Towards Software Requirements Extraction Using Natural Language Approach

AMJAD HUDAIB, BASSAM HAMMO, YARA ALKHADER
DEPARTMENT OF COMPUTER INFORMATION SYSTEMS
University of Jordan
Amman 11942 Jordan
JORDAN

Abstract: - In this paper, we present an automated support environment to reduce the time and efforts required to produce and maintain a reusable specification document. Our proposed model has two operation modes: the first one is the forward mode in which our model automatically converts English natural language requirements into UML class diagram models. While the second one is the backward mode in which our model automatically reverses UML class diagram models into English natural language requirements. We compared our model with previous models and the results are promising.

Key-Words: - Requirements Engineering, UML Class Diagram, Natural Language Processing, Specification Document, Software Design.

1 Introduction

Requirement engineering is the first step toward building software [12] [13]. Requirement engineering main concerns are to establish, structure and model software specifications into written documents namely specification documents. Those documents serve as the mean of communication between different stakeholders of software [2, 14].

To realize the importance of the specification document many of the work in requirement engineering field has been targeting the way specification documents describe their requirements [1, 11].

Although there were many attempts to use formal and semiformal languages for describing specification documents, the use of the informal natural languages remains the most widely in use [6] [5]. However, sever problems emerged while using natural language in specification documents: first of all, they can be ambiguous, inconsistent and incomplete [10] [14]. Secondly, they are never understood by computers directly without preprocessing [10].Therefore, some reinterpretations of the natural language requirements are usually conducted by the requirements engineer before proceeding with system design and development [6]. This reinterpretation is non-trivial and error prone. It needs a considerable amount of experience and it is time consuming. What makes it even implausible is the fact that requirements evolves in order to reflect real world changes. This change in the specification document requires reinterpreting the specifications into models and updating the software accordingly [9].

In our work, we present a model that has two operation modes: The forward operation mode which automates the reinterpretation of natural language requirements into UML class diagram models. The second mode is the backward operation mode which automatically reveres the models into natural language requirement specifications. The advantage from our operation scheme is to provide seamless model - natural language view.

2 Related Works

Reference [6] used the eXtensible Markup Language (XML) and Two Level Grammar (TLG) to transform natural language into the formal object oriented Vienna Development Method (VDM++). The main concern of their work was to automate the management of formal requirements keeping them compatible with their natural language counterpart.

However, in [1] they built a requirements engineering supporting environment that analyzes and synthesize different views given requirements written in natural languages. In their work, they used a shared repository and multiple viewers and modelers to provide different interfaces for the given natural language requirements. Where in [9], the authors automated the transformation of natural language into the semiformal Unified Modeling Language (UML) using role based technique, which is a conceptual model used to produce object oriented static views. In fact, they first translated

natural language requirement into the 4W language which is a constrained English language and then they used the generated set of requirements expressed using the 4W language as input to their automation process.

Reference [10] investigated the ability to determine software functionalities from software requirements specifications expressed in natural languages. The authors illustrated the deficiencies and pointed the difficulties in processing natural languages. The objective of the study was to develop criteria for identifying functions. In fact, they illustrated that the use of a simple method of determining functions is not productive.

A Knowledge-Based Natural Language System (KBNL) was introduced in [3]. The system presumes the existence of a model that describes the world and how language relates to the world. The system parses the English expression analysis theme and then it converts it into the knowledge base representation.

3 The Model Architecture

Fig.1 depicts the architecture of our model. The model is made of three different layers. The first layer is responsible of preprocessing natural language requirement specifications. This layer interacts with the natural language requirements repository in which natural language is stored. The second layer is the core of our model in which most of the processing is performed. At this layer, an XML representation is generated for the inputted natural language requirements. This representation is stored in the XML requirements repository. The third layer responsible of generating the UML diagrams from the XML representations. Afterward, the generated UML diagrams are stored in the UML class diagram repository.

Our model operates in two modes: the forward mode and the backward mode. Fig.2 represents the two operation modes and their components.

Next we describe the main components of our model.

- *The natural language preprocessor*: it is responsible for ensuring that the inputted requirements are free of spelling errors and are well structured in terms of using punctuations.
- *The natural language processor*: this component generates XML representation for the given requirements. In this representation, natural language tokens are annotated with metalanguage presenting their part of speech and their part of sentence. Fig.3 presents a natural language requirement sample collected from [6]; its XML representation is depicted in Fig.4.

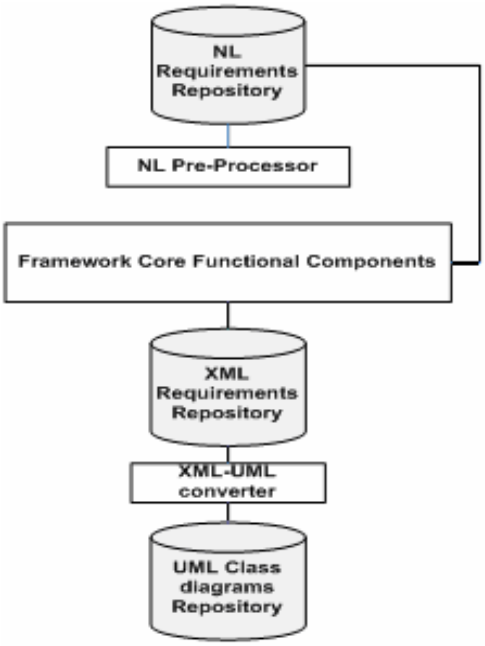

Fig.1. The Model high level architecture

- *The manual domain processor*: in this process the requirements engineer uses domain knowledge to manually eliminate redundancy and resolve similarities.
- *The rule based functional analyzer*: this process represents the core of our model. In this process, a set of rules are used to resolve ambiguity problems frequently occurring in specification documents namely: compound names, collections of objects, pronouns, connectors and relations. The output of this process is an enhanced and modified XML representation than the previously generated one.
- *The XML schema mapper*: this process generates an XML schema for the processed XML representation; the schema represents the mapping between the XML structure and our targeted model which is the UML class diagram. Fig.5 presents the XML schema generated for the XML requirements in Fig.3. The XML schema representation is used for the generation of both the class diagram and natural language depicted in Fig.6 and Fig.7 respectively.
- *The natural language extractor*: this process uses a set of rules describing the creation of English statements in order to create simple and meaningful statements out of the XML schema.

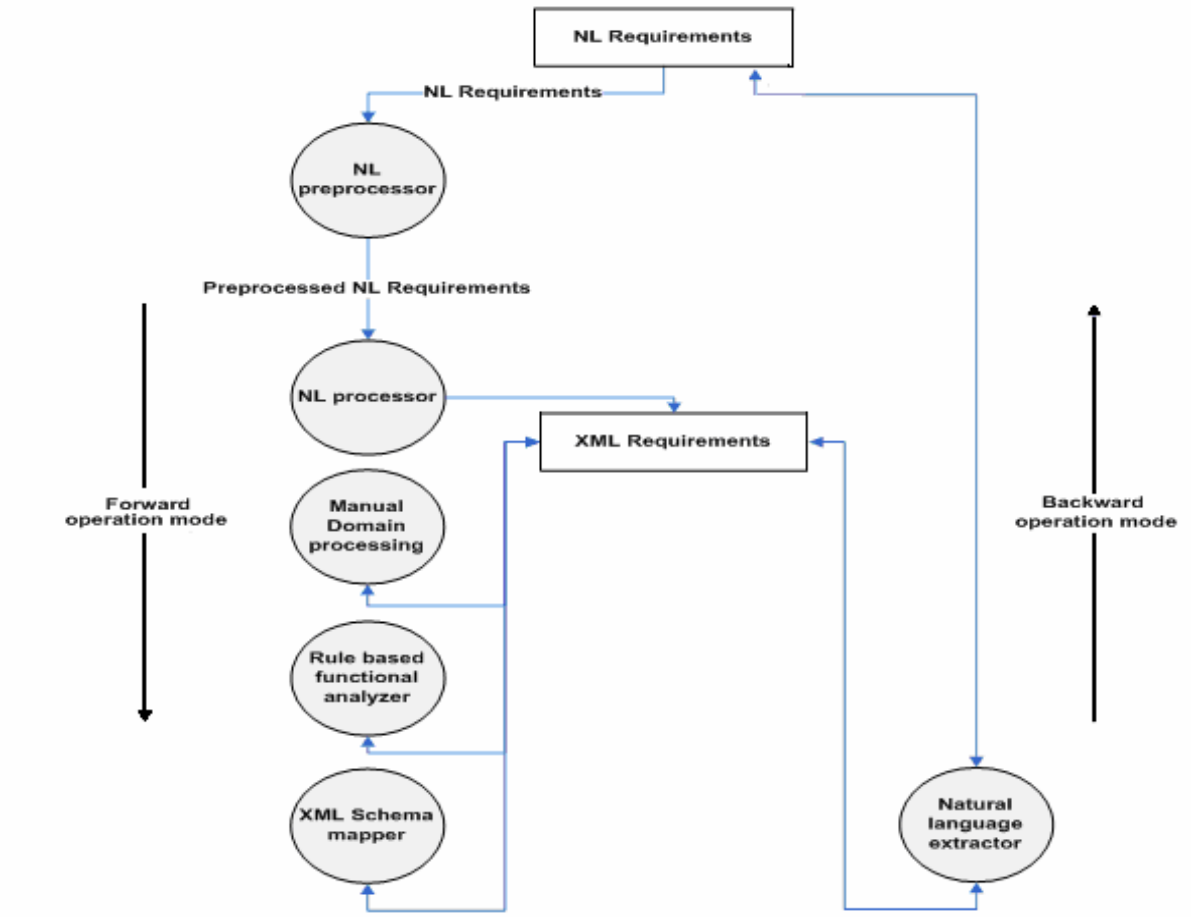

Fig.2. The Model data flow diagram illustrating the model operation modes

The hotel consists of a number of rooms. Every room has a number and a status.

Fig.3. Sample natural language requirements

```
<?xml version="1.0" encoding="UTF-8"?>
<paragraph>
 <Token category="DT">The</Token>
 <Token category="NN" POS="s">hotel</Token>
 <Token category="VBZ" POS="v">consists</Token>
 <Token category="IN">of</Token>
 <Token category="DT">a</Token>
 <Token category="NN" POS="obj">number</Token>
 <Token category="IN">of</Token>
 <Token category="NNS">rooms</Token>
 <Token category=".">.</Token>
 <Token category="DT">Every</Token>
 <Token category="NN" POS="s">room</Token>
 <Token category="VBZ" POS="v">has</Token>
 <Token category="DT">a</Token>
 <Token category="NN" POS="obj">number</Token>
 <Token category="CC" Conn="noun">and</Token>
 <Token category="DT">a</Token>
 <Token category="NN">status</Token>
 <Token category=".">.</Token>
</paragraph>
```

Fig.4. XML representation for the requirements in Fig.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="hotel">
  <xs:annotation>
   <xs:documentation>0, 0</xs:documentation>
  </xs:annotation>
  <xs:attribute name="of number of rooms">
   <xs:annotation>
    <xs:documentation>2, 2</xs:documentation>
   </xs:annotation>
  </xs:attribute>
 </xs:complexType>
 <xs:element name="hotel" type="hotel">
  <xs:annotation>
   <xs:documentation>0, 0</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:complexType name="room">
  <xs:annotation>
   <xs:documentation>4, 4</xs:documentation>
  </xs:annotation>
  <xs:attribute name="number">
   <xs:annotation>
    <xs:documentation>6, 6</xs:documentation>
   </xs:annotation>
  </xs:attribute>
  <xs:attribute name="status">
   <xs:annotation>
    <xs:documentation>8, 8</xs:documentation>
   </xs:annotation>
  </xs:attribute>
 </xs:complexType>
 <xs:element name="room" type="room">
  <xs:annotation>
   <xs:documentation>4, 4</xs:documentation>
  </xs:annotation>
 </xs:element>
</xs:schema>
```

Fig.5. XML schema generated for requirements

In our model, we used the MIMB tool [8] to transform the XML schema into a UML class diagram and vice versa. We also used the GATE tool [4] as our NLP infrastructure.

4 Running the Experiments

In order to test the performance of our model, we collected a test bed of specification documents: Our test bed is divided into two sets. The first set of documents namely: the dinning philosopher specifications document [9], the bank system specifications document [6] and the elevator specifications document [11] were collected from different academic researches where each of the documents has been used to conduct a research similar to our work. The second set of requirement documents was collected from Computer Information Systems (CIS) students in a Software Engineering (SE) course at the University of Jordan.

We classified the specification documents in our test bed into categories depending on their language characteristics. The main factor in the classification was the structure of the sentences used: if a sentence was atomic with one subject and one verb then it is considered as a simple sentence. Accordingly, the specification document which is formed of simple sentences is classified as a simple one. On the other hand, if a sentence contained more than one subject and multiple verbs then it is considered as an intermediate sentence. Accordingly, the specification document which is composed of intermediate sentences is classified as intermediate. Whereas, sentences with more than one subject and multiple verbs are complex and documents composed of them are classified as complex as well.
Table 1 and Table II illustrate our classification for each of the specification documents in our test bed.
Our classification process is preformed manually however in future works we plan to automate this process.
In our experiments, each of the documents in the first test bed was inputted into our model without any modifications in order to be able to compare our results with the results obtained in the previous works [9], [6], and [11]. For each document of this set we generated a class diagram view and a natural language view. Afterward, we analyze the gap between the inputted natural language, the generated class diagram and the generated natural language view.
Our gap analysis was a two step procedure: first, we regenerated the class diagram from the reversed engineered natural language.

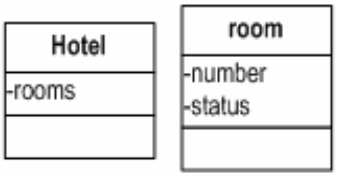

Fig.6. Class diagram view for requirements in Fig.3

Each hotel has a number of rooms.
Each room has a number,
Each room has status.

Fig.7. Sample natural language requirements in Fig.3

TABLE I
Test bed classification of specification documents

Simple	Intermediate	Complex
Dinning philosopher	Bank system	Elevator

TABLE II
Examples of specification documents collected from students along with their classification

Simple	Intermediate	Complex
Hotel reservation	School system	Book store
Therapy center	Construction system	Supermarket

Then, we compared the class diagrams generated from the originally given specifications and the reversed synthesized one. The class diagram generated for the synthesized specifications in Fig.7 was the same of that in Fig.6.
We further tested our model using the second set of specifications documents in our test bed which we collected previously from Computer Information Systems (CIS) students in a Software Engineering (SE) course at the University of Jordan. Knowing that, those students are not English native speakers and they lack the experience in writing specification documents. The collected documents are characterized by their lack of standardization and their high ambiguity Table III shows sample of the students specifications documents. Documents in this set were tested first without any preprocessing, and then they have been processed. The results of the two were compared. The preprocessed versions of the given specification documents were automatically generated by our model. The modification was added to class diagrams and our model was used to reverse the updated class diagrams into natural language requirement.

5 Results and Explanations

After running the experiments, by analyzing the output and comparing our results with previous work results, we concluded the following:

- In terms of accuracy our model achieved higher percentage for identifying objects as well as attributes than previous researches. Our model was capable of achieving this level of accuracy because it analyzed every token in the specification document. This is rationalized by the fact that every word listed in the specifications document should carry significant information related to the problem description or it should not be present in the document. Therefore, every token should be analyzed and processed.
- In terms of natural language generation capability, our model was capable of generating a natural language view from the generated class diagram view. Whereas, this feature is not available in any of the previously conducted researches.
- In terms of gap analyses there was no significant difference between the class diagrams generated in the forward direction from the ones generated in the backward direction. This enables the requirements engineer to use our model to maintain requirements automatically.
- In terms of language quality, our results showed that preprocessing requirements in the sense that poorly structured requirements are converted to highly structured ones resulted in more accurate and complete output from the model.

Our model identifies relations if they were expressed using a constrained language set. However, none of the inputted specifications had that feature and thus our model was not able to identify relations. Table III summarize the results of testing the scientific test bed along with a comparison between our results and previous researches works. However, Table IV summarizes results for a sample of the students test bed.

6 Conclusions and Future Works

The advantages of automating requirements engineering are emphasized in our work. The results we obtained after testing our model highlighted the problems that face software engineers in processing natural language specifications documents. The results express the power of our model to tackle these problems.

TABLE III
Comparing results of academic set

Experiment name	Comparison criteria		Our model	Previous work
Dinning Philosopher Spec. Document	Modeling			
		No. of objects identified	3	2
		No. of attributes identified	4	2
		No. of relations identified	0	1
	Overall modeling accuracy			
		In term of objects	More	Less
		In term of attributes	More	Less
		In term of relations	Less	More
Bank System Spec. Document	Modeling			
		No. of objects identified	8	3
		No. of attributes identified	29	11
		No. of relations identified	0	0
	Overall modeling accuracy			
		In term of objects	More	Less
		In term of attributes	More	Less
		In term of relations	Same	Same
Elevator Spec. Document	Modeling			
		No. of objects identified	1	1
		No. of attributes identified	12	4
		No. of relations identified	0	0
	Overall modeling accuracy			
		In term of objects	Same	Same
		In term of attributes	More	Less
		In term of relations	Same	Same

Our works also highlighted how our automated model reduces the time, the efforts as well as the experience needed to model and maintain those documents.

In fact, our model can also serve as a tool for maintaining specifications documents where updates on specifications can be propagated to class diagrams in the forward direction and updates to class diagram can be propagated to natural language in the backward direction.

Majority of the future improvements anticipated in our work are in favor of an overall enhancement from a model into a fully functional system that uses semantics and domain knowledge in order to

analyze specifications documents. We are planning to build our own parser for natural language text and to expand our model to work with other languages. In addition we will automate the manual process of classifying specifications documents.

TABLE IV
Comparing results of a sample of student set, with and without preprocessing

Experiment Name	Without preprocessing	With preprocessing
Hotel reservation	Requirements of good quality all objects and attributes identified.	Requirements of good quality all objects and attributes identified.
Therapy center	Requirements of good quality all objects and attributes identified.	Requirements of good quality all objects and attributes identified.
School system	Requirements of poor quality not all objects and attributes were identified.	Requirements of good quality all objects and attributes identified.
Construction system	Requirements of poor quality not all objects and attributes were identified.	Requirements of good quality all objects and attributes identified..
Book store	Requirements of poor quality not all objects and attributes were identified.	Requirements of good quality all objects and attributes identified.
Supermarket system	Requirements of poor quality not all objects and attributes were identified.	Requirements of good quality all objects and attributes identified.

References:

[1] V. Ambriola, V. Gervasi, Environmental Support for Requirements Writing and Analysis. Information Science and Technology Institute, Technical Report, Pisa, Italy, 1999.

[2] A. AMESCUA, J. GARCÍA, M. SÁNCHEZ-SEGURA, and F. MEDINA-DOMÍNGUEZ," Software Process Improvement for Practitioners Based on Knowledge Management Tools", Proceedings of the 5th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems, Madrid, Spain, 2006, pp 24-29.

[3] J. Barnett, K. Knight, I. Mani, and E. Rich, "Knowledge and natural language processing", Communications of the ACM, vol. 33, no. 8, pp. 50-71, 1990.

[4] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, C. Ursu, M. Dimitrov, M. Dowman, N. Aswani, and I. Roberts, Developing Language Processing Components with GATE, University of Sheffield, 2005.

[5] D. Duffy, C. Macnish, J. Mcdermid, and P. Morris, A Model for Requirements Analysis Using Automated Reasoning. Proceedings of the 7th International Conference on Advanced Information Systems Engineering, Lecture Notes In Computer Science, vol. 932, 68-81. 1995.

[6] B. S. Lee, Automated conversion from a requirements document to an executable formal specification. Automated Software Engineering, 2001. Proceedings. 16th Annual International Conference, 437, 2001.

[7] B. Lee, and B. Bryant, "Applying XML technology for implementation of natural language specifications", International Journal of Computer Systems Science & Engineering, vol. 18, no. 5, 2003, pp. 279-300.

[8] Meta Integration Technology, Inc., (2006). Reference Guide. Available: http://www.metaintegration.net/Products/MIMB

[9] H. G. Perez-Gonzalez. Automatically Generating Object Models from Natural language Analysis. Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 86-87, 2002.

[10] M. A. Hennell. Requirements, Specification and Testing. Software Reliability Achievement and Assessment, Edited by B. Littlewood. *Blackwell Scientific Publication.* 1987.

[11] M. Saeki, H. Horai, and H. Enomoto, "Software development process from natural language specification", Proceedings of the 11th international conference on Software engineering, pp. 64-73,1989.

[12] I. Sommerville, Software Engineering, 8th edition, Addison Wesley, 2006.

[13] K. E. Wiegers, Software Requirements, 2nd edition, Microsoft Press, 2003.

[14] W. M. Wilson, L. H. Rosenberg and L. E. Hyatt, "Automated analysis requirement specification", Proceedings of the 19th international conference on Software engineering, International Conference on Software Engineering, pp. 161-171,1997.