	0.20
Car gallery	12	5	0	0.70	1.00
Enrolment	3	13	7	0.19	0.30
ATM	7	7	0	0.5	1.00
Video rental	6	7	3	0.46	0.67
Cinema	4	10	0	0.29	1.00
Pressure	2	7	4	0.22	0.33

Table 4.4 Summary of attribute extraction process of the system

	Average Precision	Average Recall
This study	0.40	0.72
Reference study	0.86	0.90
Difference	-0.46	-0.18

CHAPTER 5

CONCLUSION

The main objective of this study is to generate class diagram from requirements in the Turkish language using natural language processing techniques, especially the dependency parser. To achieve this, firstly, requirement sentences are run through the NLP pipeline to be analyzed by dependency parser. Then, the parsed sentences are processed by a rule-based system to extract classes and their attributes, methods and relationships. Then, the extracted classes are visualized as a class diagram.

This study may contribute to the literature by being one of the very few studies about generating class diagrams from requirements in the Turkish language. It is also the first to utilize a dependency parser in doing so.

Based on the evaluation results of the proposed system, it is proven that the class diagrams can be generated from requirements in the Turkish language using natural language processing techniques. It is also proven that the dependency parser can make the extraction easier by providing smarter language analysis and is a promising NLP tool for future studies. But, as it can be understood from the evaluation results, there is still a need to develop detailed rules to extract classes correctly and fully.

5.1. Limitations

Since this study depends on other studies, tools and data, there are some limitations imposed by them. Also, this study has to be performed in a time frame, which is another limitation factor.

As developing an NLP tool is not in the scope of this study, the main limitation is the NLP tools available to use. Also, not every NLP tool is suitable because it has to support the Turkish language and it has to have the capability of dependency parsing.

The NLP tool's performance is another limitation. That is, if the tool's result is not accurate, then the proposed tool will produce results that are also not accurate.

Currently, the NLP tool is not very good at analyzing long sentences and sentences with a passive predicate.

Another limitation is the lack of available software engineering requirement data to be used in studies such as this. Such data is required to be able to develop rules to cover all forms of requirement sentences. Also, it is needed for studies to test and evaluate their proposed systems.

A rule-based system's capability is directly related to the rules it employs. The count of the rules grows with respect to the complexity of the problem. So, developing rules to cover every case of a natural language sentence is very difficult, if not impossible.

5.2. Future Work

To improve the proposed system, new rules that address more specific sentence structures can be developed. Also, there could be new rules about some specific predicate verbs such as "olmak (to be), sahip olmak (to have), yapmak (to do)" etc.

For even better results, the extraction process could be done in iterations, and in every iteration, the user's feedback may be inputted about whether the extracted classes are correct or wrong. Based on the user's input, the system would correct itself and as a result, the accuracy of the class diagram would be increased.

The proposed system processes the requirement sentences in order, but the order of the sentences does not affect extraction. However, in real-life requirement specifications, the order of the sentences can be important. Moreover, the sentences are usually ordered hierarchically from general to specific. Thus, the precision and the accuracy of the extraction process can be improved by taking advantage of the ordered structure of the requirement sentences.

The class generation system can also be used to improve requirements. If the system cannot select classes or their attributes and methods correctly, that may mean the requirement sentences are ambiguous. It could warn the user to correct the requirement sentences by clarifying the ambiguity. In this way, the requirement analyzer can be trained also.

The extraction process can be performed by a deep learning model that is trained by requirements and the matching classes. But, the biggest problem with this is the lack of requirements data.

REFERENCES

[1] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*. New York, NY: McGraw-Hill Education, 2020.

[2] M. Takahashi, S. Takahashi, and Y. Fujita, "A development method of UML documents from requirement specifications using NLP," *International Journal of Computer Applications in Technology,* vol. 33, no. 2–3, pp. 164–175, 2008.

[3] M. Jaiwai and U. Sammapun, "Extracting UML class diagrams from software requirements in Thai using NLP," in *Proceedings of the 2017 14th International Joint Conference on Computer Science and Software Engineering, JCSSE 2017*, 2017, pp. 1-5.

[4] S. Amdouni, W. Ben, A. Karaa, and S. Bouabid, "Semantic annotation of requirements for automatic UML class diagram generation," *IJCSI International Journal of Computer Science Issues*, vol. 8, no. 3, pp. 259–264, 2011.

[5] L. Mich, "NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA," *Natural Language Engineering,* vol. 2, no. 2, pp. 161–187, 1996.

[6] V. Ambriola and V. Gervasi, "Processing natural language requirements," in *Proceedings of the IEEE International Automated Software Engineering Conference, ASE,* pp. 36–45, 1997.

[7] H. M. Harmain and R. Gaizauskas, "CM-Builder: A natural language-based CASE tool for object-oriented analysis," *Automated Software Engineering,* vol. 10, no. 2, pp. 157–181, 2003.

[8] I.-Y. Song, K. Yano, J. Trujillo, and S. Lujan-Mora, "A Taxonomic Class Modeling Methodology for Object-Oriented Analysis," *Information Modeling Methods and Methodologies*, pp. 216–240, 2004.

[9] M. Landhäußer, S. J. Körner, and W. F. Tichy, “From requirements to UML models and back: How automatic processing of text can support requirements engineering,” *Software Quality Journal*, vol. 22, no. 1, pp. 121–149, 2014.

[10] P. More and R. Phalnikar, “Generating UML Diagrams from Natural Language Specifications,” *International Journal of Applied Information Systems*, vol. 1, no. 8, pp. 19–23, 2012.

[11] M. S. Jyothilakshmi and P. Samuel, “Domain ontology based class diagram generation from functional requirements,” in *International Conference on Intelligent Systems Design and Applications, ISDA*, 2012, pp. 380–385.

[12] 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, 2012.

[13] S. D.Joshi and D. Deshpande, “Textual Requirement Analysis for UML Diagram Extraction by using NLP,” *International Journal of Computer Applications*, vol. 50, no. 8, pp. 42–46, Jul. 2012.

[14] H. Herchi and W. ben Abdessalem, “From user requirements to UML class diagram*,” arXiv preprint arXiv:1211.0713*, 2012.

[15] M. Ibrahim and R. Ahmad, “Class diagram extraction from textual requirements using natural language processing (NLP) techniques,” in *2nd International Conference on Computer Research and Development, ICCRD 2010*, 2010, pp. 200–204.

[16] N. ; Zhou, X; Zhou, “Auto-generation of Class Diagram from Free-text Functional Specifications and Domain Ontology,” *Artificial Intelligence*, 2004.

[17] Y. Alkhader, A. Hudaib, and B. Hammo, “Experimenting With Extracting Software Requirements Using NLP Approach,” in *International Conference on Information and Automation, 2006 (ICIA ’06)*, 2006, pp. 349–354.

[18] D. D. Kumar and R. Sanyal, “Static UML Model Generator from Analysis of Requirements (SUGAR),” in *Proceedings of the 2008 Advanced Software Engineering and its Applications, ASEA 2008*, 2008, pp. 77–84.

[19] D. K. Deeptimahanti and M. A. Babar, “An automated tool for generating UML models from natural language requirements,” in *ASE2009 - 24th IEEE/ACM International Conference on Automated Software Engineering*, 2009, pp. 680–682.

[20] D. K. Deeptimahanti and R. Sanyal, “Semi-automatic Generation of UML Models from Natural Language Requirements,” in *Proceedings of the 4th India Software Engineering Conference*, 2011, pp. 165-174.

[21] I. S. Bajwa, A. Samad, and S. Mumtaz, “Object oriented software modeling using NLP based knowledge extraction,” *European Journal of Scientific Research*, vol. 35, no. 1, pp. 22–33, 2009.

[22] P. R.Kothari, “Processing Natural Language Requirement to Extract Basic Elements of a Class,” *International Journal of Applied Information Systems*, vol. 3, no. 7, pp. 39–40, 2012.

[23] V. B. R. Vidya Sagar and S. Abirami, “Conceptual modeling of natural language functional requirements,” *Journal of Systems and Software*, vol. 88, no. 1, pp. 25–41, 2014.

[24] A. Tripathy, A. Agrawal, and S. K. Rath, “Requirement Analysis using Natural Language Processing,” in *Proceeding of the Fifth International Conference on Advances in Computer Engineering (ACE 2014), At Kochi, India*, 2014, pp. 39–45.

[25] F. Bozyiğit, Ö. Aktaş, and D. Kılınç, “AutoClass: Automatic Text to OOP Concept Identification Model,” *International Journal of Computer Applications*, vol. 150, no. 10, pp. 29–34, 2016.

[26] C. R. Narawita and K. Vidanage, “UML generator - An automated system for model driven development,” in *16th International Conference on Advances in ICT for Emerging Regions, ICTer 2016 - Conference Proceedings*, 2017, pp. 250–256.

[27] M. A. Ahmed, W. H. Butt, I. Ahsan, M. W. Anwar, M. Latif, and F. Azam, "A Novel Natural Language Processing (NLP) Approach to Automatically Generate Conceptual Class Model from Initial Software Requirements," *Lecture Notes in Electrical Engineering*, vol. 2, pp. 467–475, 2017.

[28] J. Kuchta and P. Padhiyar, "Extracting concepts from the software requirements specification using natural language processing," in *Proceedings - 2018 11th International Conference on Human System Interaction, HSI 2018*, 2018, pp. 443–448.

[29] Ben Abdessalem Karaa, W., Ben Azzouz, Z., Singh, A., Dey, N., S. Ashour, A., & Ben Ghazala, H., "Automatic builder of class diagram (ABCD): an application of UML generation from functional requirements," *Software: Practice and Experience*, vol. 46, no. 11, pp. 1443–1458, December 2015.

[30] R. Sanyal and B. Ghoshal, "Automatic Extraction of Structural Model from Semi Structured Software Requirement Specification," in *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, 2018, pp. 543–558.

[31] R. Sharma, P. K. Srivastava, and K. K. Biswas, "From natural language requirements to UML class diagrams," in *2nd International Workshop on Artificial Intelligence for Requirements Engineering, AIRE 2015 - Proceedings*, 2015, pp. 25–32.

[32] F. Bozyigit, Ö. Aktaş, and D. Kilinç, "Automatic concept identification of software requirements in Turkish," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 27, no. 1, pp. 453–470, 2019.

[33] F. Bozyiğit, "Object Oriented Analysis and Source Code Validation Using Natural Language Processing," Ph.D. thesis, Dokuz Eylül University, İzmir, 2019.

[34] G. Eryiğit, "ITU Turkish NLP Web Service," in *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, 2015, pp. 1–4.