Generation of Class Diagrams from NLP Specifications

Chinmay Terse
Vidyalankar Institute of Technology
Mumbai, India
Email Id : cterse@gmail.com

Rahul Nair
Vidyalankar Institute of Technology
Mumbai, India
Email Id : rnair916@gmail.com

Vivek Ashokan
Vidyalankar Institute of Technology
Mumbai, India
Email Id : vivekashokan1995@gmail.com

Rugved Vivek Deolekar
Assistant Professor, Vidyalankar Institute of Technology, Mumbai, India
Email Id: rugved.deolekar@vit.edu.in

Abstract – The system proposes a method to create a process to generate UML diagrams from the user requirements provided by the user in natural language. Translating a paragraph of natural language into diagrams presents a challenging task. Whenever the user or the customer provides requirements for a system, the requirements engineers manually examine the provided requirements to understand the scope of the system and create design models from the analysis. However, such manual analysis presents various opportunities for errors and misinterpretations. Also, assigning manpower to analyse the requirements and create design models takes up a lot of time. This calls for an automated system, which can assist the Requirements Engineers to create UML diagrams from the given requirements, using Natural Language Processing. This paper proposes an approach to facilitate class diagram extraction from textual requirements using NLP techniques.

Keywords *– Software Engineering, Natural Language Processing, Requirement Engineering, Unified Modelling Language*

I. INTRODUCTION

In the industry, requirements provided by the customers for a software product are often in a pool of natural language specifications such as documents, interview excerpts, brainstorming sessions, etc. Sometimes these natural language specifications can be subject to ambiguity and their interpretation may depend upon the current geographical, psychological and sociological factors. One way to overcome this problem is to specify requirements to the Requirements Engineers in a formal or a semi-structured manner. However, this results in spending manpower, cost and time on converting the natural language requirements to formal language specifications. Some errors are also likely to be introduced in the process. It is the job of the requirements analysts to detect and fix any potential ambiguities, inconsistencies, and incompleteness in the requirements specifications documentations.

After the successful analysis of the user requirements, they are modelled into some design structures to get an overall idea of the system easily and help facilitate implementation. UML class diagrams are the main core of Object Oriented analysis and design systems where most other models are derived from.

The main aim of this paper is to use Natural Language Processing techniques to facilitate automation of the process of conversion of informal user requirements to UML diagrams.

Related Work

There are many tools devised for the analysis of text given in natural language. However, there are very few tools that extend this analysis to extraction of design information from natural language specifications. In this section, we look at some of the works that aim at deriving some kind of UML diagrams using the techniques of natural language processing and domain ontology.

Deva Kumar Deeptimahanti and Muhammad Ali Babar[2] in their paper "An Automated Tool for Generating UML Models from Natural Language Requirements" IEEE journal 2009, propose a system called UML Model Generator from

Analysis of Requirements, UMGAR for short. UMGAR is a domain-independent tool which models Use-case Diagram, Analysis class model, Collaboration diagram and Design Class Model from natural language specifications. With respect to the existing tools in this area, UMGAR provides more comprehensive support for generating models with proper relationships, which can be used for large requirement documents.

LOLITA was a natural language system proposed by Mich[7] which generates an object model automatically from natural language. This approach considers nouns as objects and use links to find relationships amongst objects. LOLITA system is built on a large scale Semantic Network (SN) that does not distinguish between classes, attributes, and objects.
Ambriola and Gervasi[4] present a Web-based environment called Circe. Circe is a complete environment which is equipped with several tools. It can build semiformal models, extract information from the NL requirements, and measure the consistency of these models.

Zhou and Zhou [8] propose a methodology that uses NLP and domain ontology. It is based on that the core classes are always semantically connected to each other by one to one, one to many, or many to many relationships in the domain. This methodology finds candidate classes using NLP through a part of speech (POS) tagger, a link grammar parser, linguistic patterns and parallel structure, and then the domain ontology is used to refine the result.

II. Proposed Design

In the previous section, we have reviewed the most recent works. The aim of our approach is to efficiently apply NLP techniques to achieve a fast and accurate analysis result.
Our proposed system will consist of four different blocks that will deal with analysis of natural language using NLP and implementation.

A. Natural Language Analysis block:

In this block, we take the natural language as input from the user and do the following processes on it.

- Sentence Splitting:

We take the input from the user in the form of a text file which contains a paragraph(s) of the user requirements. The initial step is to split this entire block of text into proper sentences, which are then passed to the Stanford parser in the following steps. The code for splitting the paragraph into sentences is written such that it considers the placement of sentence terminators to avoid confusion in the case any symbol which is used as a terminator is used in a different context. The symbol set {?, ., !} are considered as sentence terminators in our system.

- Syntactic Reconstruction:

After the sentence splitting phase, the system converts the sentences into the Subject-Predicate-Object form or the Subject-Predicate form before passing them to the Stanford parser, for the simplification of the relation extraction process. Syntactic reconstruction is naturally a complex process and hence not a perfectly accurate one, due to the ambiguities and uncertainties of natural language. Syntax reconstruction is done as per the following rules as specified in [2]:

1. If there are no verbs in a sentence, do not consider the sentence for further processing.
2. If the sentence contains a semicolon, then only consider the part of the sentence that occurs before the semicolon. Discard the part after the semicolon as extra information. This may not always be the case practically, as there may be useful information after the semicolon, but resolving such are considered in the future scope of the project.
3. Split sentences at their conjunctions. Connectives such as ‘and’, ‘or’, ‘but’ are common in user requirements. For sentences, such as “S1 and S2”, the output of after application of this rule should be two sentences S1 and S2, with proper semantics.
4. Discard prepositional phrase (PP), adjective phrase (ADJP), determiner (DT) or adjective (JJ), if they precedes the subject of the sentence.
5. If NP and VP is preceded by “No”, then convert it into “NP not VP”.
6. Noun phrases (NP) which are separated by connectives like “and, or” are taken as individual sentences. If {{NP1}{VP1{ VBZ NP2,NP3 and NP4}}} then convert it into {{NP1}{VP1{ VBZ NP2 }}}, {{NP1}{VP1{ VBZ NP3}}}, {{NP1}{VP1{ VBZ NP4}}}.
7. If a sentence is of the form {{NP1} {VP1 {NP2} {VP2 {NP3}}}}, then convert it into two sentences like {{NP1} {VP1 {NP2}}} and {{NP2} {VP2 {NP3}}}.

Some additional rules regarding proper resolution of apostrophes in the sentences have also been implemented.
The output of the syntactic reconstruction phase are sentences in Subject-Predicate-Object form or the Subject-Predicate form.

Examples of input-output of sentence simplification

Input Sentence	Rule Applicable	Processed Sentence
System shutdown in sixty seconds.	1	[Ignored]
Spring brings gentle rains and warmer weather; in addition to thunderstorms and hail.	2, 3 The semicolon and the remaining sentence is discarded and the terminator '.' is added. Also, sentence is split at "and".	Spring brings gentle rains and warmer weather. => 1)Spring brings gentle rains. 2)Spring brings warmer weather.
Library issues books and loans to students.	3 Sentence is split at "and" and appended with proper terminators.	1)Library issues books to students. 2)Library issues loans to students.
A player has a name, age and position.	6	1)A player has a name. 2)A player has age. 3)A player has position.
Schools have rooms called libraries.	7 NP1 = Schools, NP2 = rooms, NP3 = libraries, VP1 = have + NP2, VP2 = called + NP3	1)Schools have rooms. 2)Rooms called libraries.
No man is perfect.	5 "Not NP VP" => "NP not VP"	1)Man is not perfect.

B. UML concept extraction:

The aim of this module is to extract concepts according to the requirements document. In this context, heuristics can play a fundamental role to facilitate such task. Usually, candidate classes can be extracted by considering the noun phrases in the requirements text [5].

Class Extraction

Rule 1: Nouns in noun phrases will be considered classes.

Rule 2: Nouns in nouns phrases that are preceded by a possessive relation will not be considered as classes {e.g. A Team has a name, here name will not be considered as a class.}

Rule 3: Only Singular form for the Noun will be stored as a class.

Attribute Extraction

Rule 1: Attributes will be part of noun phrases that have a possessive relationship (has, have, is) with the noun in the preceding Noun Phrase.

Rule 2: Any Noun containing an underscore that contains a entity as prefix, the suffix will we taken as an attribute. E.g. {A Player has a player_id, here id is an attribute of entity player.}

Rule 3: Only singular form of the noun will be stored as attribute.

Multiplicity Extraction

Rule 1: Rule 2 of Class extraction.

Rule 2: Determiners before the number (at least, at most, some, all, each, every, none).

Rule 3: If range is present (to, and, or)

Rule 4: For Rule 3 of Relation extraction, multiplicity will be 1-1.

Rule 5: Plural form of nouns will define n type multiplicity.

Relation Extraction

Rule 1: Determine Verb between classes

Rule 2: Else if determiner is present Rule 2 of Multiplicity, Verb is between class and determiners

Rule 3: in case there are no verbs in between classes, it can be a case of reflexive relation.

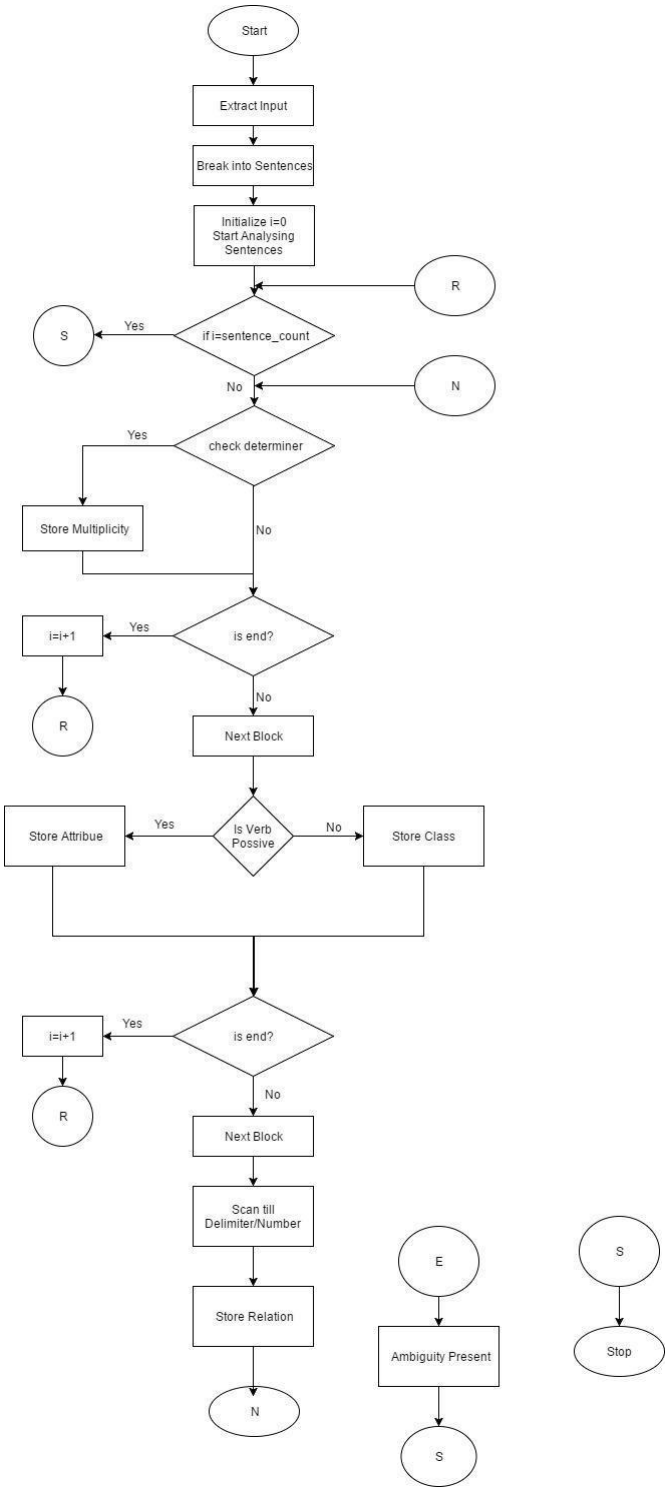
For such relations, the classes are declared at the beginning of the sentence.

Aggregation Extraction

Rule 1: Sentences with copular verbs(is, are, were, was) may imply aggregation between nouns in the subject and object.

Rule 2: Sentences where the verb is "called","known as" implies aggregation.

Flow Diagram:



The Sentences are read as blocks to handle internal conditions, e.g. verb can be comprised of 2 – 3 words. In such cases the words together will be counted as a block. Blocks are also used to detect range for multiplicity extraction: between, from, at least are determiners that can be followed be a range.

Sample Problem Statement:

A HOCKEY_LEAGUE is made up of at least four TEAMS. Each TEAM is composed of six to twelve PLAYERS. A TEAM has a name and a record. PLAYERS have a number and a position. All TEAMS play at least 2 GAMES against each other. Each GAME has a score and a location.

Sentence 1

Input Sentence	Process	Output
Spring brings gentle rains.	Check Determiner -> No Multiplicity	-
Spring brings gentle rains.	Check Class/Entity -> Store Class	Class -> Spring
Spring **brings** gentle rains.	Store Relation (Verb)	brings
Spring brings gentle **rains**.	Check Determiner -> No Multiplicity	4 -
Spring brings gentle **rains**	Check Class/Entity -> Store Class	Class -> Rain (Singular Form)

Result if first iteration: - Spring ---------- Rain.

Sentence 2

Input Sentence	Process	Output
A HOCKEY_LEAGUE is made up of at least four TEAMS.	Check Determiner -> Store Multiplicity	1
A **HOCKEY_LEAGUE** is made up of at least four TEAMS.	Check Class/Entity -> Store Class	Class -> HOCKEY_LEAGUE
A HOCKEY_LEAGUE **is made up of** at least four TEAMS.	Store Relation (Verb)	Made up
A HOCKEY_LEAGUE is made up of **at least four** TEAMS.	Check Determiner -> Store	4.. n

	Multiplicity	
A HOCKEY_LEAGUE is made up of at least four **TEAMS**.	Check Class/Entity -> Store Class	Class -> TEAM

Result of Second Iteration: - HOCKEY_LEAGUE ^{1} ----------$^{4...n}$ TEAM

Sentence 3

Input Sentence	Process	Output
A player has a name.	Check Determiner -> Store Multiplicity	1
A **player** has a name.	Check Class/Entity -> Store Class	Class -> Player
A player **has** a name.	Store Relation (Verb)	Composed of
A player has **a** name.	Check Determiner -> Store Multiplicity	1
A player has a **name**.	Check attribute -> Store attribute	Class -> name

Result of Third Iteration: - A Player (name)

Sentence 4

Input Sentence	Process	Output
Rooms called libraries	Check Determiner -> No Multiplicity	-
Rooms called libraries	Check Class/Entity -> Store Class	Class->Room
Rooms **called** libraries	Store Relation (Verb)	Aggregation(called)
Rooms called **libraries**	Check Determiner -> No Multiplicity	-
Rooms called **libraries**	Check Class/Entity -> Store Class	Libraries

Result of Fourth Iteration: - Room ----→Libraries

Sentence 5

Input Sentence	Process	Output
All TEAMS play at least 2 GAMES against each other.	Check Determiner -> Store Multiplicity	*
All **TEAMS** play at least 2 GAMES against each other.	Check Class/Entity -> Store Class	Class -> TEAM
All TEAMS **play** at least 2 GAMES against each other.	Store Relation (Verb)	Play
All TEAMS play **at least 2** GAMES against each other.	Check Determiner -> Store Multiplicity	2…*
All TEAMS play at least 2 **GAMES** against each other.	Check Class/Entity -> Store Class	Class -> GAME
All TEAMS play at least 2 GAMES **against each other**.	REVERSE MULTIPLICITY	-

Result of 5th Iteration: - TEAMS $^{2...*}$----------* GAME

III. IMPLEMENTATION

We propose to create a GUI program for user to give input and to make necessary changes to input and intermediate output will be stored in XML format. Then we will use the intermediate output to generate class diagrams using Swing API. The program is coded using Java, with the help of IDEA intellij an IDE for java.We use the stanford parser for parsing sentences.

IV. CONCLUSION

Building a system that accepts natural language as input requires some knowledge of proper sentence formation in part of the user who gives the input. Since natural language in still in stages of development where some ambiguities cannot be detected, some part of the correctness of the output also lies with the user who uses gives natural language input, but this reliance must be as minimal as possible.

Our approach to solving a problem that has quite a few solutions already proposed is to minimise the reliance on

correctness of user input and to give correct output by ignoring incorrect input rather than processing them. We believe that on further analysis, we could improve the correctness of the output by adding more cases to detect aggregation and attributes since it is complicated to differentiate attributes from entities(classes) and association from aggregation.

Sentence simplification is the main block of the process as it reduces processing to be done later. We plan to add sentence simplification of any type of compound passive sentences as well.

V. ACKNOWLEDGEMENT

We would like to thank our guide Mr. Rugved Deolekar, Assistant Professor, Vidyalankar Institute of Technology, for his invaluable guidance and suggestions.

REFERENCES

[1] Hatem Herchi, Wahiba Ben, Department of Computer Science, High Institute of Manaagement of Tunis, Tunisia “From user requirements to UML class diagrams”,IEEE journal 2010

[2]Deva Kumar Deeptimahanti, Muhammad Ali Babar “An Automated Tool for Generating UML Models from Natural Language Requirements” IEEE journal 2009

[3]Deeptimahanti Deva Kumar, Ratna Sanyal “Static UML Model Generator from Analysis of Requirements (SUGAR)” IEEE journal 2008

[4]Ambriola, V. and Gervasi, V. “Processing natural language requirements”, Proc. 12th IEEE Intl. Conf. on Automated Software Engineering, pp. 36-45,1997

[5]Subhash K. Shinde, Varunakshi Bhojane, Pranita Mahajan ”NLP based Object Oriented Analysis and Design from requirement specification”, International journal of Computer Applications, June 2012

[6] Priyanka More ,Rashmi Phalnikar Department of IT MIT College of Engineering *International Journal of Applied Information Systems (IJAIS) , April 2012*

[7] L. Mich, NL-OOPs: “From Natural Language to Object Oriented Using the Natural Language Processing System LOLITA.”, Natural Language Engineering, 2(2), 1996, 87.

[8] Xiaohua Zhou and Nan Zhou, 2004, Auto-generation of Class Diagram from Free-text Functional Specifications and Domain Ontology.