2009 IEEE/ACM International Conference on Automated Software Engineering

An Automated Tool for Generating UML Models from Natural Language Requirements

Deva Kumar Deeptimahanti, Muhammad Ali Babar
Lero, University of Limerick, Ireland
{Deva.kumar.deeptimahanti, malibaba}@lero.ie

Abstract— **This paper describes a domain independent tool, named, UML Model Generator from Analysis of Requirements (UMGAR), which generates UML models like the Use-case Diagram, Analysis class model, Collaboration diagram and Design class model from natural language requirements using efficient Natural Language Processing (NLP) tools. UMGAR implements a set of syntactic reconstruction rules to process complex requirements into simple requirements. UMGAR also provides a generic XMI parser to generate XMI files for visualizing the generated models in any UML modeling tool. 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.**

Keywords- Requirement engineering, Natural Language Processing, Unified Modeling Language

I. Introduction

Software requirements are often specified in natural language (NL). However, requirements specified in NL can often be ambiguous, incomplete, and inconsistent. Moreover, the interpretation and understanding of anything described in NL has the potential of being influenced by geographical, psychological and sociological factors. It is the job of requirements analysts to detect and fix any potential ambiguities, inconsistencies, and incompleteness in the requirements specifications documents.

However, human reviewers can overlook some defects while reading complex NL descriptions which can lead to multiple interpretations and difficulties in recovering implicit requirements when the requirement analyst does not have extensive domain knowledge. Hence, tool support for automating some of the tasks involved in this activity is highly desirable.

Li [1] has provided a comparison of approaches and tools proposed in this area. As per this paper, most of the approaches have varying level of automation, in which most of the tools generate analysis class models but needs human intervention to generate these models. These work for a small set of requirements (<200 words) and do not provide any normalization of the requirement text, which can result in the loss of information while processing large requirement documents. While complete automation of this activity with NLP appears to be impossible, more advanced automation is achievable. To address the limitations of previous tools, we have developed a tool for providing complete automated support for developing both static and dynamic design models from NL requirements.

II. The Approach

UMGAR follows the Object- Oriented Analysis and Design (OOAD) [2] approach for object elicitation from requirements described in NL to generate analysis and design class models by following an approach based on a combination of the Rational Unified Process (RUP) [3] and ICONIX process [4] using the most efficient NLP tools. The current version of UMGAR can generate the Use-case diagram, analysis class model (conceptual model), collaboration diagram, and the design class model.

A. Theoretical Foundations

The Noun-Phrase technique [5] of RUP is aimed at helping in developing the analysis class model. This technique helps a requirements analyst to identify all possible objects from a given requirements document and generate analysis class model by attaching attributes and methods with the associated object. This technique categorizes a list of nouns into three classes, namely relevant classes, fuzzy classes and irrelevant classes. Fuzzy classes are further sub-divided into adjective, attribute and redundant classes.

The ICONIX process [4], a derivative of RUP, supports robustness analysis to identify analysis classes and stereotypes, which should be known prior to generating a collaboration diagram. RUP [3] also provides a Use-case driven, sequence/collaboration modeling approach for generating a collaboration diagram. We have combined both of these approaches, provided by RUP and ICONIX, to develop our tool for the automatic generation of collaboration diagrams from which design class models can be generated.

B. Overview

UMGAR aims to assist developers in generating analysis and design class models from NL requirements expressed in active voice. Some of the key features of UMGAR are:

1. This tool has been developed using the most efficient NLP technologies like Stanford Parser [6], JavaRAP [7], and WordNet2.1 [8], which can handle large requirements documents.
2. UMGAR uses a glossary to avoid any communication gaps among team members to create unambiguous requirements.

3. UMGAR generates XMI file according to XMI v2.1 version [9] which can be imported in any UML modeling tool which has import XMI file option to visualize a UML model.
4. Extra rules are formulated and implemented to overcome the limitations of NLP tools in handling large NL SRS, like 8 syntactic reconstruction rules [10] for reducing ambiguity in requirements, 21 rules to handle compound word morphological analysis where WordNet [8] fails, and identified 247 determiners which are specific to requirements engineering context.
5. Key-Word-In-Context (KWIC) feature helps to identify the context of a particular OO element in the requirement document with an aim to provide traceability between requirements and models. .
6. We have proposed 8 rules for generating a Collaboration diagram from a Use-case Specification template covering major sentences which occur in Use-case specifications.
7. Java code model is generated for each developed design class model using code generation feature of Enterprise Architect.
8. UMGAR implements Concept location feature [11] to enhance traceability between requirements and code by providing search functionality for a particular requirement in source code.

C. *Process Architecture of UMGAR*

The process architecture of UMGAR consists of two main components for model generation from NL requirements, namely:

1) Normalizing requirements component (NLP Tool Layer)

This component aims at normalizing NL requirements to remove ambiguous requirements and identify incomplete requirements. This component consists of the following sub components:

a) Syntactic Reconstruction

The tool takes stakeholder's requests as input and performs syntactic reconstruction to split a complex sentence into simple sentences to extract all possible information from the requirements document. We have defined 8 syntactic reconstructing rules that have been implemented in UMGAR. The tool scans each sentence to test whether that requirement satisfies the Statement sentence structure, which is of the form "Subject: Predicate" or "Subject: Predicate: Object", and applies rules accordingly. If a sentence does not satisfy the proposed rules, then it prompts a message to the user to change the sentence accordingly to the statement structure.

b) NLP Technologies Used

The following are the NLP tools used for developing UMGAR:

1. **Stanford Parser** [6] is used to generate a parse tree for each requirement up to 40 words in length, from which relevant information like actors, use-cases, classes, methods, associations, attributes etc., can be easily extracted. This parser avoids usage of various other NLP tools like using tokenization, sentence splitting and part-of-speech (POS) tagging tools.
2. **WordNet2.1** [8] is a large English language lexical database in which nouns, verbs, and adjectives are grouped into cognitive synonyms (Synsets). It helps in performing morphological analysis for converting plurals into singulars.
3. **JavaRAP** [7] helps resolve pronouns up to third person pronouns. JavaRAP is used in UMGAR to replace all the possible pronouns with its correct noun form. This tool is quite helpful when analyzing large requirement documents.

2) Model Generator component

This component aims at generating various OO models using normalized requirements. This component consists of the following three sub-components:

a) Use-case Model Developer

This takes stakeholder's requests as input, which are basically functional requirements which are processed using proposed syntactic reconstruction rules and will be of the form Subject-Predicate-Object or Subject-predicate, in which UMGAR identifies subject and objects as actors, and predicate as use-cases with association between actors and use-cases using the parse tree generated from Stanford Parser [6].

b) Analysis class model developer

The analysis model serves as an abstract model for the design model, by realizing use-cases. This uses a Noun-phrase technique [5] which categorizes the noun list into relevant, attribute, adjective and irrelevant classes to generate an analysis class model. Stanford Parser [6] is used to identify all candidate objects from a given requirements document and generates an analysis class model by connecting attributes and methods with the associated object. The glossary is used to eliminate redundant classes and morphological analysis is performed using WordNet [8] to eliminate ambiguity.

c) Design class model developer

Design model serves as an abstract model for the implemented model, by realizing use-case specifications. Each use-case identified during Use-case modeling should be manually specified using the rule "Who do what to whom?" This describes who initiates the message, and what it wants to send and to whom, according to the Use-case Specification Template expressed in active voice form and processed using 8 syntactic reconstruction rules [10]. The collaboration diagram is generated by using proposed rules as follows:

1. Subject (Noun Phrase) in the sentence is considered as sender object.
2. Object (Noun Phrase) is considered as receiver object. And Predicate (VP) can also contain Noun Phrase which can be treated as receiver object based on the Verb Phrase (VP) structures.
3. The verb phrase between subject and object is taken as message passed between objects.

4. If sentence is having subject and predicate, with out any object, then sequence stated in the Use-case specification helps to identify the relation between both messages.
5. Conditional statements represent sequence of statements; and can be handled by keeping If clause at the beginning of the sentence and an end_If clause at the end of the sentence.
6. Concurrent statements show sequence of actions to be performed at the same time, and are handled by keeping Start_ConCurrent clause at the beginning and End_Concurrent clause at the end of the concurrent statements.
7. Iterative statements are handled using Start_While statement at the beginning and End_While at the end of the iterative statements.
8. Synchronization statements are handled by keeping Start_Sync word after the first sentence to show the synchronous message started and after the last sentence End_ Sync word is used.

Stanford Parser is used to parse both basic and alternative flows in the Use-case specification template to identify sender, receiver and the messages between them. UMGAR generates a design class model from the generated collaboration diagram, in which actors and identified objects in the collaboration diagram are considered as design classes. Messages between objects are extracted as methods associating them with corresponding classes using Stanford Parser and association relationships from event flow sequences in the Use-case specification. JavaRAP [7] is used to eliminate pronouns during the flow of events. Finally, UMGAR generates Java based code model using the Enterprise Architect code generation feature for demonstrating traceability between requirements and code using concept location feature [11]. Fig. 1 shows UMGAR's Process Architecture.

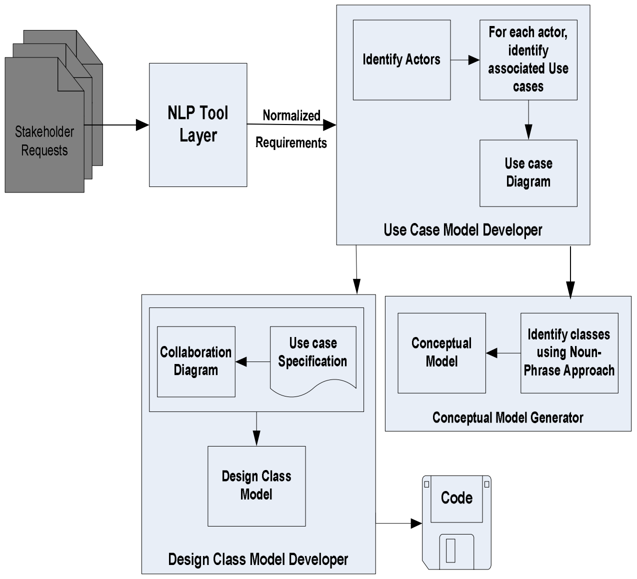

Figure 1. Process Architecture of UMGAR

III. CONCLUSIONS

This demonstration of UMGAR aims to show the process of generating UML models from complex NL requirements, by using proposed syntactic reconstruction rules and available NLP tools to extract required OO artifacts like use-cases, actors, classes, operations and attributes.

The unique features of UMGAR are the underlying methodology for generating use-case and analysis class models from NL requirements and collaboration and design class models from Use-case specifications along with proper relationships. UMGAR's advantages are the rapid generation of UML models with proper relationships, and the process of handling domain knowledge using efficient NLP tools. UMGAR is able to visualize UML diagrams in any UML modeling tool which has the XMI import facility. We plan to extend this work to automatically generate state chart diagrams to test the class models without the need for generating code.

ACKNOWLEDGMENT

The authors acknowledge Dr. Ratna Sanyal for her initial contribution to this work. This tool research was later supported by the Programme for Research in Third-Level Institutions (PRTLI) funded by the Higher Education Authority (HEA), Republic of Ireland.

REFERENCES

[1] K.Li, et al., Object-oriented Analysis using Natural Language Processing, T. R. HW-MACS-TR-0033, Heriot-Watt University, 2005.
[2] Simon Bennett, et al., Object-Oriented Systems Analysis and Design Using UML, McGraw Hill, 2005.
[3] P. Kruchten, The Rational Unified Process An Introduction, Second Edition, Addison Wesley, 2000.
[4] Doug Rosenberg and K. Scott, Use Case Driven Object Modeling With UML: A Practical Approach, Addison-Wesley, 1999.
[5] Rebecca Wirfs-Brock, et al., Designing Object-Oriented Software, Prentice-Hall, 1990.
[6] D. Klein and C. Manning, "Stanford Parser 1.6," 2007; http://nlp.stanford.edu/software/lex-parser.shtml.
[7] L. Qiu, "JavaRAP," 2004; http://www.comp.nus.edu.sg/~qiul/NLPTools/JavaRAP.html.
[8] G.A. Miller, "WordNet2.1," 2006; http://wordnet.princeton.edu/.
[9] OMG, "XML Metadata Interchange," 2007; http://www.omg.org/technology/documents/formal/xmi.htm
[10] D. Deva Kumar and R. Sanyal, "Static UML Model Generator from Analysis of Requirements (SUGAR)," Proc. Advanced Software Engineering and Its Applications, 2008. ASEA 2008, 2008, pp. 77-84.
[11] A. Marcus, et al., "An Information Retrieval Approach to Concept Location in Source Code," Proc. Proceedings of the 11th Working Conference on Reverse Engineering, IEEE Computer Society, 2004, pp. 214-223.