European Journal of Scientific Research
ISSN 1450-216X Vol.35 No.1 (2009), pp 22-33

http://www.eurojournals.com/ejsr.htm

Object Oriented Software Modeling Using NLP Based Knowledge Extraction

Imran Sarwar Bajwa
Department of Computer Science and IT
The Islamia University of Bahawalpur, Pakistan
E-mail: imran.sarwar@iub.edu.pk

Ali Samad
Department of Computer Science and IT
The Islamia University of Bahawalpur, Pakistan
E-mail: ali6345@hotmail.com

Shahzad Mumtaz
Department of Computer Science and IT
The Islamia University of Bahawalpur, Pakistan
E-mail: shahzadz22@hotmail.com

Abstract

This paper presents a natural language processing based automated system for NL text to OO modeling the user requirements and generating code in multi-languages. A new rule-based model is presented for analyzing the natural languages (NL) and extracting the relative and required information from the given software requirement notes by the user. User writes the requirements in simple English in a few paragraphs and the designed system incorporates NLP methods to analyze the given script. First the NL text is semantically analyzed to extract classes, objects and their respective, attributes, methods and associations. Then UML diagrams are generated on the bases of previously extracted information. The designed system also provides with the respective code automatically of the already generated diagrams. The designed system provides a quick and reliable way to generate UML diagrams to save the time and budget of both the user and system analyst.

Keywords: OO analysis/design, software engineering, Knowledge Extraction, Natural Language Processing, Automatic Code Generation.

1. Introduction

The fast growing information technology field demands changes in conventional methods of software modeling and designing. Where, these changes has affected the use of existing tools and techniques in the software engineering methodologies, at the same time, the new tools and techniques also have been asked for to cop up with the increasing demands of modern information technology needs. The problem addressed in this research is primarily related to the software analysis and design phase of the software development process. Modern software engineering is based on object oriented design (OOD) paradigm. OOD paradigm encourages use of Unified Modeling Language (UML) for modeling the

user requirements, visual the multiple dimensions and levels of details, document software assets and accommodate incremental development and re-development [1]. UML is a comprehensive and authentic way to of information representation and it is beneficial for the later stages of software development. OOD based software modeling is also called Component Added Software Engineering (CASE) [2]. In recent times, a number of software tools are available for OOD based software modeling and these tools are commonly called Object Modeling tools or CASE tools. These CASE tools provide services to draw UML diagrams, e.g. Rational Rose, Smart Draw, Visual UML, GD Pro, MS Visio, etc.

In a conventional OOD software modeling approach, the system analyst first of all has to do a lot of work for deducing the business logic and understanding the user requirements. After the complete and compact analysis of the business requirements, orthodox CASE tools are used to draw the UML diagrams. These CASE tools have dull nature due to their graphical editor like user interface. The use of these software tools require absolute understanding and expertise to uses these software tools. Keeping in view the requirements of modern software designing, many methods and techniques have been proposed the use of NL text for automatic OO software modeling. Some researchers have proposed primary level of frameworks and systems: GOOAL [18], UML-Rebuilder [21], LOLITA [22], CM-Builder [23], MOVA [33] etc. No one from these tools is able to extract the complete information i.e. classes, objects and their respective, attributes, methods and associations. Some tools just extract names of classes and some of them just deal with objects. K. Li [19] suggested some enhancements to the existing OOA systems but still there was need of a NL-based CASE tool that may have ability of doing automatic analysis of the NL text and extract required information for OO modeling and also generate code from the generated model.

This main idea of research was to design a theory that can comprehensively analyze the natural language text and then implement the theory to develop a software tool UMLG (UML-Generator). UMLG can extract the required information from given piece of natural language text and then afterwards, transform this information into UML class diagram. An additional facility was also provided in the software that it can also convert the user modelling information into the blocks of programming source code. Code generation was made available in two languages; Java and VB.Net. An Integrated Development Environment has also been provided efficient Input and output handling.

This article presents a rule-based system for NL-based OO Software Modeling. It starts by the overview of the previously presented theories and designed systems for NL text based OO analysis and modeling. Next section points the methods of analyzing the natural language text and then a methodology has been presented to generate the software modeling diagrams. Then implementation details have been provided with the analysis of the performed experiments. In the end, Results and analysis have been presented with the advantages and disadvantages of the used approach.

2. Related Work

2.1. Methods for NL Text processing

Natural languages have been an area of interest for researchers for last many decades. In the late nineteen sixties and seventies, so many researchers as Noam Chomsky (1965) [5], Chow, C., & Liu, C (1968) [6] contributed in the area of information retrieval from natural languages. They contributed for analysis and understanding of the natural languages, but still there was lot of effort required for better understanding and analysis. Some authors concentrated in this area in eighties and nineties as, Krovetz, R., & Croft, W. B (1992) [7], Salton, G., & McGill, M (1995) [8], Maron, M. E. and Kuhns, J. L (1997) [9], Losee, R. M (1998) [10]. These authors worked for lexical ambiguity and information retrieval [7], probabilistic indexing [9], data bases handling [10] and so many other related areas.

For information extraction and processing at semantic level, Pedro Domingos [11] presented Markov Logic. Markov logics are simple extension to FOL. There are many applications of Markov logics; link prediction, collective classification, entity resolution, social network analysis, etc. Alchemy

system [12] facilitates implement the inference and learning algorithms. Alchemy is based on a declarative programming language that is similar to Prolog. Markov logic has ability to handle uncertainty and learn form the training data. We also have presented a rule based system [13] that is able to extract desired information from the natural language text. The system understands context and then extracts respective information. This model is further enhanced in this research to capture the information from NL text that is further used for automatic OO modeling.

2.2. Methods of OO Software Modeling

Two decades ago, it was proposed by R. J. Abbot an idea that natural language text can be used for object oriented software modeling. He suggested that the state of a class or an object can be identified by nouns and the behaviour or functionality of a class or object can be identified by verbs [14] in a sentence. On the other hand, H. Buchholz proposed that nouns not only specify classes and objects but also properties [15] of an object or a class. Afterwards, S. Naduri proposed that 'associations' in different objects can be pointed out by verbs [16]. Further detailed categorization of associations was done by N. Juristo into binary association, identification association, n-ary association, etc [17].

Hector also presented a semi-natural language (4WL) [18] in 2002 to automatically generate object models from natural language text. Its prototype tool GOOAL [18] produces OO static and dynamic model views of the problem. K. Li also presented hi his work to solve problems related to NL that can be addressed in OOA [19]. Different NLP based tools have been proposed for this purpose. Nan Zhou proposed another conceptual modeling system based on linguistic patterns [20]. A framework was proposed to generate class diagrams from unstructured system requirement documents.

The discussed methods and techniques are not automatic as they involve system analyst to take many decisions during OO analysis and modeling. On the other hand, these methods were just dealing with only basic OO concepts.

2.3. NLP based UML Case tools

The work discussed in previous section was used by many researchers and they proposed many initial level CASE tool those were using NL text for OO modeling. A. Oliveira used natural language constituents for OO data modeling and presented a CASE tool named REBUILDER UML [20]. This tool integrates a module for translation of natural language text into an UML class diagram. This module uses an approach based on Case-Based Reasoning and Natural Language Processing. But, this case tool needs continuous up gradation of case-base and only deals with class diagrams. On the other hand, if a query related case is not available in case-base, case is not created. LOLITA [21] is another tool to generate an object model from NL text. LOLITA only identifies objects from NL text but it cannot distinguish between classes, attributes and their respective attributes.

There was another significant contribution by Harmain and Gaizauskas as they presented another NL based CASE tool named CM-Builder [22]. This CASE tool was restricted to create a primary class model. There was no appropriate mechanism for confining objects from NL text. Borstler and Overmyer presented another NL based system to generate class diagrams from user requirements given in NL text [23], [24]. MOVA [33] is another tool that models, measures and validates the UML class diagrams. But these systems were incapable of automatically identifying OO constituents. User had to involve in all this procedure to help out the designed system to identify classes, objects and their respective methods and attributes.

3. Used Methodology

Analysis and design of an information system relates to understand and intend the framework of accomplishing the actual job. Typically, design is related to manage and control the complexity by splitting a big task into small ones [2]. Object-oriented design use variables to manifest the state of an object and methods or procedures to implement the behaviour of an object. For example, a ball could

be an object. There can be different parameters related to the stare of this object: shape, colour, size, diameter, type, etc. This object can also exhibit a particular behaviour as throw, roll, catch, hit, etc. The major task in analysis and design phase is to identify the valid classes, objects and specify their states and behaviours and associations among valid classes and objects.

3.1. OOA using NL text

For this purpose, distinct classes have been defined of the various constraints of natural language text. The class of a particular word or phrase is identified by analyzing the type of prepositions used with that particular word or phrase. The defined classes are as following:

1. Object Class
2. Method Class
3. Attribute Class

These main classes have further been classified into sub-classes to handle the ambiguity in the natural language text. To identify these sub classes detailed rules have been defined that have been discuses in the flowing section.

i. Object Class

This class contains the objects and classes in a particular scenario. Objects can be defined in many terms in the natural language text e.g. a person/ thing who is performing some task or a person/ thing for whom some action is being performed or a person/ thing which is under some action. Following can be the possible classes.

i. *Main Actor Object***:** The main actor in a sentence is a person or thing who is performing some action. The ‘subject’ in a sentence is referred as main actor object. For example in the sentence, "User opens the main menu". ‘User’ is the main actor object. But in a passive sentence, the main actor object also may appear, e.g. "The menu is opened by the user."
ii. *Co-Actor Object:* The Co-Actor is a person in a sentence who is doing an action with the main actor. Co-Actor may appear after any token: ‘and’, ‘with’ or a comma. Main and Co-actor carry out the action together. For example in the sentence "Employee and customer performs a transaction", employee is actor and customer is co-actor.
iii. *Recipient Object:* The recipient is the person for whom an action has bee performed. In a sentence, the ‘object’ is nominated as a recipient object. For example in the sentence "Employee guides the customer", customer is recipient or beneficiary.
iv. *Thematic object:* The thematic object is a person/thing in a sentence, on which a certain task is being performed. The thematic object undergoes a change. Often the thematic object is the same as the syntactic direct object. For example in the sentence "Employee makes the bill", ‘bill’ is the thematic object.

ii. Method Class

In method class, all the tokens related to some action are characterized. The action performed in the sentence represents the method or function of an object. For example, “Customer pays the bill”. In this example, ‘pays’ is action or method of object customer, which is the main actor object. Methods can be identified in two different ways:

- Methods or actions in a sentence can appear after the noun or pronoun and there is no helping-verb and the action has ‘s’ or ‘es‘ suffix. For example, “Customer buys a pen”. Here ‘buy’ is a method of object ‘customer’.
- Methods or actions in a sentence can appear after a helping-verb. For example, “Customer is buying a pen”. Here ‘buy’ is a method and has appeared after ‘is’ helping-verb
- Methods or actions in a sentence can appear after a preposition; to, of, etc. For example, “Customer is demanding to book his order”. Here “book” is a method that has appeared after ‘to’ preposition.

iii. Attribute Class

In this class, the attributes of an object are identified. In a sentence, adjectives are attributes of an object. All the adjectives related to an object are nominated as the attributes of that object. For example, "Customer buys the red ball". Here 'ball' is the object and 'red' is its attribute. All the adjectives related to the 'ball' object are characterized its attributes.

3.2. Designed Algorithm

A rule based algorithm was written to analyze NL text and then extract various OO modeling elements. The major steps of the algorithm are as following.

1 In first step, UMLG reads and tokenizes the text containing software requirements by the user e.g. the output of a sentence "The Chair has four legs." is [*Customer*] [*has*] [*purchased*] [*a*] [*red*] [*ball*] [.]

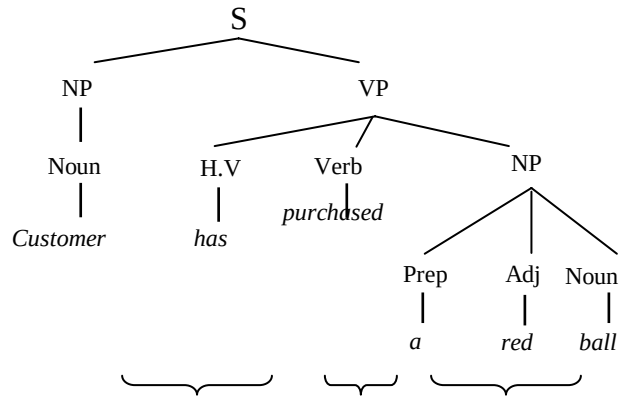
2 In second step, morphological analysis is performed of given text to define the structuring and transformation of the words. POS Tagging is also performed to identify different parts of speech.

[*Customer*] [*has*] [*purchased*] [*a*] [*red*] [*ball*]

Noun H.V Verb Article Adjective Noun

The test is lexical and syntactically analyzed and a parse tree is generated for semantic analysis. Figure 2 shows the generated parse tree [7] of the above example.

Figure 1.0: Parse tree generated for the example



3 Syntactic Analysis carried out to validate phrases and sentence according to grammatical rules defined by the English language. This step also helps in identifying the main parts of a sentence; object, subject, actions, attributes, etc.

[*The*] [*chair*] [*has*] [*four*] [*legs*] [.]

Subject Verb Object

4 In this step, associations are identified by doing semantic analysis. It is determined in this specified that which actions have been performed by which object and a set of attributes belong to which object.

Four Legs Chair

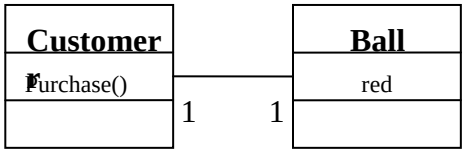
5 Then a rule based module specifies subject nouns as objects, verbs as methods of the objects, and adjectives as attributes of the object. Object nouns are some times specified objects and sometimes as attributes.

6 In this step associations and relationships among extracted classes and objects are performed. Prepositions are major tool for identifying relationships and associations.

7 A logical model of the class diagrams is generates on the basis of previously extracted information.

8 A drawing module converts the logical model into the class diagrams by connecting small pieces of images already stored in database.
9 In next step, associations among generated class diagrams will be also produces.
10 After generating class diagrams, diagrams are labeled with appropriate labels.

Figure 2.0: Final class diagram generated



11 The final step is conversion of logical model to VB.NET and Java Coding.

4. Designed System Architecture

The designed UMLG system has ability to draw UML diagrams after reading the text scenario provided by the user. This system draws diagrams in six modules: Text input acquisition, text understanding, knowledge extraction, generation of UML diagrams and finally multi-lingual code generation. Brief description of all these modules is following:

a. Text Input Acquisition

This module helps to acquire input text scenario. User provides the business scenario in from of paragraphs of the text. This module reads the input text in the form characters and generates the words by concatenating the input characters. This module is the implementation of the lexical phase. Lexicons/ tokens are generated in this module.

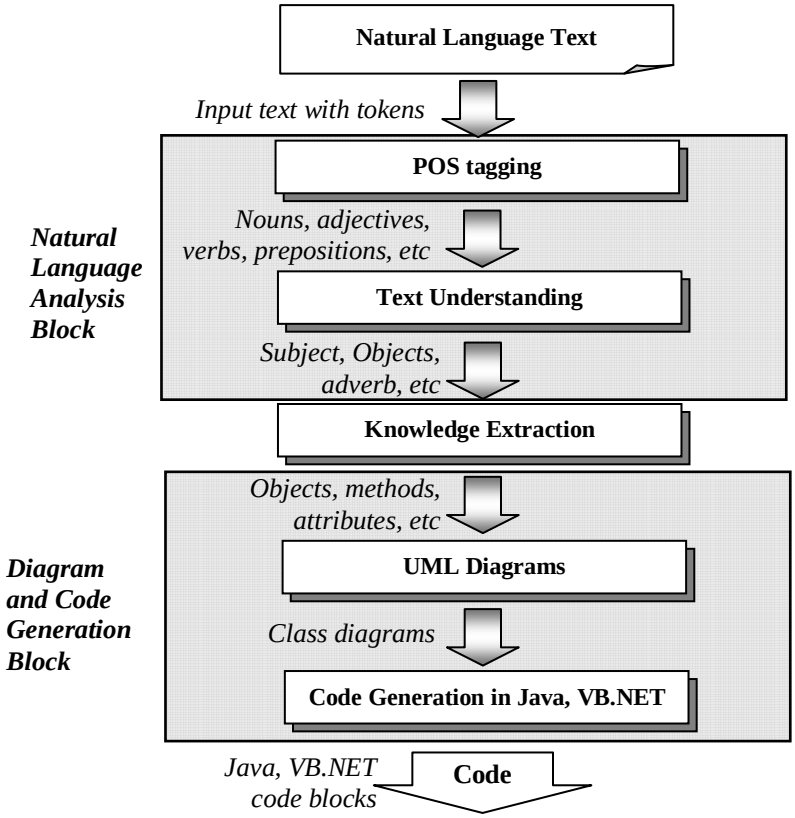
b. POS Tagging

Parts of Speech (POS) tagging [6] is the first phase of text processing. Each token is analyzed and classified into its respective POS classification: Noun, verb, pronoun, adverb, Helping-verb, adjective, prepositions, etc.

c. Text Understanding

This module reads the input from module 2 in from of words and their POS representation. These words are categorized into various further classes as subject, part, object part, verb part, adverb part, etc. This classification helps to find out classes and their respective methods and attributes.

Figure 3.0: Architecture of the designed system



d. Information Extraction

This module, extracts different objects and classes and their respective attributes on the basses of the input provided by the preceding module. Nouns are symbolized as classes and objects and their associated attributes are termed as attributes.

e. Class Diagram Generation

This module finally uses UML symbols to constitute the various UML diagrams by combining available symbols according to the information extracted of the previous module. As separate scenario will be provided for various diagrams as classes, sequence, activity and use cases diagrams, so the separate functions are implemented for respective diagram.

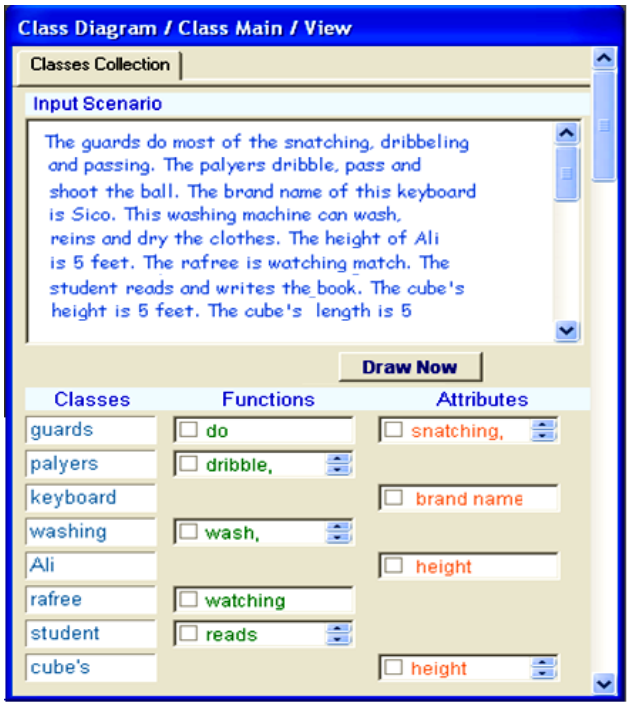
f. Java Code Generation

This is the last module, which ultimately generate code in the different popular languages as Java, C#.Net and VB.Net to help the programmer. The generated code is structured according to the knowledge extracted in the previous modules and the UML diagrams generated.

5. Experiments and Results

This section describes results of some preliminary experiments. A number of experiments were performed to check the performance of the designed system. Following is the set of phases those were followed during the experiments. First of all there is Class Diagram window. The window has been shown in figure 4.0.

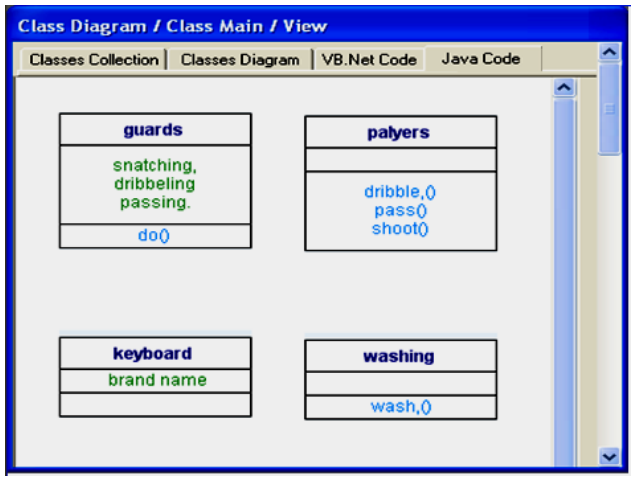
Figure 4.0: Extracting classes, functions and their attributes



Class Diagram window is the preliminary window, where the user types the natural language text related to his required business scenario. Given text is in English language and follows all the primary grammatical rules [9] for better accuracy in the output. In first step, the available classes/ objects in the given text are extracted with their related attributers and methods. Here an extra facility has been provided to the software designer that he may select or de-select the extracted attributes and methods for his required classes. Diagrams are generated for only his prescribed classes/ objects.

In next step, the actual class diagrams are generated in the class diagram output window. All the class diagrams also have their respective methods and attributes. If the software designer is not satisfied with the generated output window, he can go to the previous window and change the specification of the diagram and can re-generate the class diagrams according his required specification. The output has been shown in the following figure 5.0.

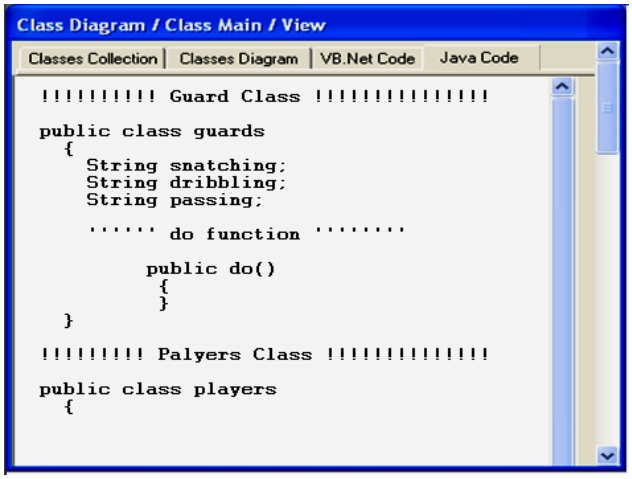
Figure 5.0: Generating class diagrams



With the diagrams generation the respective code in Java and VB.Net language code is also generated. If software designed re-generates the diagrams, the code will also be re-generated according

to the new specification provided by the software designer. Figure 6.0 shows the generated code in Java language.

Figure 6.0: Generating Code in Java Language



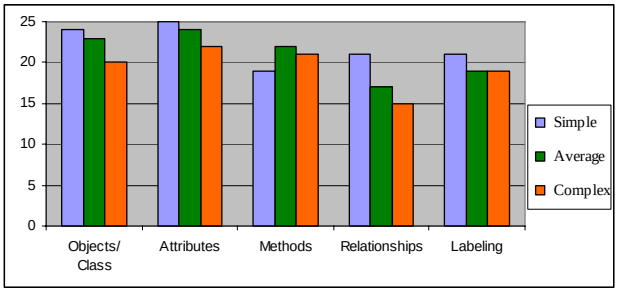
To test the accuracy of the diagrams generated by the designed system four parameters had been decided. Each generated class diagram is tested under: no. of objects and classes, no. of attributes, no. of methods, no. of associations and diagram labeling. Maximum score was declared 25. According to the wrong nominations and extractions, the points were counted. A matrix of results of generated diagrams is shown below.

Table 1.0: Testing results of Class UML Diagrams

Scenario Type	Objects/Classes	Attributes	Methods	Relationships	Labeling	Total
Simple	24	25	19	21	21	87%
Average	23	24	22	17	19	84%
Complex	20	22	21	15	19	80%

A matrix representing UML diagrams accuracy test (%) for class, activity, sequence and use case diagrams has been constructed. Overall diagrams accuracy for all types of UML diagrams is determined by adding total accuracy of all categories and calculating average of it. Following graph is showing the accuracy ratio of various diagram types in terms of objects, attributes, sequence and labeling parameters.

Graph 1.0: Testing results of Class UML Diagrams



6. Conclusion and Future Work

This research is all about the dynamic generation of the UML diagrams and their respective code by reading and analyzing the given scenario in English language provided by the user. The designed system can find out the classes and objects and their attributes and operations using an artificial intelligence technique such as natural language processing. Then the UML diagrams such as Activity dig., Sequence dig., Component dig., Use Case dig., etc would be drawn. The given scenario should be complete and written in simple and correct English. Under the scope of our project, software will perform a complete analysis of the scenario to find the classes, their attributes and operations.

A graphical user interface has also been provided to the user for entering the Input scenario in a proper way and generating class diagram. The designed system for generating UML diagrams and their respective code in multiple languages typically Java was started with the aims that there should be a software which can read the scenario given in English language and can draw the all types of the UML diagrams such as Class diagram, activity diagram, sequence diagram, use case diagram. But other types except class diagram are still under working. We need improved text analysis algorithms to generate complex UML diagrams i.e activity and sequence diagrams. Current accuracy of generating diagrams is about 80% to 85%. It can be enhanced up to 95% by improving the algorithms and inducing the ability of learning in the system.

Acknowledgement

The research project is sponsored by HEC, Pakistan, as part of PhD studies scholarship. I am also thankful to Dr. Irfan Hayder and Dr. M.A. Choudhary for their technical support and guidance during this project work.

References

[1] Joseph Schmuller (1991) "Sams Teach Yourself UML", *TechMedia*, 1999

[2] Rumbaugh, J., I. Jacobson, and G. Booch, (1998) "The Unified Modeling Language Reference Manual" *Reading, MA: Addison-Wesley*. 1998

[3] Daniel Jurafsky, J.H.M. (2000) "Speech and Language Processing", *Prentice Hall*

[4] Zahariev, M., [2004] "A linguistic approach to extracting acronym expansions from text", *KAIS: Knowledge and Information Systems* 6(3), pp. 366-373.

[5] Chomsky, N. (1965) "Aspects of the Theory of Syntax. *MIT Press, Cambridge*, Mass, 1965.

[6] Chow, C., & Liu, C. (1968) "Approximating discrete probability distributions with dependence trees". *IEEE Transactions on Information Theory*, 1968, IT-14(3), 462–467.

[7] Krovetz, R., & Croft, W. B. (1992) "Lexical ambiguity and information retrieval", *ACM Transactions on Information Systems*, 10, 1992, pp. 115–141

[8] Salton, G., & McGill, M. (1995) "Introduction to Modern Information Retrieval" *McGraw-Hill, New York*., 1995

[9] Maron, M. E. & Kuhns, J. L. (1997) "On relevance, probabilistic indexing, and information retrieval" *Journal of the ACM*, 1997, 7, 216–244.

[10] Losee, R. M. (1988) "Parameter estimation for probabilistic document retrieval models". *Journal of the American Society for Information Science*, 39(1), 1988, pp. 8–16.

[11] S. Kok and P. Domingos, (2005) "Learning the structure of Markov logic networks', *In Proc. of ICML-05, Bonn, Germany, 2005. ACM Pres*, pp 441–448

[12] S. Kok, P. Singla, M. Richardson, and P. Domingos, (2005) "The Alchemy system for statistical relational AI", *Technical report, Department of Computer Science and Engineering*, http://www.-cs.washington.edu/ai/alchemy.

[13] Imran Sarwar Bajwa, M. Abbas Choudhary, (2006) "A Rule Based System for Speech Language Context Understanding" *Journal of Donghua University, (English Edition)* 23 (6), pp. 39-42.

[14] R.J. Abbott, (1983) "Program Design by Informal English Descriptions", *Communications of the ACM*, Nov. 26(11), pp. 882-894.

[15] H. Buchholz, A. Dusterhoft, B. Thalheim, (1996), "Capturing Information on Behavior with the RADD_NLI: A Linguistic and Knowledge Base Approach", *Proc. Second Workshop Application of Natural Language to Information System*, IOS Press pp. 185-196

[16] S. Naduri, S. Rugaser (1994), "Requirements Validation via Automated Natural Language Parsing", *Proc. 28th Hawaii Int'l Conf. Systems Science: Collaboration Tech., Organizational Systems, and Technology*, IEEE Computer Society, pp. 362-367.

[17] N. Juristo and A.M. Moreno, "How to Use Linguistic Instruments for Object-Oriented Analysis", *IEEE Software*, May/June 2000, pp. 80-89

[18] Hector G. Perez-Gonzalez, (2002) "Automatically Generating Object Models from Natural Language Analysis", *17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ACM New York, USA, pp: 86 – 87

[19] K. Li, R.G.Dewar, R.J.Pooley, [2003] "Object-Oriented Analysis Using Natural Language Processing", *www.macs.hw.ac.uk:8080/techreps/docs/files/HWMACS-TR-0033.pdf*

[20] Nan Zhou, Xiaohua Zhou, (2004) "Automatic Acquisition of Linguistic Patterns for Conceptual Modeling", *INFO 629: Artificial Intelligence*, Fall 2004

[21] António Oliveira, Nuno Seco and Paulo Gomes (2004) "A CBR Approach to Text to Class Diagram Translation", *TCBR Workshop at the 8th European Conference on Case-Based Reasoning*, Turkey, September 2006.

[22] L. Mich, R. Garigliano, (1996) "A linguistic approach to the development of object-oriented system using the NL system LOLITA", *Object Oriented Methodologies and Systems, (ISOOMS)*, LNCS 858, pp. 371-386.

[23] H. M. Harmain and R. Gaizauskas, (2003) "CM-Builder: A Natural Language-based CASE Tool", *Journal of Automated Software Engineering*, 10, 2003, pp. 157-181

[24] S.L.V. Overmyer, (2001) O. Rambow, "Conceptual Modeling through Linguistics Analysis Using LIDA", *23rd international conference on Software engineering*, July 2001

[25] J. Borstler, (1996) “User-Centered Requirements Engineering in RECORD - An Overview”, *the Nordic Workshop on Programming Environment Research, Proceedings* NWPER'96, pp. 149-156.

[26] Hugo Liu and Push Singh [2004] “Commonsense reasoning in and over natural language” Proc. 8th International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES-2004)

[27] P. Sturt, F. Costa, V. Lombardo, and P. Frasconi, (2003) “Learning first-pass structural attachment preferences using dynamic grammars and recursive neural networks,” *Cognition*, vol. 88, pp. 133–169

[28] Power, R., Scott, D. & Hartley, A. [2003] "Multilingual Generation of Controlled Languages" in *Proc. 4th Controlled Language Applications Workshop* (CLAW03), Dublin, Ireland.

[29] P. C. R. Lane and J. B. Henderson, (2001) “Incremental syntactic parsing of natural language corpora with simple synchrony networks,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 2.

[30] Mich, L. (2001) "Ambiguity Identification and Resolution in Software Development: a Linguistic Approach to improve the Quality of Systems" in *Proc. Of 17th IEEE Workshop on Empirical Studies of Software Maintenance*, Florence, Italy

[31] F. Costa, P. Frasconi, V.Lombardo, and G. Soda, (2003) “Towards incremental parsing of natural language using recursive neural networks,” Applied Intelligence, vol. 19, no. 1–2, pp. 9–25.

[32] J. Henderson, (2003) “Neural network probability estimation for broad coverage parsing,” in *Proc. of 10th conference of the European Chapter of the Association for Computational Linguistics* (EACL 2003), pp. 131–138.

[33] Manuel Clavel, Marina Egea, Viviane Torres da Silva, (2007) “The MOVA Tool: A Rewriting-Based UML Modeling, Measuring, and Validation Tool”, *in Proc. 12th Conference on Software Engineering and Databases* Zaragoza (Spain), 2007.