EXTRACTING SOFTWARE REQUIREMENTS SPECIFICATION: A NATURAL LANGUAGE APPROACH

By

Yara Alkhader

Supervisor

Dr. Amjad Hudaib

Co-supervisor

Dr. Bassam Hammo

This Thesis Was Submitted In Partial Fulfillment For The Requirements For The Master's Degree Of Science In Computer Science.

Faculty of graduate studies

The University of Jordan

August, 2006

DEDICATION

To my parents who have always been my inspiration, their trust, believing in me and support were my guidance to finish this work.

ACKNOWLEDGMENT

I would like to express my deep gratitude to Dr. Amjad Hudaib and Dr. Bassam Hammo for their support, patience and guidance to finish this work.

I would also like to thank ESKADENIA Software Solution and especially Mrs. Doha Salah for giving me the opportunity to pursue my higher education.

LIST OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Description
BMDATC	Ballistic Missile Defense Advance Technology Center
DID	Data Item Description
GSFC	Goddard Space Flight Center
IEEE	Institute of Electrical and Electronics Engineers
KBNL	Knowledge-Based Natural Language
MIMB	Meta Integration Model Bridge
NASA	National Aeronautics and Space Administration
NLP	Natural Language Processing
SATC	Software Assurance Technology Center
TLG	Two Level Grammar
TRAM	A Tool for Requirements and Architecture Management
UML	Unified Modeling Language
VDM	Vienna Development Model
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

LIST OF APPENDICES

EXTRACTING SOFTWARE REQUIREMENTS SPECIFICATION: A NATURAL LANGUAGE APPROACH

By

Yara Alkhader

Supervisor

Dr. Amjad Hudaib

Co-supervisor

Dr. Bassam Hammo

ABSTRACT

This thesis expresses the possibility of automating the conversion of requirements expressed in natural language to a class diagram, and to reverse the conversion from a class diagram to natural language with a minimum data loss.

The driving theory behind this study is that semantical aspects are independent of the language presenting it. This means that one can describe the same semantics using different languages.

In this thesis, we suggested and implemented a framework, which is capable of processing requirements expressed in English natural language and generating an XML representation for it. The target XML representation is mapped into a model using a rule based functional analyzer whereas this model can be mapped back to XML. The XML representation serves as our framework repository.

The needs for such automation can be summarized as follows:

First, we can benefit from specification reusability were the requirements are kept to date with the system at hand.

Second, time efficiency when time is always a constraint while building a project.

Third, experience unavailability when a competent requirement engineer is not available to design and model the requirements.

We tested our framework thoroughly using detailed test cases to illustrate our framework capabilities.

The test bed collected for the experiments was gathered from different domains some of which are scientific while others are collected from students taking an undergraduate Software Engineering course at the University of Jordan. The collected specification documents cover distinct subjects and vary in their English language quality and complexity.

The results of the experiments indicated that the suggested framework can be used for modeling requirements and maintaining specifications documents. In addition, it can be used in requirements elicitation as it showed a high percentage in identifying objects and attributes.

The framework is characterized of being extensible which will enable the addition of continuous enhancements to promote it to a full functional system.

1. INTRODUCTION

Building a software is a continuous process. The ultimate goal of producing a software system is to realize the customers' needs. In this sense, development of the software is a progressive refinement from one abstraction level to another; where customer requirements are the most abstract level, and the software itself is the most concrete.

Requirements' engineering is inherently a continuous process. Its output specification document is subjected to continuous evolution reflecting the changes in the world it describes.

Based on these ideas, we began to investigate the possibility of automating the process of modeling the requirements and the process of updating it.

1.1 Problem Overview

Natural language requirements suffer from being ambiguous, inconsistent and incomplete. Modeling them requires a considerable level of experience; which might not be available all the time.

Maintaining and evolving the specification document is expensive in the sense that updates can be conducted at varying levels of abstraction in the software process, and all updates must be propagated and traced forward to the software and backward to the specification.

Propagating updates is very important to the software process in the sense that it ensures consistency between specification and other levels of abstraction revealing a reusable maintainable specification document consistent with the software it describes.

1.2 The Significance of the Study

The importance of this study comes from the importance of the specifications document to the software production. Engineers working on software may lack certain experiences, or the budget of the project may not cover the expenses of hiring highly experienced staff. The collected requirements are subject to continuous change, where time constitutes a constant dilemma that compromises the importance of keeping requirements up to date with the changes, while finishing the system on time. All those facts make the process of modeling and maintaining the specification a very good candidate to be automated.

Our suggested framework uses a set of rules, which reflect the basic knowledge that a requirements engineer uses in order to manually model and maintain requirements. In that sense modeling can precede without the need for experts and time can be saved by lowering the time spent on carrying out the updates manually.

1.3 Research Objectives

The main objective of our work is to build a framework that takes natural language requirements as its input and is capable of:

1. Generating a class diagram for the requirements.
2. Reversing the generated class diagram back to natural language requirements.

The English language was our choice of the natural languages our framework can support. However support for other natural language will be implemented in future works.

Our main contributions in this work are:

1. The creation of a framework, which is capable of modeling requirements and reverses engineering the model into natural language.
2. Providing the requirements engineers with a framework, which enables them to elicitate objects related to the system at hand.
3. Optimize the time consumed for modeling requirements and maintaining them.

1.4 Thesis Organization

This thesis is organized as follows: chapter two gives a literature review. An overview of the software process and the specifications document is presented, along with some back ground knowledge on UML and XML. In addition, several works and researches related to this thesis are reviewed. Chapter three introduces the framework and elaborates on its architecture and design. Chapter four presents the experiments conducted using the framework including data acquisition, testing and results discussion. Finally, chapter five expresses the conclusions of the research along with future works and enhancements.

2. LITERATURE REVIEW

In this chapter, we review some of the previous work that is related to our research. The software process and the requirements engineering process are both overviewed; a definition of a good specification document is introduced. We also present some background on UML (Unified Modeling Language) and XML (eXtensible Markup Language).

2.1 Software Engineering and the Software Process

The notion of software engineering was introduced in the late 60's at a conference discussing what was then called the "software crisis". The crisis as explained in (Naur and Randell, 1968) was the result of the introduction of the third generation computer hardware, which made more complex software applicable. Thereby the need to combine both computer science and engineering methodology became necessary and resulted in the foundation of software engineering (Broy, 2006).

Thus, software engineering can be thought of as establishing and using sound engineering principles to generate software which is reliable cost effective and works on real machines (Leffingwell and Widrig, 2003).

IEEE software engineering definition (IEEE, 1998) suggested that software has a life cycle starting from its creation until its end. The software life cycle proceeds as a systematic set of activities. These activities and their direct and indirect results leading to the production of software are exactly what software process is all about (Sommerville, 2004) (Pressman R., 2005).

There are some variations to the set of activities that a software process can have. However the main activities such as: software specification, software design and implementation, software validation and software evolution; remain common to all model of the process (Sommerville, 2004) (Pressman R., 2005).

2.1.1 Software Specification

Software specification also known as requirements engineering is concerned with establishing the required service from the system as well as the constraints on the system operations and development (Sommerville, 2004). Several definitions have been proposed for software specification, but the most referenced one is the one proposed in (Zave, 1997) which highlights a couple of important points:

- A system is a realization of real-world goals, thus those precise goals represent the basis for analyzing and validating the system requirements.
- A system evolves overtime emphasizing the reality of a changing world and its effect on the system specification.
- The requirement engineering is an iterative process; it is often regarded as a front-end activity in the software process.

The core activities in the requirement engineering process are (Wiegers, 2003):

- *Eliciting requirements:*

  The main focus of eliciting requirements is to identify the purpose of software, any system has goals which are the services and needs required by its customers, To find out what are the system goals one must start by identifying system boundaries

and stakeholders (all interacting parties with the software). This will facilitate the recognition, understanding and classifying of requirements.

The core of elicitation is to establish commutation between requirement engineers and system stakeholders. Requirement elicitation is an art and the requirement engineer must have high communication skills such as filtered listening, the ability to describe and explain, and the ability to crab new abstract concepts and must have the interest in solving other people problems (Callele and Makaroff, 2006).

- *Modeling and analyzing requirements:*

  Modeling is the construction of abstract descriptions that are amenable to interpretation. Requirements engineers may understand the requirements but they may not be able to communicate them. Modeling solved such problems by communicating requirements using abstract notations annotated with natural language. The more precise those notations are the less annotation is needed.

  Different modeling approaches exist such as: structured, formal and object oriented modeling. Each of them offers different analysis and reasoning power. One must choose the most appropriate approach.

- *Communicating requirements:*

  Requirements are structured and modeled into written documents and diagrams.

  The output of the requirements specification plays an important role ensuring clear communication of requirements.

  Different standards exist for the requirements specification document where each of them provides guideline in structuring requirements. However, there are the

IEEE Software Requirement Specification (IEEE, 1998) and the NASA DID Requirements (NASA).

- *Agreeing requirements:*

  Agreeing requirements is achieved through validating them, where validation means that requirements conform to what the customer wants. Requirement validation is curial to system acceptance as they guarantee that the system to be developed is what the customer had in mind.

- *Evolving requirements:*

  Software requirement was first elicited from real-world goals, as real-world goals evolve so must its requirements. Those changes must be carried out to the requirements specification as well. Tracing, monitoring and managing requirements are key concept in requirements evolution.

2.1.2 Requirements Specification Document

The requirements specification is a tool for communication between stakeholders. For this communication to be successful, the requirements specification document must be characterized as follows (Wilson et al., 1997):

- Complete:

  Defines precisely all real world situations that will be encountered and how to respond to them.

- Consistent:

  No conflict between individual requirement statements specifying behavioral and constraint properties.

- Correct:

  Identify accurately and precisely the individual conditions and limitations of all situations.

- Modifiable:

  Requirements must be structured into related chunks, limiting the side effect for any update.

- Ranked:

  Different requirements have different priority and the document must be able to convey this prioritization.

- Traceable:

  Requirements must be uniquely identified, so it can be linked and traced forward into development and backward into elicitation.

- Unambiguous:

  Each requirement must have one interpretation -one meaning-.

- Valid and Verifiable:

  If all stakeholders can understand, analyze and are able to accepts and prove requirements then those requirements are valid, if any level of requirements abstraction is consistent with other levels of abstraction, then it is verifiable.

2.2 A Background on UML and XML

In this section a background on both UML (Unified Modeling Language) and XML (eXtensible Markup Language) is presented. UML and XML notations are used thoroughly in our research. A good resource to read more about UML and XML is (Jacobson et al., 1999) and (Hunter et al., 2004).

2.2.1 UML (Unified Modeling Language)

UML is visual notation used for modeling software (Jacobson et al., 1999). Modeling can be thought of as a simplified description of a system, whereas a description means that the model forms some kind of representation of the system; not the system it self. As a matter of fact the formed description is considered as a simplified representation of a system, this representation hides the complexity of the system by exposing some aspects of a system graphically while abstracting others. This simplified representation assists the software engineer in understanding and reasoning about the system; humans proved to be more efficient in understanding ideas expressed graphically.

Different UML diagrams that can be used for system analysis and design exist. Among those we are particularly interested in the class diagram model, which is an object oriented view of a particular system (Rambaugh et al., 1999).

The basic building block in the class diagram is the class. A class has a name and is composed of a set of attributes and a set of operations working on these attributes. Classes represent real world objects and their attributes represent their state while their operations transform the object from one state to another. Classes are extracted from a statement of system problem. They represent sets of objects and operations from system domain (Boggs and Boggs, 1999).

Classes are connected to other classes through relationships such as aggregation and inheritance. Inheritance is the relationship declaring that one class is a specialization of another class. Aggregation is the relationship when one class is a collection of other classes as its subparts.

UML is becoming the *de-facto* standard for modeling (Boggs and Boggs, 1999). It is supported by a vast number of modeling tools (Boggs and Boggs., 1999), (VTC). Those tools not only have a digitized graphical view of a model but they also offer tremendous advantages of which most importantly they understand the model helping the designer making designs.

On the other hand, one of the major disadvantages for these tools is that the data in the model is incorporated with graphical representation data such as colors and position on screen. Those additions to the model disable the interoperability of the model between different tools; the model is only understandable in the tool which generated it (Laird, 2001).

An example of a UML model representing a car object is shown in Figure 1. This model is generated by the IBM Rational Rose © tool. The class is divided into three parts, where the first part holds the name of the object; the second holds the attributes and the last holds the operations. In Figure 1 the name of the object is Car, while the attributes are namely: Doors and Tires, the object presented in the figure has no operations thus its operation part is empty.

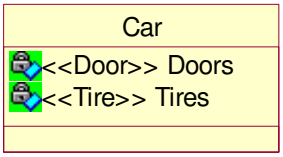

Figure 1: A class diagram for the car object generated using IBM Rational Rose

2.2.2 XML (eXtensible Markup Language)

XML is a mark up language. It serves as metalanguage, which means a language that describes another language. The main concern of XML is the presentation of structured data (Hunter et al., 2004).

XML is a mechanism for storing structural data in text files. As a matter of fact it becomes the mainstream for data warehousing and exchange. The power of XML lies in its interoperability and extensibility.

An XML file presents data in a hierarchical tree format. The first line in an XML document is an XML declaration. It is followed by a set of XML nodes.

An XML schema declares the rules to which an XML file must obey (Hunter et al., 2004). An XML schema as its name suggests is written using XML, it permits complex validation for the XML file. An example of an XML file is given in Figure 2. The file contains an XML representation of the car structure presented in Figure 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<car xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=" s.xsd">
    <doors/>
    <tires/>
</car>
```

Figure 2: XML representations describing the car object of Figure 1

The car XML file conforms to the car XML schema shown in Figure 3. Any variation of the car XML to its XML schema is considered as unacceptable.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="car">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="doors"/>
                <xs:element ref="tires"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="doors">
        <xs:complexType/>
    </xs:element>
    <xs:element name="tires">
        <xs:complexType/>
    </xs:element>
</xs:schema>
```

Figure 3: XML schema describing the car XML of Figure 2

2.2.3 Combining UML with XML

UML and XML are both standards. Each one is useful in a different way. UML is used for modeling, while XML is used for data exchange. The question arises here is there any use of combining UML with XML?

The answer to this question would be yes. As we have mentioned before that UML models generated by UML modeling tools lack the trait of interoperability while interoperability is a key factor in large scale software. (Laird, 2001).

The UML language is modeled using the UML metamodel, which is a UML model describing the UML language. This metamodel as all UML models have multiple views to which the XML schema is considered as its precise view. Thus this XML schema describing the UML metamodel serves as a model that describes the rules to which an XML structure must obey in order to be valid.

In this sense the more detailed XML structure conforming to the XML schema describing the UML metamodel can be transformed into the less detailed UML. There exist a number of tools capable of carrying out this transformation automatically (Rambaugh et al., 1999) (Meta, 2006).

Figure 4 presents a simplified metamodel describing the relation between classes and attributes in UML models. While Figure 5 presents the XML schema describing the model in Figure 1. Local elements in the schema depict attributes while global elements depict classes (Marchal, 2004).

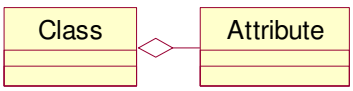

Figure 4: A simplified representation for the UML metamodel describing the class-attribute relation

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:complexType name="Attribute">
      </xs:complexType>
      <xs:complexType name="Class">
            <xs:sequence>
                  <xs:element ref="Attribute" minOccurs="2147483647" maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
      <xs:element name="Attribute" type="Attribute">
            <xs:annotation>
                  <xs:documentation/>
            </xs:annotation>
      </xs:element>
      <xs:element name="Class" type="Class">
            <xs:annotation>
                  <xs:documentation/>
            </xs:annotation>
      </xs:element>
</xs:schema>
```

Figure 5: XML schema describing the car XML of Figure 4

2.3 NLP (Natural Language Processing)

Natural languages are tools of communication. Studying natural languages, one can realizes that linguistics structures are related to world structures for example objects in world structure are represented as nouns in linguistics structures. The goal of natural language processing is producing computational representations based on the relationships existing between linguistic structures and a computation model representing the world structure such as the object oriented class model.

2.4 Related Works

The abstract nature of software and the vast variety of problems that admit to it was the driving force for repeatedly recognizing requirement engineering importance over the past decades 25 years (Mannion and Keepence, 1995).

Quality plays a significant role in software creation, where a high quality requirement specification results in creating a high quality software system. Consistency, completeness

and ambiguity are all essential characteristics of a high quality requirement (Bell and Thayer, 1976).

U.S Army Ballistic Missile Defense Advance Technology Center (BMDATC) conducted a study targeting the identification and resolution of fundamental problems plaguing software community. The study introduced a technique for verifying system specifications before initiating software design; emphasizing the importance of software requirement specification quality, where quality measures to what extend the software apply to its customer needs (Belford et al., 1976).

Over 8000 projects in 350 US companies were surveyed to reveal that one half of them suffer cost overrun, software significant delays and incomplete functionalities (Standish, 1995).

Managing requirements and their tractability to the software are key factors in system development and evolution. TRAM a tool for managing software requirements and system architecture and the tractability between them was introduced in (Han, 2001). TRAM primary objective is being practical with no overhead.

The cost of late correction to requirements can be 200 times more than those during requirements engineering. Accordingly the earlier they are identified the easier they are fixed (Boehm, 1981).

The Goddard Space Flight Center (GSFC) Software Assurance Technology Center (SATC) developed a tool that assesses requirements specified in Natural Language using quality indicators. The tool generates a report indicating requirements to be improved (Wilson et al., 1997).

Formal languages, semiformal language and informal languages have all been used in software requirement specification document, the latter being the most wildly used (Lee, 2003). However; hybrid representations also exist; an integrating of both formal and informal representation was suggested in (Duffy et al., 1995) where formal requirements are annotated with natural language comments.

Natural languages can be ambiguous, inconsistent and incorrect (Wilson et al., 1997), none the less it remains the most natural thus preferable way of communication requirements between both software engineers and software clients alike (Ambriola and Gervasi, 2000).

Automating the conversions between informal natural languages and formal ones became evident. Not only had it allowed the software requirements to be viewed in a user friendly way but also the conversion allowed the requirements to be viewed in a developer friendly way as well. In (Lee, 2003) natural language was transformed into the formal VDM++ language using TLG to break the gab between formal and informal languages. Their work targeted the elimination of the inherent natural language problems and automated the management of formal requirements keeping them compatible with their natural language counterpart.

In (Presland and Hennell, 1986) they investigated the ability to determine software functionalities from software requirements specifications expressed in natural languages. They illustrated the deficiencies and pointed the difficulties in processing natural languages. The objective of the study was to develop criteria for identifying functions. In their work they illustrated that the use of a simple method of determining functions is not productive.

In (Barnett et al., 1990) a Knowledge-Based Natural Language System (KBNL) was introduced. It presumes the existence of a model that describes the world and how language relates to the world. The system parses the English expression analysis them and then if converts them into the knowledge base representation.

Requirements engineering supporting environment was developed in (Ambriola and Gervasi, 2000). The interactive environment given requirements written in natural languages is used to analyze and synthesize different views using one shared repository and multiple modeling and viewing components.

Using XML the tangibility of natural language is improved for the automation of natural language translation into formal language (Lee and Bryant, 2003).

The language 4W is a constraint natural language proposed by (Perez-Gonzalez and Kalita, 2002). In their work they automated the transformation of natural language into the semiformal UML using role poset technique, which is a conceptual framework used to produce object oriented static views. They translated natural language requirement into 4W language and then used the generated set of requirements as input to their automation process.

Software development process from natural language specification was a process developed in 1989 (Saeki et al., 1989). In their work natural language was used to derive incrementally a formal specification through a design-elaborate cycle. The system model is extracted in the design phase from the informal English. Each word such as noun and verb in natural language sentence is associated with a software concept. Elaboration is the refinement of the natural language description based on the derived model in the design

phase. It is through the iterative cycling between the design and elaborate phases, the formal description starts to take form.

3. FRAMEWORK OVERVIEW

This chapter introduces the concept behind building our framework. It also gives an overview of its design and the high level architecture.

3.1 The Framework Data View

The inputted data to our framework is natural language requirements expressed in the English language. While the output of our framework is a class diagram model representing those requirements. The transformation from natural language into semiformal UML notations was possible due to the fact that: requirements weather expressed in formal, semiformal or informal languages assume certain linguistic aspects such as lexical, syntactical, semantical and pragmatical aspects.

Linguistic form of requirements

The following lists the linguistic aspects of the requirements expressed in natural language vs. semiformal language:

- ***Lexical aspects***

  Lexical aspects are applicable at the tokens level in the informal natural language, and labels level on the semiformal class diagram model. Each relevant unit (token, label) is of equivalent meaning in both presentations

- ***Syntactical aspects***

  Syntactical aspects present the grammar by which one can group lexical units into well formed sentences or models.

- ***Semantical aspects***

  Semantical aspects present the assumed meaning of a sentence or a model where the underlying semantic is independent of the language presenting it.

- ***Pragmatical aspects***

  Substantial amount of information can be carried out between the lines. The choices of words or the particular layout of a diagram can infer different meanings in different contexts.

3.2 The Framework Architecture View

The main objectives we are targeting from the suggested framework are: to be able to generate class diagrams out of natural language requirements and to generate natural language requirements out of class diagrams. In this sense our framework should accepts natural language requirements as input, it should be able to store it and produce class diagrams for it. In this sense the set of requirements our framework must satisfy became clear. We summarize these requirements as follows:

- **The framework must be able to accept requirements documents expressed in natural language provided that the documents are both well formed and well structured**.

  The presented framework accepts requirements documents written in English, other languages can be adopted in the future. The documents must be structured into paragraphs were a paragraph contains related data. Non related data are separated into separate paragraphs. The documents should be lexically and syntactically correct in terms of grammar, spelling and punctuations.

- **The framework must be able to store and retrieve requirements.**

  The presented framework accepts documents in text format and transform them into XML format. The output of the transformation is stored in a repository on the filing system and from this repository the requirements are synthesized.

- **The framework must be able to present class diagrams for the stored requirements.**

  The presented framework should be able to provide a class diagram view for the requirements document. The generation of the class diagram proceeds iteratively from the XML requirements version.

- **The framework must be able to maintain the contextual meaning and lexical content as far as possible.**

  The presented framework must be able to transform the requirements forward into XML form and from XML form backward to natural language with the minimum changes possible in reference to the original requirements document. This is achievable by saving all the processing output as annotation to the XML file. In this way the lexical content of the requirements as well as its contextual meaning is intact so it is preserved.

- **The framework must be extensible.**

  The framework presented must allow one to extend it by adding, updating or replacing components.

Framework architecture

The main components of our framework are shown in Figure 6. In addition, we have used some tools such as The GATE (Cunningham et al., 2005) and (Meta Integration Model Bridge) MIMB (Meta, 2006). We will explain the tools and how we used them is section 3.3.

The framework interacts with three repositories:

- The natural language requirements repository: this is used to store the requirements before processing them in the system as well as after preprocessing and it also stores the requirements generated from the reverse engineering process.
- The XML requirements repository: this is used to store the intermediate processing outputted from the framework core components.
- The UML class diagram repository: this is used to store the class diagrams generated from the MIMB tool.

The preprocessor and the core functional components will be illustrated in details in section 3.3.

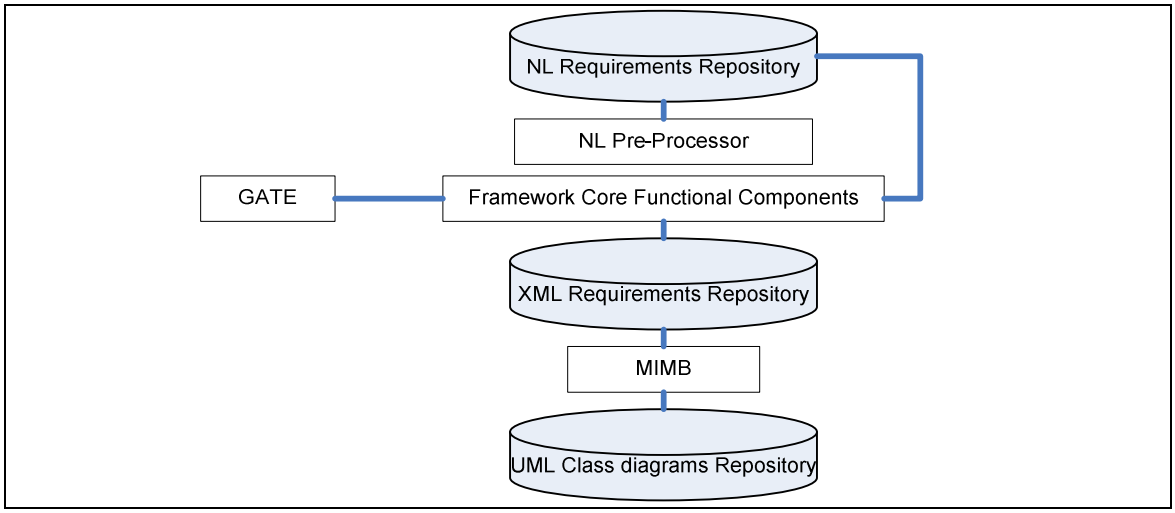

Figure 6: The Framework high level architecture

3.3 The Framework Components View

In Figure 7, we express the data processing components and how data is converted in our framework. We illustrate the process using a subset of a bank requirements document borrowed from (Lee, 2003).

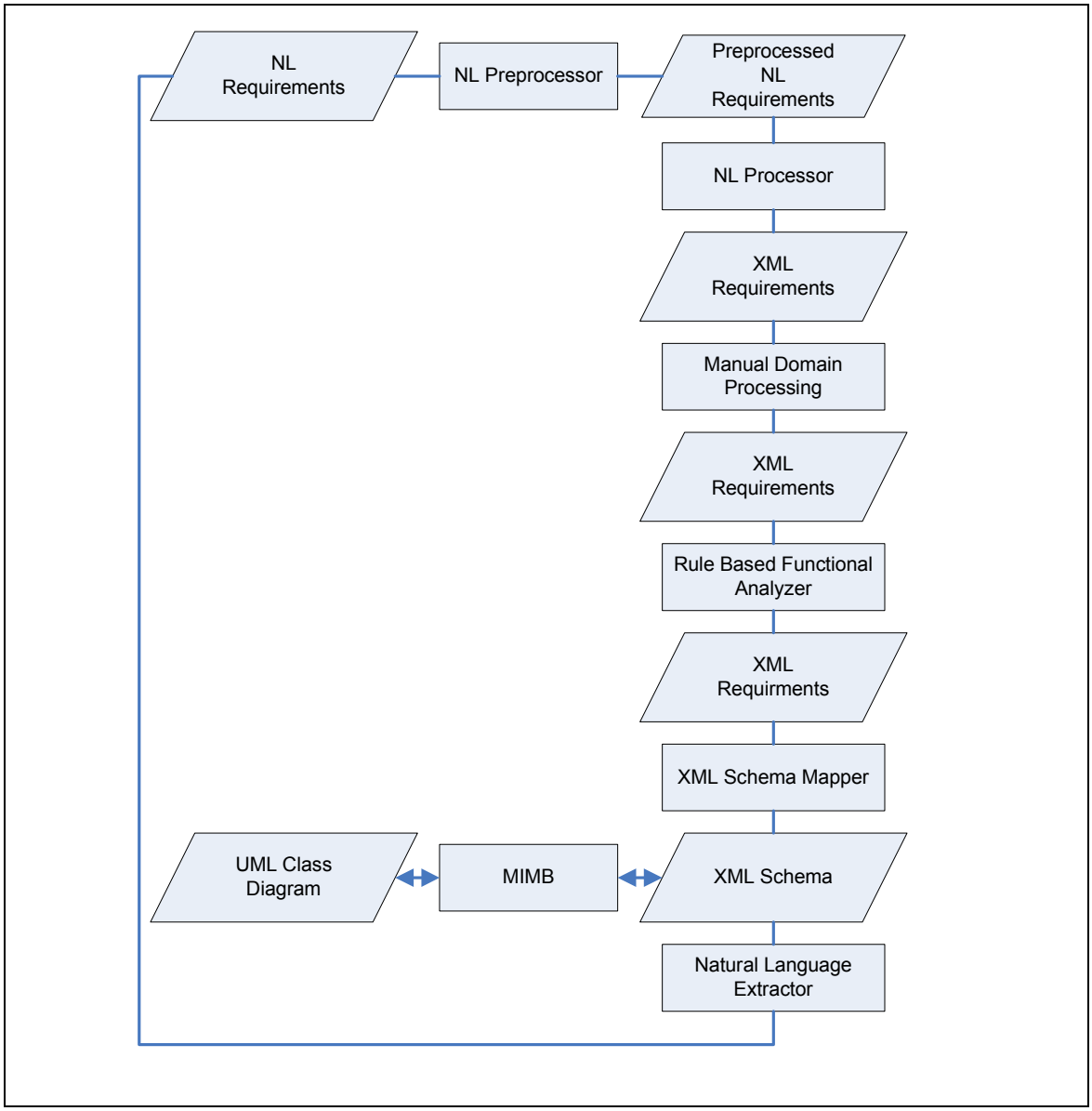

Figure 7: Flow diagram showing the processes and data conversions in the framework

Our framework has the following data processing components:

a. Preprocessing component

This is an optional stage in the framework. Its main objective is to produce a natural language document which is well structured and well formed.

Well formed documents are ones that have no spelling errors and are grammatically sound. While well structured means that documents are well paragraphed and each paragraph contains only related information.

Preprocessing refines natural language requirements to improve their quality and to make them ready for automated processing. Accordingly the better quality the requirements are the less preprocessing is needed.

Basic functionalities

A document is first checked by a spell checker and then by a syntax checker. In our work we chose to use Microsoft word for spell and syntax checking. Misspelled words, wrong syntaxes are highlighted for manual correction taken care of by the software engineer. Our choice of limiting the preprocessing to lexical and syntactical checks can be justified by the nature of specification documents, which assumes being revised and well written.

Figure 8 shows the subset of the bank requirements, which will be used to explain our framework functionalities (Lee, 2003).

Bank verifies ID and PIN giving the balance in the following order. It selects the account from the list of accounts where: the account ID equals the ID and the account PIN equals the PIN.

Figure 8: Bank requirements checked by Microsoft word

b. Natural language processing component

"Natural language processing can be defined as a process to construct a formal structure and meaning of sentence in a way that helps the computer to understand the sentence" (Lee, 2003).

The main role of natural language processing is to parse each sentence to find the part of speech for every token in the sentence and the role of each part in the sentence.

Natural language processing undergoes a set of activities: natural language requirements are first tokenized to determine the part of speech for each word. The output of this process is then parsed to determine the role of each token in the sentence.

Basic functionalities

In this component we use the GATE tool (Cunningham et al., 2005) along with the Minipar (Cunningham et al., 2005) plug in to construct our natural language processing model.

GATE is an infrastructure for developing and deploying software components that process human language. While Minipar is a shallow parser developed as a plugin component to GATE framework. Our choice of the GATE infrastructure is based on the following facts:

- It is free and open source.
- It has been used in a wide number of researches.
- Its generated output is in the form of XML.
- It is written entirely in java and conforms to the java specifications.

Our natural language processing model is an iterative one. The output of each iteration is an XML representation of the natural language requirements. Each word is annotated with its part of speech and its part of sentence respectively

The XML file represents our internal repository for natural language requirements. This representation allows us to:

- **Maintain the natural language with as less modification as possible**

  This is actually possible as the result of processing is added to the natural language tokens as annotations so those words stay intact.

- **It is extensible where new annotation can be easily added**

  Adding or updating components to the framework can be easily integrated as new annotation can be added to the generated XML, which is extensible by nature.

- **It enables the storage and retrieval of natural language requirements**

  Storing and retrieving can be thought of as annotating each token and removing the annotation respectively.

Domain knowledge must be provided as input to this phase. It improves pragmatic analysis dramatically.

Domain knowledge is implemented in our framework as a manual process where software engineers gets to remove redundant entities, resolve constructs with similar meaning and group them together. This part can be automated in

future works but the samples applicable to our current framework are of considerable size so automation was of no significant.

Figure 9 expresses the XML output of the natural language processor of the requirements in Figure 8.

c. Rule based functional analyzer component

Rule based functional analyzer is a set of pluggable processes those processes form the basis for our work. Each of the processes applies a specific rule; that either helps or identify a functional specification.

The analyzer updates and modifies annotations and store the result into the XML repository.

```
<paragraph>
    <Token category="NN" POS="s">Bank</Token>
    <Token category="NNS" POS="v">verifies</Token>
    <Token category="NNP" POS="obj">ID</Token>
    <Token category="CC">and</Token>
    <Token category="NNP">PIN</Token>
    <Token category="VBG">giving</Token>
    <Token category="DT">the</Token>
    <Token category="NN" POS="obj">balance</Token>
    <Token category="IN">in</Token>
    <Token category="DT">the</Token>
    <Token category="VBG">following</Token>
    <Token category="NN">order</Token>
    <Token category=".">.</Token>
    <Token category="PRP" POS="s">It</Token>
    <Token category="VBZ" POS="v">selects</Token>
    <Token category="DT">the</Token>
    <Token category="NN" POS="obj">account</Token>
    <Token category="IN">from</Token>
    <Token category="DT">the</Token>
    <Token category="NN">list</Token>
    <Token category="IN">of</Token>
    <Token category="NNS">accounts</Token>
    <Token category="WRB">where</Token>
    <Token category=":">:</Token>
    <Token category="DT">the</Token>
    <Token category="NN">acount</Token>
    <Token category="NNP" POS="s">ID</Token>
    <Token category="VBZ" POS="v">equals</Token>
    <Token category="DT">the</Token>
    <Token category="NNP" POS="obj">ID</Token>
    <Token category="CC">and</Token>
    <Token category="DT">the</Token>
    <Token category="NN">acount</Token>
    <Token category="NNP" POS="s">PIN</Token>
    <Token category="VBZ" POS="v">equals</Token>
    <Token category="DT">the</Token>
    <Token category="NNP" POS="obj">PIN</Token>
    <Token category=".">.</Token>
</paragraph>
```

Figure 9: Bank XML requirements output of NLP

Basic functionalities

Each analyzer process checks the requirements against a specific rule. Our choice of the set of applicable rules was based on the most common problems specific to specification documents as well as the relations our processing model assumes which are namely: objects, attributes and actions; each rule in the set helps in resolving a problem. other information existent in the natural language requirements and not applicable to our processing model is discarded (Lee, 2003) (Presland and Hennell, 1986). Thus the set of rules applied are the following:

- **Resolving compound nouns**

  A noun followed by another noun (ignoring determiners) is considered as one compound noun, where the compound noun is annotated with "NN" as its part of speech. The part of sentence annotation reflects the first part of sentence attribute found.

The XML output of compound name resolution is depicted in Figure 10.

```
<Token category="DT">the</Token>
<Token category="NN" POS="s"> account ID</Token>
<Token category="VBZ" POS="v">equals</Token>
```

Figure 10: Snapshots after compound nouns resolution

- **Resolving collections of object**

  A singular noun followed by a proposition and a plural noun, or a singular noun followed by a preposition which is followed by another singular noun then another preposition and a plural noun, can be considered as collections. For example: account in the list of accounts.

Annotation proceeds as the preceding rule suggested taking into consideration that determiners are also ignored.

The XML output of collection resolution is depicted in Figure 11.

```
<Token category="DT">the</Token>
<Token category="NN" POS="obj"> the  account from the list of
 accounts</Token>
```

Figure 11: Snapshots after collections resolution

- **Resolving pronouns**

  Pronouns can refer to either the preceding subject or the recently referenced subject with the later having a higher priority than the former. The pronoun is annotated with a tag indicating its reference.

The XML output of pronoun resolution is presented in Figure 12.

```
<Token category="PRP" POS="s" ref="Bank">it</Token>
<Token category="VBZ" POS="v">assigns</Token>
```

Figure 12: Snapshots after pronoun resolution

- **Resolving connectors**

  If a noun is followed by a connector and then another noun it is considered as a sentence connecter provided there is a verb coming some where after the second noun and before the end of sentence. Otherwise it is considered a noun connector. In case of sentence connectors we add a tag indicating that. Else the two nouns are considered as one compound noun.

The XML output of connector resolution is depicted in Figure 13.

```
<Token category="NN" POS="s">Bank</Token>
<Token category="NNS" POS="v">verifies</Token>
<Token category="NNP" POS="obj">ID</Token>
<Token category="CC" Conn="noun">and</Token>
<Token category="NNP" POS="obj">PIN</Token>
<Token category="VBG">giving</Token>
<Token category="DT">the</Token>
```

Figure 13: Snapshots after connector resolution

- **Resolving association and generalization**

  In resolving association and generalization we choose a subset of the English language to be identified as a relation.

  If an object is associated with another object then the framework expects this to be written in the natural language requirements precisely using the word (associated).

  In case of generalization the framework expects the words: is a generalization of, to be present in the natural language requirements to be able to identify the relation.

  In future work we plan to make the framework capable of identifying relations without restricting the natural language to a fixed subset of words.

d. XML Schema mapper

Our target of creating this process is to map XML requirements into an XML schema representing UML. The schema can be later transformed to UML, imported by a UML tool and displayed as a class diagram model.

The relevant set of nouns and verbs are first correlated as follows:

- Nouns occurring in same paragraph are considered relevant. This also applies to verbs. The justification of this is based on our first assumption of having a well written and well structured specification document.
- Verbs are excluded in this transformation as XML schema is capable of representing the data but not the operations. In future work we plan on

using XMI (XML Metadata Interchange) instead as it is capable of representing both data and operations.

- An ID is generated and added to both XML schema and XML file for the sake of traceability in order to eliminate data lose caused by the transformation from the more detailed XML to the less detailed UML.

Mapping is a bidirectional process; it can be forward from XML into UML or backward from UML into XML.

Basic functionalities

The forward mapping proceeds by generating an XML schema file where each mapped token is given an ID to be used for backward traceability. Then MIMB (Meta Integration Model Bridge) tool (Meta, 2006) is used for transforming XML schema into UML.

The backward mapping proceeds by transforming UML into XML schema. All previous transformations are done using the MIMB tool. The XML schema is then transformed into an XML representation for the natural language requirements.

e. Natural language extractor

The XML presenting natural language requirements is used to extract natural language sentences and paragraphs.

The generated language is a one to one mapping between the XML and language constructs; if no changes occurred on the UML model then the

generated document must present all the information available in the original natural language requirements using simpler sentences.

Basic functionalities

The XML file presenting storage for the requirements is transformed into natural language by eliminating all the XML annotations combining the result into a text file, using a set of simple rules describing how to build simple statements, for instance a singular name staring with a vowel is proceeded with (an), while a singular name staring with any other letter is proceeded with (a).

4. EXPERIMENTING WITH THE FRAMEWORK

This chapter introduces the methodology of testing the framework along with the detailed results and test cases. It also analyzes the results of the test cases.

4.1 The Test Bed

Our test bed of requirement documents is divided into two sets. The first set of documents was collected from different academic researches. Each document has been used to conduct a research similar to our work. The results of analyzing each document has been compared with the original obtained from its resource. Those documents although varying in subjects and in authors but all share some common features. They are well written using well structured English sentences, and their content is fairly unambiguous.

The second set of requirement documents was collected from Computer Science students in a Software Engineering course at the University of Jordan. Those students are both not native speakers of the English language and are not well experienced in writing requirements. Thus documents in set two are not well written; they lack correct punctuations, and they are highly ambiguous.

The two sets are further classified according to the requirements sentences into three categories: simple, intermediate and complex.

Simple documents are composed of simple sentences, whereas a simple sentence is a sentence that carries out one atomic fact. Those sentences usually have a subject, a verb and an object, and each ends with a fullstop.

Intermediate documents are composed of more complicated sentences rather than simple ones. An intermediate sentence is one that carries out more than one fact. It has one subject, multiple verbs and objects, it may contain commas and it ends with a fullstop.

Complex documents are those composed of complex sentences. A complex sentence is a sentence that carries out many facts. It could have many subjects, verbs and objects, it contains commas and it ends with a fullstop.

4.2 Conducting the Experiments

Documents in the test bed were executed in a complexity ascending manner; starting from the simplest and walking through the most complicated. Our first experiment starts with the test bed extracted from previous academic researches.

4.2.1 Test Bed from Previous Researches

This test bed contains three requirements documents:

- The dining philosophers' requirements document, conducted by (Perez-Gonzalez and Kalita, 2002).
- The bank requirements document, conducted by (Lee, 2003).
- The elevator requirements document, conducted by (Saeki et al., 1989).

According to our proposed classification, the above documents are classified as follows: the first one being the simplest while the others are the intermediate and complex respectively. Documents have been inputted to our system as they have been originally written without any modification.

4.2.1.1 The Dining Philosophers Requirements Document

Conducting the Experiment

The requirements document for the dining philosophers (Perez-Gonzalez and Kalita, 2002) depicted in Figure 14 describes the famous problem of the 5 philosophers and 5 forks where each philosopher needs two forks to eat.

> 5 philosophers and 5 forks around a circular table. Each philosopher can take 2 forks on either side of him. Each fork may be either on the table or used by one philosopher. A philosopher must take 2 forks to eat.

Figure 14: Dinning philosopher requirements document

The set of requirements in the requirements document are processed by our proposed framework. The requirements are first processed by our NL processor to generate an XML representation, and the XML representation is processed by our rule based functional analyzer and mapped later on to an XML schema using our XML schema mapper.

The XML schema depicted in Figure 15 represents the requirements of the dinning philosopher problem. This schema is generated by the system automatically. However redundant entities should be eliminated. The process of eliminating redundancy identifies words representing the same entities. For example, the words philosophers and philosopher as well as forks and fork represent the same entity. In this step we erase forks and philosophers from the schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="circular  table">
        <xs:annotation>
            <xs:documentation>6, 6</xs:documentation>
        </xs:annotation>
    </xs:complexType>
    <xs:element name="circular  table" type="circular  table">
        <xs:annotation>
            <xs:documentation>6, 6</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="fork">
```

```
        <xs:annotation>
            <xs:documentation>17, 17</xs:documentation>
        </xs:annotation>
        <xs:attribute name="table">
            <xs:annotation>
                <xs:documentation>22, 22</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="philosopher">
            <xs:annotation>
                <xs:documentation>27, 27</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:element name="fork" type="fork">
        <xs:annotation>
            <xs:documentation>17, 17</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="forks">
        <xs:annotation>
            <xs:documentation>4, 4</xs:documentation>
        </xs:annotation>
    </xs:complexType>
    <xs:element name="forks" type="forks">
        <xs:annotation>
            <xs:documentation>4, 4</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="philosopher">
        <xs:annotation>
            <xs:documentation>8, 8</xs:documentation>
        </xs:annotation>
        <xs:attribute name="forks">
            <xs:annotation>
                <xs:documentation>12, 12</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="side of">
            <xs:annotation>
                <xs:documentation>14, 14</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="forks">
            <xs:annotation>
                <xs:documentation>33, 33</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:element name="philosopher" type="philosopher">
        <xs:annotation>
            <xs:documentation>8, 8</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="philosophers">
        <xs:annotation>
            <xs:documentation>1, 1</xs:documentation>
        </xs:annotation>
    </xs:complexType>
    <xs:element name="philosophers" type="philosophers">
        <xs:annotation>
            <xs:documentation>1, 1</xs:documentation>
        </xs:annotation>
    </xs:element>
</xs:schema>
```

Figure 15: Dinning philosopher XML schema

The XML schema is then mapped using the MIMB (Meta, 2006) tool into the UML class diagram depicted in Figure 16. The three boxes in the diagram represent the fork object, the philosopher object and the table object. The fork object has two attributes, namely, *the fork is placed on a table* and is *used by a philosopher*. On the other hand the philosopher object has two attributes as well: *the philosopher has two sides* and *the philosopher uses a fork*. The table object has no identified attributes.

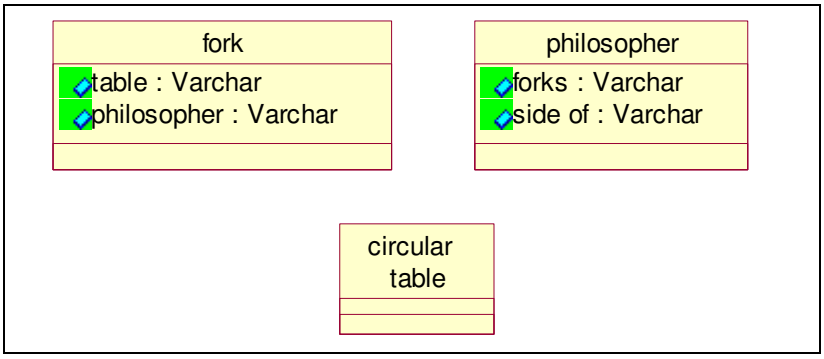

Figure 16: Dinning philosopher UML model

Our framework is capable of converting the UML class diagram depicted in Figure 14 back to natural language. This reverse engineering process produced the natural language requirements depicted in Figure 17.

Each fork has a table,
Each fork has a philosopher.
Each philosopher has forks,
Each philosopher has a side of,
Each philosopher has forks.

Figure 17: Dinning philosopher reversed natural language

Results Analysis and Comparison

Now we compare our results with (Perez-Gonzalez and Kalita, 2002). In their work the authors identified two namely objects fork and philosopher with a validation threshold of 50%. Other objects could be identified if a threshold of a lower percentage was selected.

Both the fork object and the philosopher object contained a location attribute and they were interconnected with the relation *take*; where a philosopher can take a fork. On the other hand our work was able to identify three objects: philosopher, fork and table. The philosopher object has two attributes *fork* and *side* and the fork object has two attributes *philosopher* and *table*. The object fork having a *philosopher* attribute and the philosopher object having a *fork* attribute resembles the association between the two objects, which was identified in the original paper as the *take* relation. While the *side* and *table* attributes resembles the attribute location which has been identified in the original paper as well. In our work we were able to transform the generated model into natural language, which is a one to one representation of the model and can be used for validating both user requirements and the represented model. Here one can think of the reversed language as a semiformal representation of the requirements where both the model and the language fall into the same abstraction level thus comparing them is straight forward. While in the original paper they used the inputted requirements for validating the model were both lie in different layers of abstraction and comparing them is highly dependent on the person doing the validation process. Accordingly validating them is relative, complex and error prone. Table 1 summarizes the result comparison between our work and the previous research work.

Table 1: The dining philosopher requirements document experiment comparison between our work and the previous research work.

Criteria	Our Results	Research Results
No. of Objects Identified	3	2
No. of Attributes Identified	4	2
No. of Relations Identified	0	1

4.2.1.2 The Bank System Requirements Document

Conducting the Experiment

The requirements document for the bank system (Lee, 2003) depicted in Figure 18 describes a banking system where users have accounts and use ATM services.

> Bank keeps list of accounts. It verifies ID and PIN giving the balance and updates the balance with ID.
> An account has three data fields: ID, PIN, and balance.
> ID and PIN are integers, balance is a real number.
> ATM has 3 service types: withdraw, deposit and balance check.
> For each service first it verifies ID and PIN from the bank giving the balance.
> ATM withdraws an amount with ID and PIN giving the balance in the following sequence. If the amount is less than or equal to the balance then it decreases the balance by the amount. And then it updates the balance in the bank with ID.
> ATM deposits an amount with ID and PIN giving the balance in the following order. It increases the balance by amount and then updates the balance in the bank with ID.
> ATM checks the balance with ID and PIN giving the balance.

Figure 18: Bank system requirements document

The XML schema generated after processing the requirements and then eliminating redundant objects is listed in appendix A Figure A1. The UML model mapped from generated schema in appendix A Figure A1 is listed in appendix A Figure A2. The natural language requirements reversed engineered from the module in appendix A Figure A2 is listed in appendix A Figure A3.

Results Analysis and Comparison

As discussed earlier the results of our work are compared with the results in (Lee, 2003).

In their work three objects were identified: bank, account and ATM. The bank object has the attributes: account list, ID, PIN and balance. The account object has the attributes: ID, PIN and balance. The ATM object has the attributes balance, amount, ID and PIN.

In our work we were able to identify the bank, ATM and account objects; the three of them can be considered a like in both works. In addition our work as able to identify the ID, the PIN and the balance as objects (data types). Identifying data types is of great value because it is essential in representing new types and in validating the represented model. Further more our work identifies the service object which can be considered as an abstraction where all services shares a common interface. However in our work the amount was wrongly identified as an object. For this example we also revered the model to natural language which was used to represent the model in a more unambiguous fashion were both the engineer and the client can share their understanding equally. Table 2 summarizes results comparison between our work and the previous conducted research.

Table 2: The bank system requirements document experiment comparison between our work and the previous research work.

Criteria	Our Results	Research Results
No. of Objects Identified	8	3
No. of Attributes Identified	29	11
No. of Relations Identified	0	0

4.2.1.3 The Lift System Requirements Document

Conducting the Experiment

The requirements document for an elevator system (Saeki et al., 1989) listed at Figure 19 describes an elevator system installed in a building.

An n lift system is to be installed in a building with m floors. The lift and the control mechanism are supplied by the manufacturer.
The internal mechanisms of these are assumed.
The problem concerns the logic to move lifts between floors according to the following constraints.
Each lift has a set of buttons, one for each floor.
These illuminate when pressed and cause the lift to visit the corresponding floor.
The illumination is cancelled when the corresponding floor is visited by the lift.
Each floor has two buttons, one to request an up-lift and one to request a down-lift.
These buttons illuminate when pressed.
The illumination is cancelled when a lift visits the floor and is either moving in the desired direction, or has no outstanding requests.
In the later case, if both floor buttons are pressed only one should be cancelled.
The algorithm to decide which to service should minimize the waiting time for both requests.
When a lift has no requests to service, it should remain at its final destination with its door closed and await further request.
All requests for lift from floors must be serviced eventually, with all floors given equal priority.
All requests for floors within lifts must be serviced eventually, with floors being serviced sequentially in the direction of travel.
Each lift has an emergency button which, when pressed causes a warning signal to be sent to the site manager.
The lift is then deemed out-of-service.
Each lift has a mechanism to cancel its out-of-service status.

Figure 19: Lift system requirements document

The XML schema generated after processing the requirements and then eliminating redundant objects is listed in appendix B Figure B1. The UML model mapped from generated schema in appendix B Figure B1 is listed in appendix B Figure B2. The natural language requirements reversed engineered from the module in appendix B Figure B2 is listed in appendix B Figure B3.

Results Analysis and Comparison

As with early tests the results of our work are compared with the results in (Saeki et al., 1989). In their work the attributes lift button, direction, status and request were identified while in our work the at tributes: manufacturer, buttons list, floor, priority, emergency button, warning signal, site manager, mechanism, status, request, destination, and door were identified.

We believe that although manufacturer, site manager and mechanism were listed at the requirements and correctly identified, but they are of no importance to the scope so they could be ignored. However the attributes priority, emergency button, warning signal, destination and door were not identified in the original paper while they are of importance to both the lift system and the completeness of the requirements. On the other hand in our work we incorrectly were not able to identify the direction attribute as attribute for the lift object.

The original paper introduce a template for writing requirements were an engineer after analyzing the requirements and modeling them should manually write then into the introduced template. While in our work we allow the engineer both automatic generations of the requirements after reversing the model into natural language. And the flexibility to express the requirements in any format while maintaining a simple structured language.

Table 3 summarizes results comparison between our work and the previous conducted research.

Table 3: The lift system requirements document experiment comparison between our work and the previous research work.

Criteria	Our Results	Research Results
No. of Objects Identified	1	1
No. of Attributes Identified	12	4
No. of Relations Identified	0	0

4.2.2 Test Bed from Students'

This test bed contains six requirements documents:

- The hotel reservation requirements document.
- The therapy center requirements document.
- The school system requirements document.
- The construction system requirements document.
- The book store requirements document.
- The supermarket requirements document.

The first and second are the simplest while the third and fourth are the intermediate and the other two are the complex ones.

Each of the documents in the set has been inputted into the system with and without preprocessing. The more unambiguous and complete the requirements are the more accurate the generated module is. Preprocessing stage aims at modifying those documents from ambiguous into unambiguous ones. Preprocessing was executed at the model level which is reversed into the natural language automatically. Then this document was inputted into the system and the module was generated one more time, only this time it represents the preprocessed document.

For this set we will only present the input and the module in the two cases with and without preprocessing, intermediate steps are not listed.

4.2.2.1 The Hotel Reservation System Requirements Document

Conducting the Experiment

Figure 20 depict the requirements collected describing the hotel reservation system. The requirements were processed into an XML schema and then mapped into the UML class diagram listed in appendix C Figure C1.

The hotel consists of a number of rooms. Every room has a number, rate types, notes, status, occupancy type, TV status, phone status, balcony status, and the number of room is unique. Every customer has an SSN, first name, last name, address, home phone number, mobile number, email, number of adults and children, can reserve one or more rooms, and the SSN of the customer is unique. Every reservation has a number, arrival date, departure date, reservation date, payment method, sales tax, charges, amount paid, total payable and the number of the reservation in unique.

Figure 20: Hotel reservation system requirements document without preprocessing

For this set of documents we conducted the experiment by first processing the requirements without modifying them. And then after generating the class-diagram for the requirements we added the modification at the model level thereby allowing our framework to generate the modified natural language requirements automatically instead of rewriting them.

The UML class diagram listed in appendix C Figure C1 represents the generated model for the requirements without any modification. The natural language reversed engineered from the modified UML class diagram is presented appendix C Figure C2 and is processed to the UML model presented in appendix C Figure C3.

A modification we added in our work to the class diagram was the addition of relations Those relations were translated into natural language requirements and thus could be mapped forward and back word into a model and natural language respectively. This

addition was due to the fact that those relations were not explicitly expressed in the natural language provided and ignoring them would make the module incomplete.

However after the addition of relationships the generated language from the model had those relations explicitly expressed thus those requirements became more clear and unambiguous.

Results Analysis and Comparison

The generated model from the requirements without considering preprocessing was of very good accuracy as all the objects and attributes were identified. Preprocessing the requirements was not of great value as no direct effect was noticed on the represented model. The fact that the original requirements was good written with simple sentences made the accuracy quite evident.

4.2.2.2 The Therapy Center Requirements Document

Conducting the Experiment

Figure 21 depicts the requirements collected describing a therapy center. The requirements were processed into an XML schema and then mapped into the UML class diagram listed in appendix D Figure D1. The reversed engineered natural language requirements generated after modifying the UML class diagram in appendix D Figure D1 is presented in appendix D Figure D2 and its corresponding UML class diagram is presented in appendix D Figure D3.

Make the patients reservations. Produce work reports. Add new patients to the system. Retrieve patients files. Arrange patients with therapists.

Figure 21: Therapy center requirements document without preprocessing

Results Analysis and Comparison

The generated model from the requirements without considering preprocessing was of good accuracy as most of the objects and attributes were identified. However preprocessing the requirements was of great value as the original sentences although simple in structure but they failed to represent the system correctly. Therefore accuracy of the model after preprocessing is quite evident.

4.2.2.3 The School System Requirements Document

Conducting the Experiment

Figure 22 depicts the requirements collected describing a school system. The requirements were processed into an XML schema and then mapped into the UML class diagram listed in appendix E Figure E1. The reversed engineered natural language requirements generated after modifying the UML class diagram in appendix E Figure E1 is presented in appendix E Figure E2 and its corresponding UML class diagram is presented in appendix E Figure E3.

School contains teachers, manages, students, workers.
All the mentioned stakeholders are allowed to use the program.
The main target is to help the above beneficiaries to get needed information, control the data and manage it depending on the level of each user.

Figure 22: School system requirements document without preprocessing

Results Analysis and Comparison

The generated model from the requirements without considering preprocessing was of bad accuracy as most of the objects and attributes were not identified. However; preprocessing the requirements was of great value, as the original sentences failed in representing the system correctly. Therefore accuracy of the model after preprocessing is quite evident.

4.2.2.4 The Construction System Requirements Document

Conducting the Experiment

Figure 23 depicts the requirements collected describing a construction system. The requirements were processed into an XML schema and then mapped into the UML class diagram listed in appendix F Figure F1. The reversed engineered natural language requirements generated after modifying the UML class diagram in appendix F Figure F1 is presented in appendix F Figure F2 and its corresponding UML class diagram is presented in appendix F Figure F3.

User requires a form to input the new employees, information. User requires a form to input the new projects information. User requires a form to input the sub-contractor information. User requires a form to input the machine information. User requires a form to input the payments information. User requires a form to input the items taken from the store. User requires a monthly report for the employee salary. User requires daily report for machines working hours. User requires a daily report for project working hours.

Figure 23: Construction system requirements document without preprocessing

Results Analysis and Comparison

The generated model from the requirements without considering preprocessing was of bad accuracy as most of the objects and attributes were not identified. However; preprocessing the requirements was of great value, as the original sentences failed in representing the system correctly. Therefore accuracy of the model after preprocessing is quite evident.

4.2.2.5 The Book Store System Requirements Document

Conducting the Experiment

Figure 24 depicts the requirements collected describing a book store system. The requirements were processed into an XML schema and then mapped into the UML class diagram listed in appendix G Figure G1. The reversed engineered natural language requirements generated after modifying the UML class diagram in appendix G Figure G1 is presented in appendix G Figure G2 and its corresponding UML class diagram is presented in appendix G Figure G3.

An employee can view the availability of any book to sell it.
And employee can register any customer and view his account at any time.
An employee can sell any book to any customer whether registered or not.
An employee can loan any book to customers who are registered.

Figure 24: Book store system requirements document without preprocessing

Results Analysis and Comparison

The generated model from the requirements without considering preprocessing was of bad accuracy as most of the objects and attributes were not identified. However; preprocessing the requirements was of great value, as the original sentences failed in representing the system correctly. Therefore accuracy of the model after preprocessing is quite evident.

4.2.2.6 The Supermarket System Requirements Document

Conducting the Experiment

Figure 25 depicts the requirements collected describing a supermarket system. The requirements were processed into an XML schema and then mapped into the UML class diagram listed in appendix H Figure H1. The reversed engineered natural language requirements generated after modifying the UML class diagram in appendix H Figure H1

is presented in appendix H Figure H2 and its corresponding UML class diagram is presented in appendix H Figure H3.

The client logs in the system using his user name and password; while he already specified during registration phase. If the client fails to login 3-times; the system will block his account for 10-minutes for security purposes. client can restore his password; the client has to supply the email used in the registration process and the password will be emailed to it.

Figure 25: Supermarket system requirements document without preprocessing

Results Analysis and Comparison

The generated model from the requirements without considering preprocessing was of bad accuracy as most of the objects and attributes were not identified. However; preprocessing the requirements was of great value, as the original sentences failed in representing the system correctly. Therefore accuracy of the model after preprocessing is quite evident.

4.2.3 Discussion

After analyzing the results of each of the conducted experiment we arrived at the following:

- Our framework was able to identify all objects identified in all of the previous researches. Table 4 shows all the objects identified in the previous researches are also identified in our work. Our frames work identified more objects than those identified in previous researches thus the modeled requirements were more accurate and complete.

Table 4: A comparison of the objects identified in our work and previous researches work

Experiment Name	Our Results	Research Results
Dinning Philosophers'	The objects identified are: *philosopher*, *fork* and *table*.	The objects identified are: *philosopher*, *fork*.
Bank	The objects identified are: *bank*, *account*, *ATM*, the data type *ID*, the data type *PIN*, the data type *balance* and the object *amount*.	The objects identified are: *bank*, *account* and *ATM*.
Elevator	The Object *left* was identified.	The Object *left* was identified.

- Our framework was able to identify most of the attributes identified in all of the previous researches. Table 5 shows all the attributes identified in the previous researches are also identified in our work. All the attributes identified in previous researches were also identified in our work except for one attribute, which is the direction in the elevator experiment.

Table 5: A comparison of the attributes identified in our work and previous researches work

Experiment Name	Our Results	Research Results
Dinning Philosophers'	The attributes identified: *philosopher*, *fork*, *side* and table.	The attribute identified in both objects was *location*.
Bank	The attributes identified: *service types*, *deposit*, *balance check*, *ID*, *PIN*, *bank*, *balance*, *amount*, *sequence*, *order*, *updates balance*, *account list*, *real number*, *integer* and *data fields*.	The attributes identified: *Account list*, *ID*, *PIN*, *balance* and *amount*.
Elevator	The attributes identified: *manufacturer*, *buttons list*, *floor*, *priority*, *emergency button*, *warning signal*, *site manager*, *mechanism*, *status*, *request*, *destination*, and *door*.	The attributes identified: *lift button*, *direction*, *status* and *request.*

- Our framework can identify relations if they were expressed using subset of English natural language, thus while running the scientific set experiments without preprocessing our framework was not able to identify any relation. However the comparison results of identifying relations are presented in Table 6.

Table 6: A comparison of the relations identified in our work and previous researches work

Experiment Name	Our Results	Research Results
Dinning Philosophers'	None	Take relation was identified.
Bank	None	None
Elevator	None	None

- Our framework was capable of transforming the generated module back to natural language enabling us to read the generated module in a natural way, while this was not possible in the previous researches. Table 7 presents the comparison results in turn of generating natural language requirements.

Table 7: A comparison of the ability to generate natural language out of class diagrams

Experiment Name	Our Results	Research Results
Dinning Philosophers'	Present	Not present
Bank	Present	Not present
Elevator	Present	Not present

- The output generated from our framework is highly dependant on the quality of its input. This is quite visible in Table 8 where good quality resulted in the identification of all the objects and attributes present in the requirements documents provided.
- Preprocessing requirements in the sense that poorly structured requirements are converted to highly structured ones resulted in more accurate and complete output

from out framework. Table 8 expresses the difference between the results from running the experiments with and without preprocessing.

Table 8: A comparison of the objects and attributes identified after running the experiments 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.

- Our framework is capable of identifying relations ships if they were explicitly expressed in the requirements. Table 9 expresses the relation ships identified while running the experiments with and without preprocessing.

Table 9: A comparison of the relations identified after running the experiments with and without preprocessing

Experiment Name	Without preprocessing	With preprocessing
Hotel reservation	No relations were identified.	Relations were identified.
Therapy center	No relations were identified.	Relations were identified.
School system	No relations were identified.	Relations were identified.
Construction system	No relations were identified.	Relations were identified.
Book store	No relations were identified.	Relations were identified.
Supermarket system	No relations were identified.	Relations were identified.

5. CONCLUSIONS AND FUTURE WORKS

This chapter presents some conclusions and recommended future works.

5.1 Conclusions

After running the experiments it was clear that the accuracy of the model generated is determined by the quality of the requirements presented. The quality of the requirements as mentioned earlier is measured by the level of completeness and unambiguity they posses. For that reason the academic set has higher accuracy rates rather than the student set excluding the output after preprocessing.

Our framework succeeded in identifying more objects and attributes than those were identified in the original papers for the academic set. We consider this ability as a benefit for elicitation what could form an important object and should be mentioned in the requirements document and what is not. As the requirements document forms the contract between the supplier and the client every entity in that document should be of value in the module, if it is not it should not be in the document in the first place. Thus our framework serve as a tool for analyzing requirements where one can identify objects and attributes and work from that point into detailing their description and there relation to the described system. One can also think of our framework as a tool that aids in the design process as it can convert from natural language into a class diagram model. The framework can also be used as a natural language generator or a class diagram explaining system, this usage is possible because the built framework is capable of reverse engineering class diagrams into natural language. The generated language from reverse engineering a class diagram is characterized by being simple, accurate and complete. Last but not least our framework

can be used as an environment for maintaining both the model and the requirements document where neither one becomes obsolete from the other.

5.2 Future Works

Despite of the good results obtained from the built framework, we think that many enhancements are yet to be done aiming to build a system rather than a framework.

Recommended future work that may be done regarding to this thesis can be summarized as follows:

- Enhancing the capabilities of our framework by adopting other natural languages beside the English language.
- Enhancing the capabilities of the parser by building our own customized parser than using a shallow one.
- Enhancing the system to model functional operation as well as data, this could be done by incorporating the use of XMI instead of XML schema.
- Building a customized semantical analyzer that uses domain knowledge to give a percentage of strength to the identified objects. And using a threshold to determine valid ones from invalid ones.
- Using the semantical analyzer in automating redundancy elimination and the identification of different words sharing a similar meaning.
- Enhancing the relation extractor by enabling it to extract relations without the need for a customized subset of the English language.
- Enabling the system to use an SQL database repository as well as an XML repository.

- Building a GUI along with the available command line interface to make the system more user-friendly.
- Enhancing the capabilities of the natural language extractor by adopting a rule based language generator that uses a domain based database of language constructs and usage heuristics.

REFERENCES

(Ambriola and Gervasi, 2000) Ambriola, V. and Gervasi, V., (2000). **Environmental Support for Requirements Writing and Analysis**. Thesis, Information Science and Technology Institute, Pisa, Italy.

(Barnett et al., 1990) Barnett, J., Knight, K., Mani, I. and Rich, E., (1990). Knowledge and Natural Language Processing, **Communications of the ACM**, vol. 33, no. 8, pp. 50-71.

(Belford et al., 1976) Belford, P. C., Bond, A. F., Henderson, D. G. and Sellers, L. S., (1976). Specifications A Key to Effective Software Development, Proceedings of the 2nd international conference on Software engineering, **International Conference on Software Engineering**, pp. 71-79.

(Bell and Thayer, 1976) Bell, T. E. and Thayer, T. A., (1976). Software Requirements: Are They Really a Problem?. Proceedings of the 2nd international conference on Software engineering, **International Conference on Software Engineering**, pp. 61-68.

(Boehm, 1981) Boehm, B. W., (1981). **Software Engineering Economics**, Prentice Hall.

(Boggs and Boggs, 1999) Boggs, W. and Boggs, M., (1999). **Mastering UML with Rational Rose**, SYBEX Inc.

(Broy, 2006) Broy, M., (2006). Challenges in automotive software engineering. Proceeding of the 28th international conference on Software engineering, **International Conference on Software Engineering**, pp. 33-42.

(Callele and Makaroff, 2006) Callele, D. and Makaroff, D., (2006). Teaching requirements engineering to an unsuspecting audience, Proceedings of the 37th SIGCSE technical symposium on Computer science education, **Technical Symposium on Computer Science Education**, pp. 433-437.

(Cunningham et al., 2005) Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Ursu, C., Dimitrov, M., Dowman, M., Aswani, N. and Roberts, I., (2005). **Developing Language Processing Components with GATE**, University of Sheffield.

(Duffy et al., 1995) Duffy, D., Macnish, C., Mcdermid, J. and Morris, P., (1995). A Framework for Requirements Analysis Using Automated Reasoning, Proceedings of the 7th International Conference on Advanced Information Systems Engineering, **Lecture Notes In Computer Science**, vol. 932, pp. 68-81.

(Han, 2001) Han, J., (2001). TRAM: A Tool for Requirements and Architecture Management. Proceedings of the 24th Australasian conference on Computer science, **ACM International Conference Proceeding Series**, vol. 11, pp. 60-68.

(Hunter et al., 2004) Hunter, D., Watt, A., Rafter, J., Duckett, J., Ayers, D., Chase, N., Fawcett, J., Caven, T. and Patterson, B., (2004). **Beginning XML**, 3rd edition, Wiley Publishing.

(IEEE, 1998) IEEE. The Institute of Electrical and Electronic Engineers, Inc., (1998). IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830. **http://www.laas.fr/~kader/srs.pdf**

(Jacobson et al., 1999) Jacobson, I., Booch, G. and Rumbaugh, J., (1999). **The Unified Software Development Process**, Addison Wesley.

(Laird, 2001) Laird, C., (2001). XML and UML Combine to Drive Product Development, **http://www-128.ibm.com/developerworks/xml/library/x-xmi.**

(Lee, 2003) Lee, B. S., (2003). **Automated Conversion from a Requirements Document to an Executable Formal Specification Using Two-Level Grammar and Contextual Natural Language Processing**, Thesis, University of Alabama, Birmingham.

(Lee and Bryant, 2003) Lee, B. and Bryant, B., (2003). Applying XML Technology for Implementation of Natural Language Specifications, **International Journal of Computer Systems Science & Engineering**, vol. 18, no. 5, pp. 279-300.

(Leffingwell and Widrig, 2003) Leffingwell, D. and Widrig, D., (2003). **Managing Software Requirements: A Use Case Approach**, 2nd edition, Addison Wesley.

(Mannion and Keepence, 1995) Mannion M. and Keepence B., (1995). SMART Requirements, ACM SIGSOFT, **Software Engineering Notes** vol. 20 no. 2, pp. 42-47.

(Marchal B., 2004) Marchal, B., (2004). Working XML: UML, XMI, and code generation, **http://www-128.ibm.com/developerworks/xml/library/x-wxxm23**

(Meta, 2006) Meta Integration Technology, Inc., (2006). Reference Guide, **http://www.metaintegration.net/Products/MIMB**

(NASA) NASA Software Assurance Technology Center. NASA Software Documentation Standard. **http://satc.gsfc.nasa.gov/assure/docstd.html**.

(Naur and Randell, 1968) Naurm P. and Randellm B., (1968). NATO Science Committee, **Proc. of the NATO Working Conference on Software Engineering**. Oct. 1968.

(Perez-Gonzalez and Kalita, 2002) Perez-Gonzalez, H. G. and Kalita, J. K, (2002). Automatically Generating Object Models from Natural Language Analysis, Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, **Conference on Object Oriented Programming Systems Languages and Applications**, pp. 86-87.

(Presland and Hennell, 1986) Presland, S. and Hennell, M. A., (1986). **Detecting Functionality in Natural Language Text**, Unpublished Tech. Report, Dept. of Statistical and Computational Mathematics.

(Pressman R., 2005) Pressman, R., (2005). **Software Engineering**, 6th edition, McGRAW.HILL

(Rambaugh et al., 1999) Rambaugh, J., Jacobson, I. and Booch, G., (1999). **The Unified Modeling Language Reference Manual**, Addison Wesley.

(Saeki et al., 1989) Saeki, M., Horai, H. and Enomoto, H., (1989). Software Development Process from Natural Language Specification, Proceedings of the 11th international conference on Software engineering, **International Conference on Software Engineering**, pp. 64-73.

(Sommerville, 2004) Sommerville I., (2004). **Software Engineering**, 7th edition , Addison Wesley.

(Standish, 1995) The Standish Group, (1995). Software Chaos, **www.projectsmart.co.uk/docs/chaos_report.pdf**

(VTC) VTC. Microsoft Visio 2003 Tutorial, **http://www.softwaretrainingtutorials.com/ms-visio-2003.php**.

(Wiegers, 2003) Wiegers, K. E., (2003). **Software Requirements**, 2nd edition, Microsoft Press.

(Wilson et al., 1997) Wilson, W. M., Rosenberg, L. H. and Hyatt, L. E., Automated Analysis Requirement Specification, Proceedings of the 19th international conference on Software engineering, **International Conference on Software Engineering**, pp. 161-171.

(Zave, 1997) Zave, P., (1997). Classification of research efforts in requirements engineering, **ACM Computing Surveys**, vol. 29, no. 4, pp. 315-321.

Appendix A: Bank Requirements Document

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="ATM">
        <xs:annotation>
            <xs:documentation>38, 38</xs:documentation>
        </xs:annotation>
        <xs:attribute name="service types">
            <xs:annotation>
                <xs:documentation>41, 41</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="deposit">
            <xs:annotation>
                <xs:documentation>45, 45</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="balance check">
            <xs:annotation>
                <xs:documentation>47, 47</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="ID">
            <xs:annotation>
                <xs:documentation>54, 54</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="PIN">
            <xs:annotation>
                <xs:documentation>56, 56</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="bank">
            <xs:annotation>
                <xs:documentation>58, 58</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="balance">
            <xs:annotation>
                <xs:documentation>60, 60</xs:documentation>
            </xs:annotation>
```

```
        </xs:attribute>
        <xs:attribute name="amount">
            <xs:annotation>
                <xs:documentation>64, 64</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="ID">
            <xs:annotation>
                <xs:documentation>66, 66</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="PIN">
            <xs:annotation>
                <xs:documentation>68, 68</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="balance">
            <xs:annotation>
                <xs:documentation>70, 70</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="sequence">
            <xs:annotation>
                <xs:documentation>74, 74</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="balance">
            <xs:annotation>
                <xs:documentation>96, 96</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="bank">
            <xs:annotation>
                <xs:documentation>98, 98</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="ID">
            <xs:annotation>
                <xs:documentation>100, 100</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="amount">
            <xs:annotation>
                <xs:documentation>104, 104</xs:documentation>
```

```
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="ID">
            <xs:annotation>
                <xs:documentation>106, 106</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="PIN">
            <xs:annotation>
                <xs:documentation>108, 108</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="balance">
            <xs:annotation>
                <xs:documentation>110, 110</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="order">
            <xs:annotation>
                <xs:documentation>114, 114</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="balance">
            <xs:annotation>
                <xs:documentation>118, 118</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="amount">
            <xs:annotation>
                <xs:documentation>120, 120</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="then  updates balance">
            <xs:annotation>
                <xs:documentation>122,  118,  118,  122,  118,
118</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="bank">
            <xs:annotation>
                <xs:documentation>124, 124</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="ID">
```

```
            <xs:annotation>
                <xs:documentation>126, 126</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="balance">
            <xs:annotation>
                <xs:documentation>130, 130, 136</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="ID">
            <xs:annotation>
                <xs:documentation>132, 132</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="PIN">
            <xs:annotation>
                <xs:documentation>134, 134</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="balance">
            <xs:annotation>
                <xs:documentation>136, 130, 130, 136, 136, 130, 130,
136</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:element name="ATM" type="ATM">
        <xs:annotation>
            <xs:documentation>38, 38</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="Balance">
        <xs:annotation>
            <xs:documentation>34, 34</xs:documentation>
        </xs:annotation>
        <xs:attribute name="real  number">
            <xs:annotation>
                <xs:documentation>36, 36</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:element name="Balance" type="Balance">
        <xs:annotation>
            <xs:documentation>34, 34</xs:documentation>
```

```
        </xs:annotation>
    </xs:element>
    <xs:complexType name="Bank">
        <xs:annotation>
            <xs:documentation>0, 0</xs:documentation>
        </xs:annotation>
        <xs:attribute name="accounts List">
            <xs:annotation>
                <xs:documentation>2, 2</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="ID">
            <xs:annotation>
                <xs:documentation>6, 6, 14</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="PIN">
            <xs:annotation>
                <xs:documentation>8, 8</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="balance">
            <xs:annotation>
                <xs:documentation>10, 10</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="updates balance">
            <xs:annotation>
                <xs:documentation>12,  10,  10,  12,  10,
10</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="ID">
            <xs:annotation>
                <xs:documentation>14,  6,  6,  14,  14,  6,  6,
14</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:element name="Bank" type="Bank">
        <xs:annotation>
            <xs:documentation>0, 0</xs:documentation>
        </xs:annotation>
    </xs:element>
```

```
<xs:complexType name="ID">
    <xs:annotation>
        <xs:documentation>28, 28</xs:documentation>
    </xs:annotation>
    <xs:attribute name="integers">
        <xs:annotation>
            <xs:documentation>32, 32</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:complexType>
<xs:element name="ID" type="ID">
    <xs:annotation>
        <xs:documentation>28, 28</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="PIN">
    <xs:annotation>
        <xs:documentation>30, 30</xs:documentation>
    </xs:annotation>
    <xs:attribute name="integers">
        <xs:annotation>
            <xs:documentation>32, 32</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:complexType>
<xs:element name="PIN" type="PIN">
    <xs:annotation>
        <xs:documentation>30, 30</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="account">
    <xs:annotation>
        <xs:documentation>16, 16</xs:documentation>
    </xs:annotation>
    <xs:attribute name="data fields">
        <xs:annotation>
            <xs:documentation>19, 19</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="ID">
        <xs:annotation>
            <xs:documentation>21, 21</xs:documentation>
        </xs:annotation>
    </xs:attribute>
```

```
        <xs:attribute name="PIN">
            <xs:annotation>
                <xs:documentation>23, 23</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="balance">
            <xs:annotation>
                <xs:documentation>26, 26</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:element name="account" type="account">
        <xs:annotation>
            <xs:documentation>16, 16</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="amount">
        <xs:annotation>
            <xs:documentation>77, 77</xs:documentation>
        </xs:annotation>
        <xs:attribute name="balance">
            <xs:annotation>
                <xs:documentation>84, 84, 88</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="balance">
            <xs:annotation>
                <xs:documentation>88,  84,  84,  88,  88,  84,  84,
88</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="amount">
            <xs:annotation>
                <xs:documentation>90, 90</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:element name="amount" type="amount">
        <xs:annotation>
            <xs:documentation>77, 77</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="service">
        <xs:annotation>
```

```
            <xs:documentation>50, 50</xs:documentation>
        </xs:annotation>
        <xs:attribute name="ID">
            <xs:annotation>
                <xs:documentation>54, 54</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="PIN">
            <xs:annotation>
                <xs:documentation>56, 56</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="bank">
            <xs:annotation>
                <xs:documentation>58, 58</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="balance">
            <xs:annotation>
                <xs:documentation>60, 60</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:element name="service" type="service">
        <xs:annotation>
            <xs:documentation>50, 50</xs:documentation>
        </xs:annotation>
    </xs:element>
</xs:schema>
```

Figure A1: Bank system XML schema

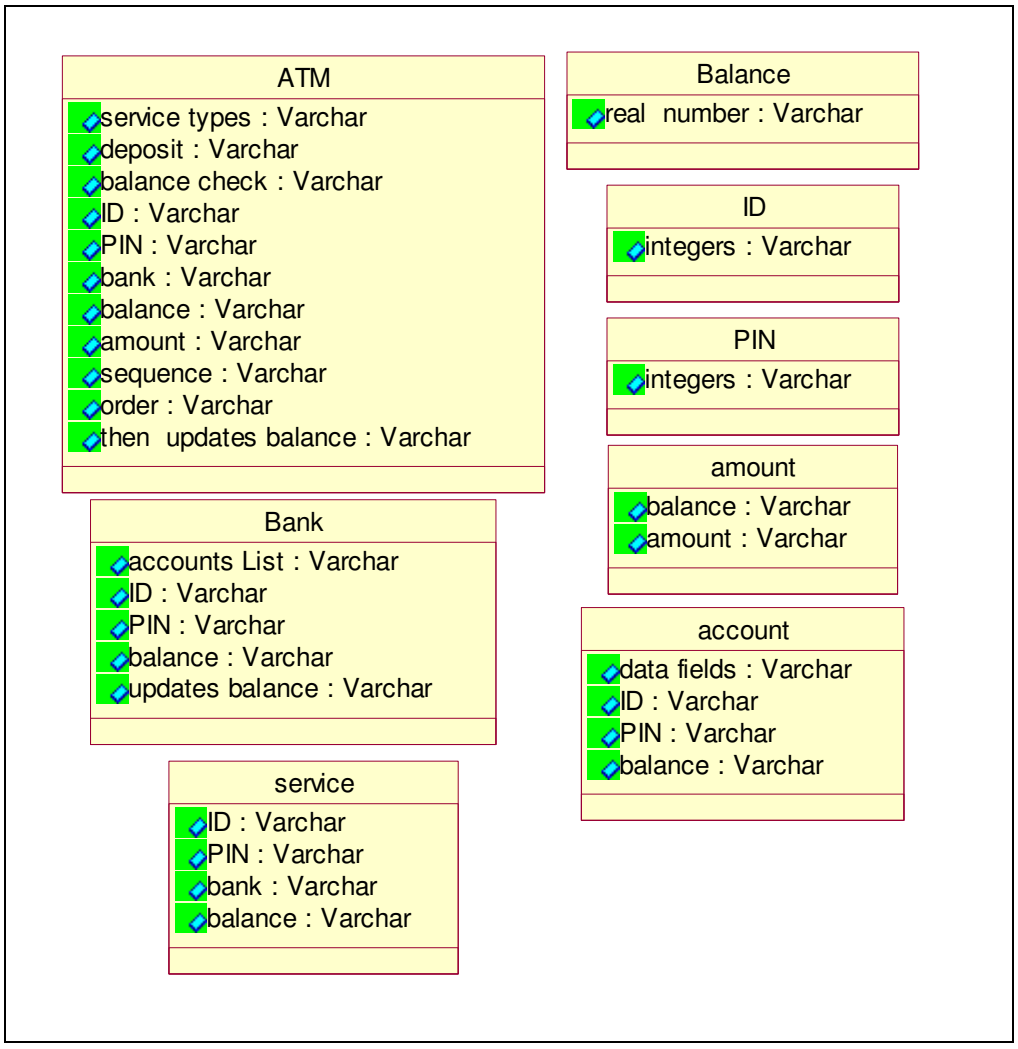

Figure A2: Bank system UML model

Each ATM has service types,

Each ATM has a deposit,

Each ATM has a balance check,

Each ATM has a ID,

Each ATM has a PIN,

Each ATM has a bank,

Each ATM has a balance,

Each ATM has an amount,

Each ATM has a sequence,

Each ATM has an order,

Each ATM has a then updates balance.
Each Balance has a real number.
Each Bank has an accounts List,
Each Bank has a ID,
Each Bank has a PIN,
Each Bank has a balance,
Each Bank has an updates balance.
Each ID has integers.
Each PIN has integers.
Each account has data fields,
Each account has a ID,
Each account has a PIN,
Each account has a balance.
Each amount has a balance,
Each amount has an amount.
Each service has a ID,
Each service has a PIN,
Each service has a bank,
Each service has a balance.

Figure A3: Bank system reversed natural language

Appendix B: Elevator Requirements Document

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="lift">
        <xs:annotation>
            <xs:documentation>10, 10</xs:documentation>
        </xs:annotation>
        <xs:attribute name="manufacturer">
            <xs:annotation>
                <xs:documentation>16, 16</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="buttons List">
            <xs:annotation>
                <xs:documentation>39, 39</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="floor">
            <xs:annotation>
                <xs:documentation>43, 43</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="equal  priority">
            <xs:annotation>
                <xs:documentation>166, 166</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="emergency button">
            <xs:annotation>
                <xs:documentation>188, 188</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="warning signal">
            <xs:annotation>
                <xs:documentation>194, 194</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="site manager">
            <xs:annotation>
                <xs:documentation>199, 199</xs:documentation>
            </xs:annotation>
```

```
        </xs:attribute>
        <xs:attribute name="mechanism">
            <xs:annotation>
                <xs:documentation>209, 209</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="out-of-service  status">
            <xs:annotation>
                <xs:documentation>213, 213</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="requests">
            <xs:annotation>
                <xs:documentation>135, 135</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="final  destination">
            <xs:annotation>
                <xs:documentation>144, 144</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="door">
            <xs:annotation>
                <xs:documentation>147, 147</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:element name="lift" type="lift">
        <xs:annotation>
            <xs:documentation>10, 10</xs:documentation>
        </xs:annotation>
    </xs:element>
</xs:schema>
```

Figure B1: Lift system XML schema

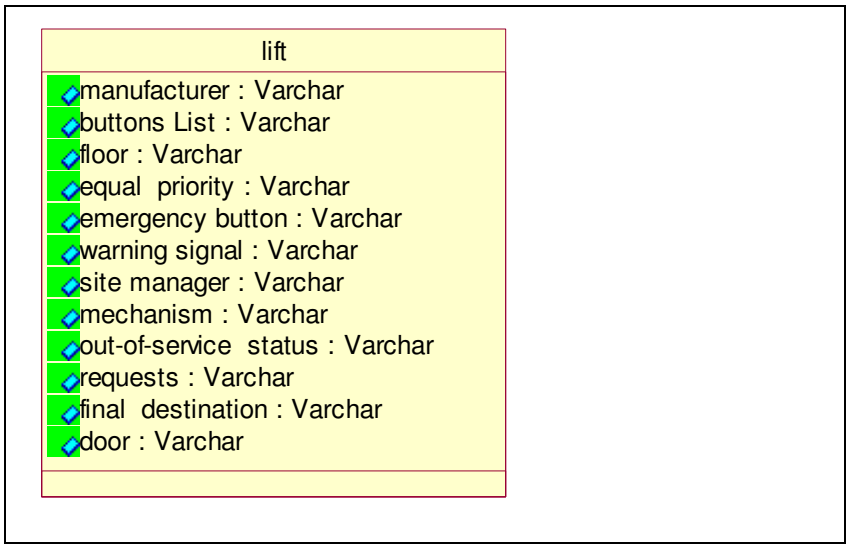

Figure B2: Lift system UML model

Each lift has a manufacturer,

Each lift has a buttons List,

Each lift has a floor,

Each lift has an equal priority,

Each lift has an emergency button,

Each lift has a warning signal,

Each lift has a site manager,

Each lift has a mechanism,

Each lift has out of service status,

Each lift has requests,

Each lift has a final destination,

Each lift has a door.

Figure B3: Lift system reversed natural language

Appendix C: Hotel Reservation Requirements Document

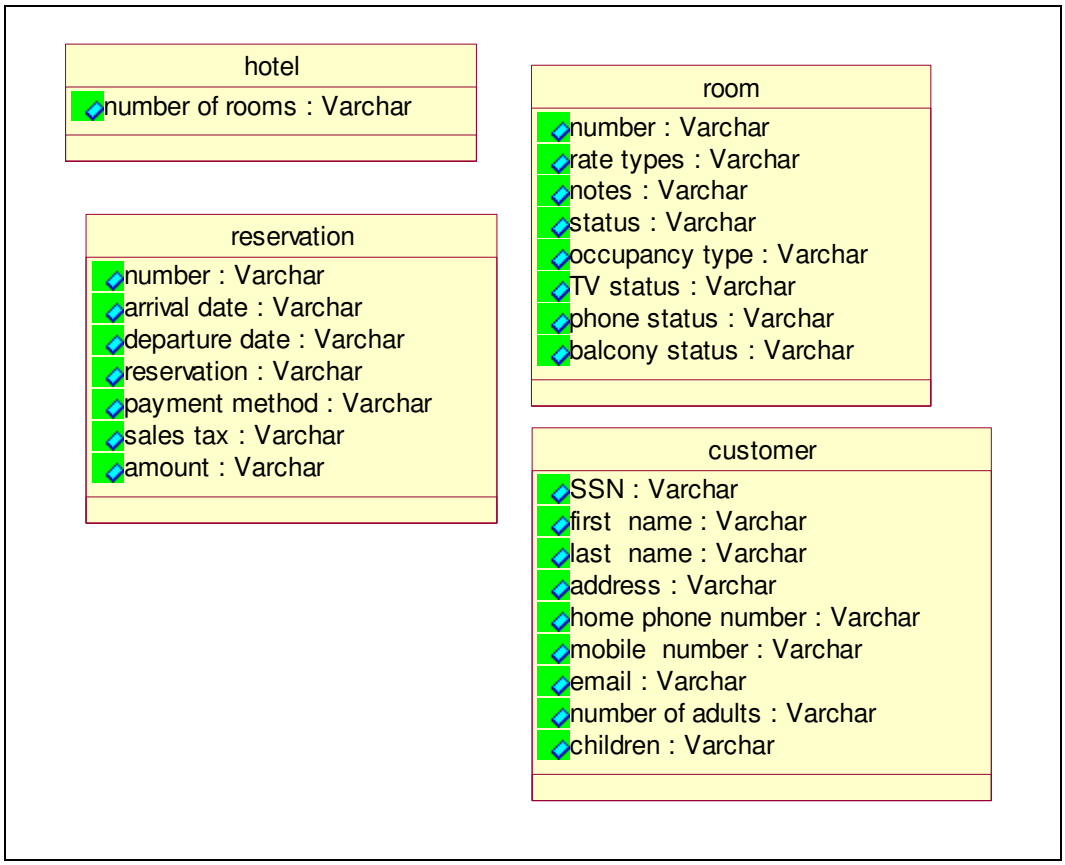

Figure C1: Hotel reservation system UML model

Each customer has a SSN.

Each customer has a first name.

Each customer has a last name.

Each customer has address.

Each customer has a home phone number.

Each customer has a mobile number.

Each customer has an email.

Each customer has number of adults.

Each customer has a children.

Each hotel has number of rooms.

Each reservation has a number.

Each reservation has an arrival date.
Each reservation has a departure date.
Each reservation has a reservation.
Each reservation has a payment method .
Each reservation has a sales tax.
Each reservation has an amount.
Each room has a number.
Each room has rate types.
Each room has notes.
Each room has status.
Each room has an occupancy type.
Each room has TV status.
Each room has phone status.
Each room has balcony status.
customer is associated with reservation.
customer is associated with hotel.
customer is associated with room.
hotel is associated with room.
hotel is associated with reservation.
hotel is associated with reservation.
hotel is associated with customer.
hotel is associated with reservation.
reservation is associated with customer.
reservation is associated with hotel.
reservation is associated with hotel.
reservation is associated with hotel.
room is associated with hotel.
room is associated with customer.

Figure C2: Hotel reservation system reversed and preprocessed requirements

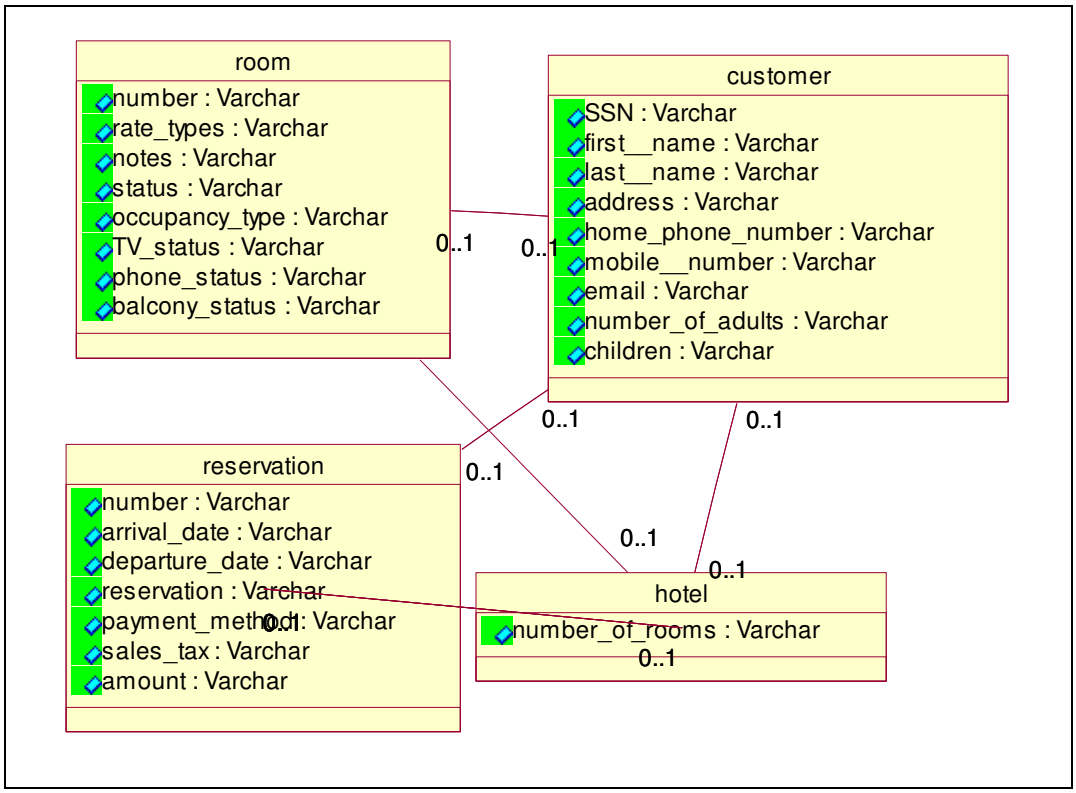

Figure C3: Hotel reservation system UML model after preprocessing

Appendix D: Therapy Center Requirements Document

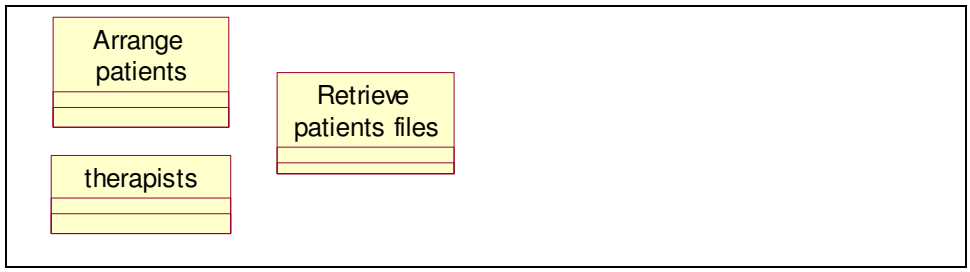

Figure D1: Therapy center UML model

All Patients have a reservation.

All Patients have a file.

Patients is associated with Therapists.

Patients is associated with System.

System is associated with Therapists.

System is associated with Patients.

Therapists is associated with System.

Therapists is associated with Patients.

Figure D2: Therapy center reversed and preprocessed requirements

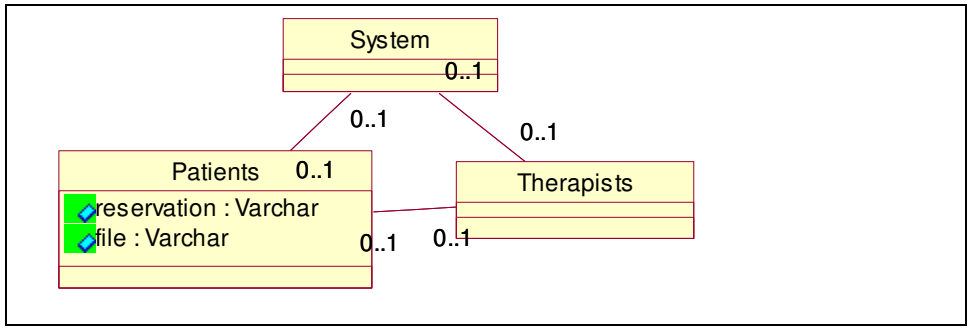

Figure D3: Therapy center UML model after preprocessing

Appendix E: School system Requirements Document

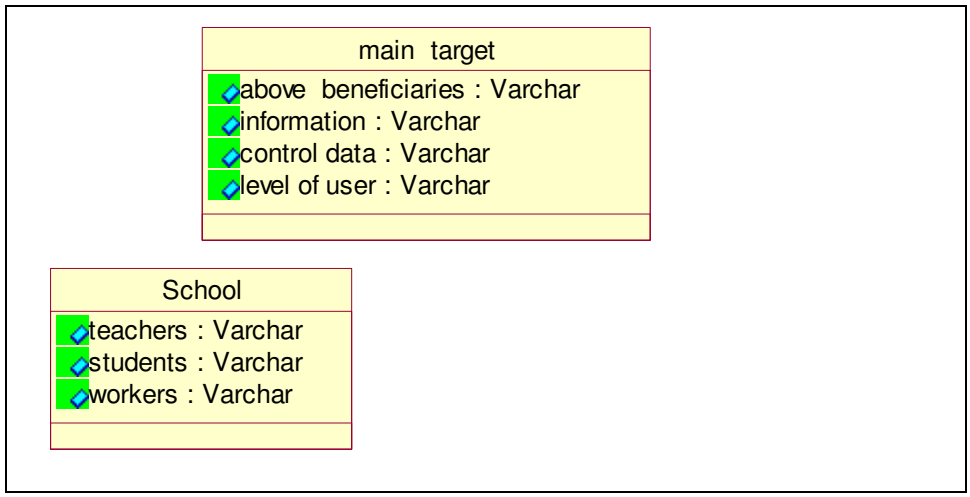

Figure E1: School system UML model

Data is associated with System.

School is associated with Workers.

Students is associated with School.

System is associated with System.

Teachers is associated with School.

Workers is associated with School.

Figure E2: School system reversed and preprocessed requirements

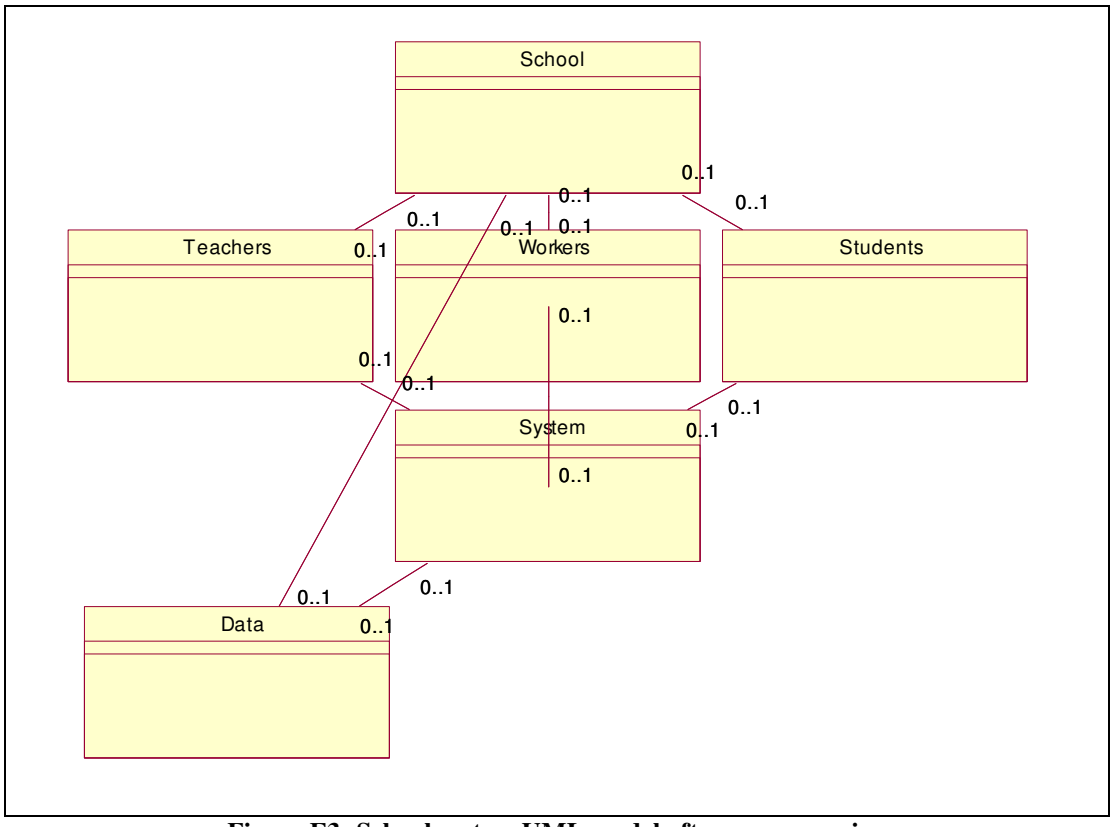

Figure E3: School system UML model after preprocessing

Appendix F: Construction System Requirements Document

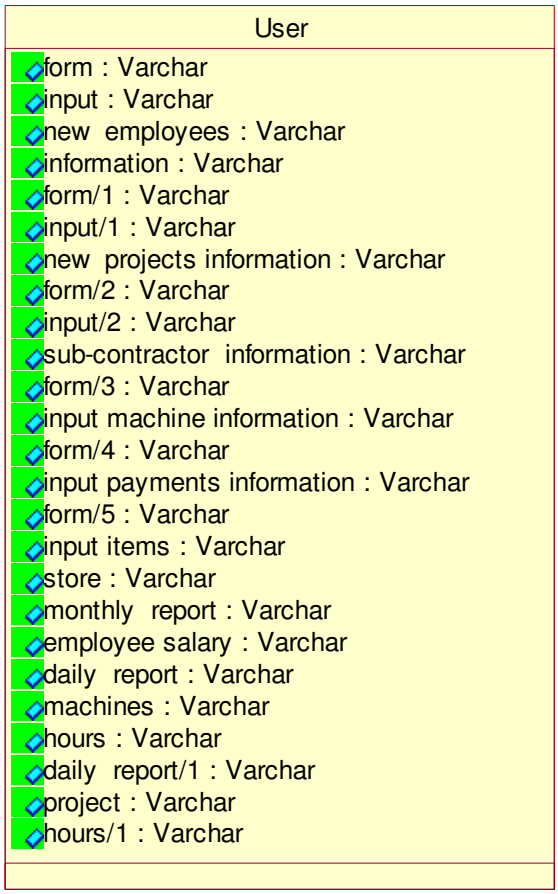

Figure F1: Construction system UML model

Each User has a form.

Each User has a monthly report.

Each User has a daily report.

Each daily report has generalization of machine status.

Each daily report has generalization of project status.

Each form has a generalization of new employee information.

Each form has a generalization of new projects information.

Each form has a generalization of sub-contractor information.

Each monthly report has a generalization of employee salary.

User is associated with monthly report.

daily report is associated with User.

form is associated with User.

monthly report is associated with monthly report.

Figure F2: Construction system reversed and preprocessed requirements

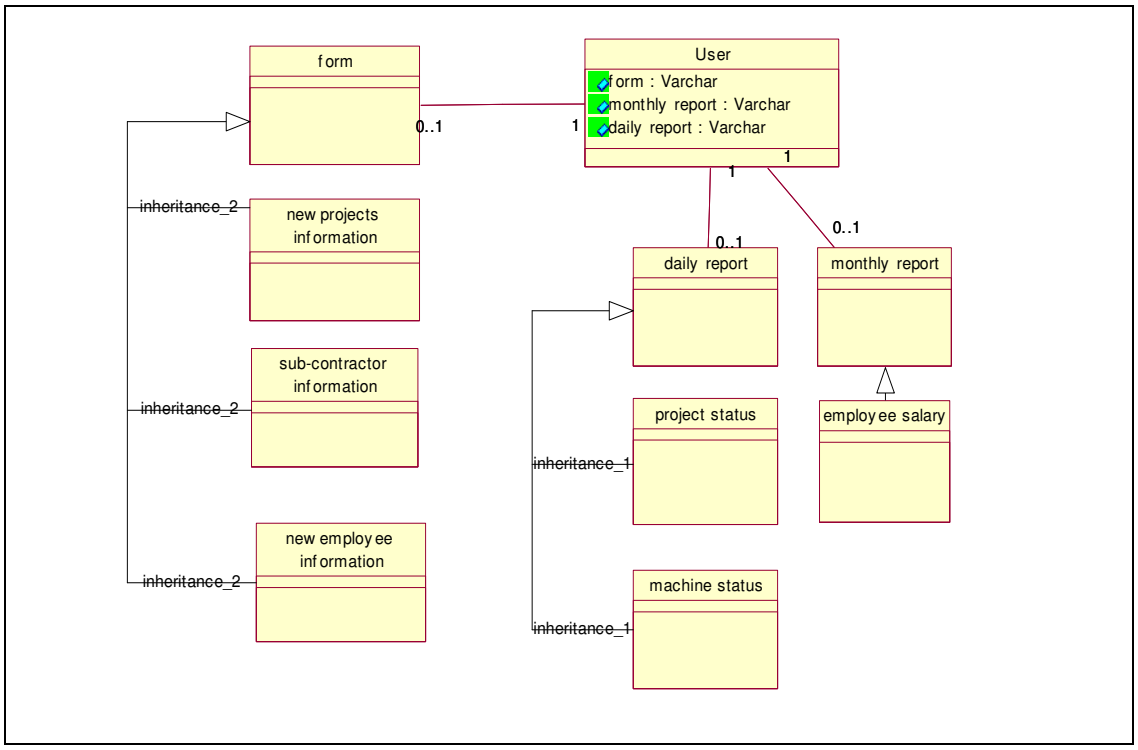

Figure F3: Construction system UML model after preprocessing

Appendix G: Book Store Requirements Document

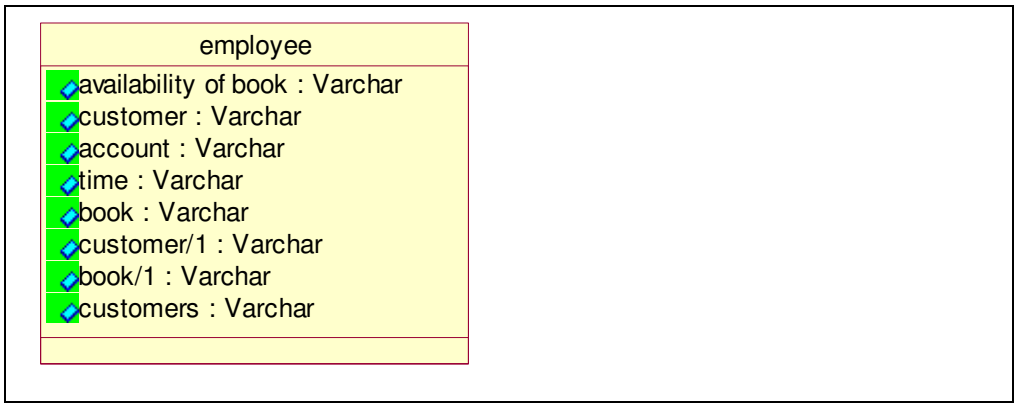

Figure G1: Book store system UML model

Each book has availability status.

Each employee has a book.

Each employee has a customer.

account is associated with customer.

book is associated with customer.

customer is associated with employee.

employee is associated with book.

Figure G2: Book store system reversed and preprocessed requirements

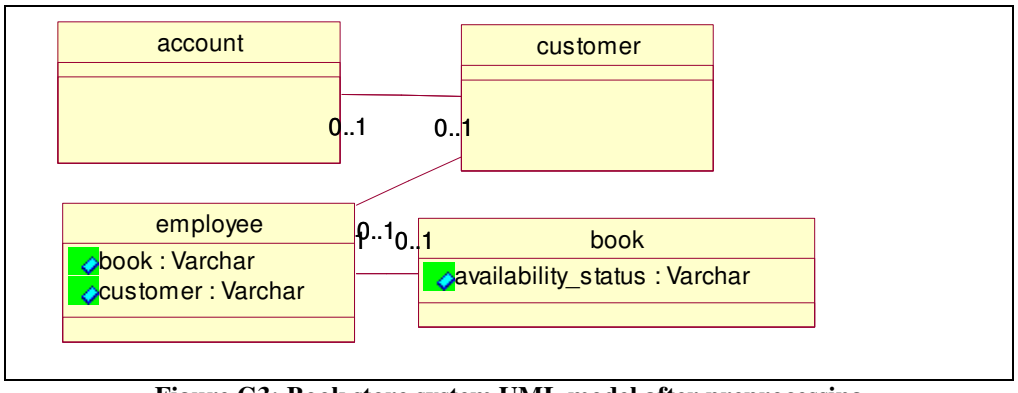

Figure G3: Book store system UML model after preprocessing

Appendix H: Supermarket Requirements Document

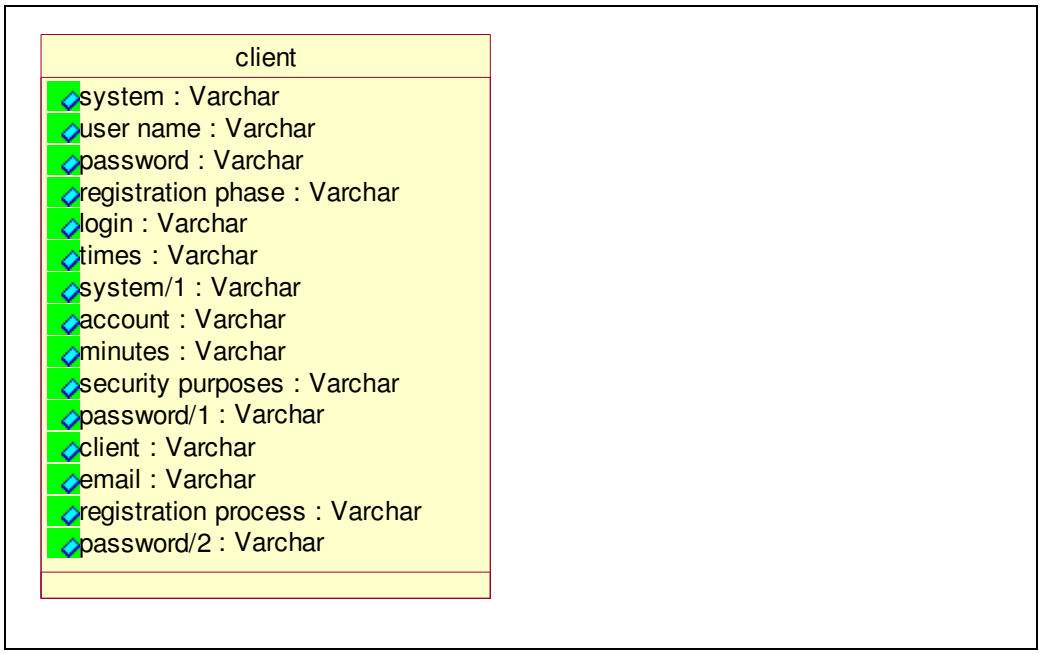

Figure H1: Supermarket system UML model

Each client has an user name.

Each client has a password.

Each client has an account.

Each client has an email.

System is associated with client.

client is associated with System.

Figure H2: Supermarket system reversed and preprocessed requirements

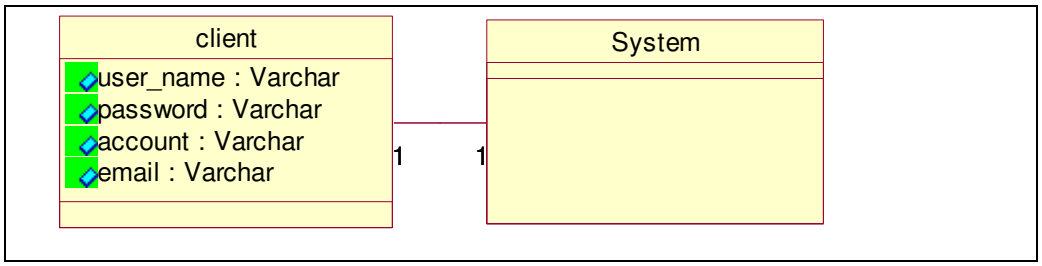

Figure H3: Supermarket system UML model after preprocessing

استخلاص وصف متطلبات البرمجيات باستخدام معالجة اللغات الطبيعية

إعداد
يارا الخضر

المشرف
الدكتور أمجد هديب

المشرف المشارك
الدكتور بسام حمو

ملخص

تتناول هذه الرسالة إمكانية أتمتة تحويل متطلبات النظام المعبر عنها باستخدام اللغات الطبيعية إلى (Class Diagram) وعكس هذا التحويل من (Class Diagram) إلى اللغة الطبيعية مع أقل خسارة في البيانات.

إن النظرية التي تقود هذه الدراسة هي استقلالية المعنى اللغوي عن اللغة التي تقدمه وهذا يعني أنه يمكن وصف المعنى ذاته باستخدام لغات مختلفة.

في هذه الدراسة تم اقتراح و تطبيق إطار عمل قادر على معالجة المتطلبات المعبر عنها باللغة الإنجليزية وإنتاج تمثيل (XML) لها. ومن ثم و باستخدام محلل نظم يعتمد على قواعد معينة يترجم تمثيل (XML) إلى (Class Diagram).

كما أن النظام يستطيع وباستطاعة النظام أن يحول (Class Diagram) إلى تمثيل (XML). ومن الممكن النظر إلى تمثيل (XML) على أنه الطريقة التي يخزن بها النظام البيانات.

ويمكن تبرير الاحتياجات لهذه الأتمتة كالتالي:

أولا، إعادة استعمال المواصفات حيث أن النظام المقترح يحافظ على توافر المتطلبات معدلة و متوافقة مع النظام الذي تصفه هذه المتطلبات.

ثانيا، كفاءة الوقت حيث يمثل الوقت شرط يحد بناء المشروع.

ثالثا، وعدم توفر الخبرة وذلك عند عدم توافر مهندس متطلبات كفؤ لتصميم المتطلبات.

في هذه الرسالة نقدم تصميم إطار العمل المقترح، بالإضافة إلى حالة دراسية مفصلة لعرض إمكانية إطار العمل المقترح.

إن المتطلبات التي جمعت للحالة الدراسية تم جمعها من مجالات مختلفة مصدرها إما علمي و إما من طلاب هندسة البرمجيات. و تختلف المتطلبات المجمعة من حيث جودة لغتها و تعقيدها.

تشير نتائج التجارب المنبثقة من إطار العمل المقترح إلى إمكانية استعمال هذا النظام لتصميم المتطلبات و المحافظة عليها بشكل متناسق و مواكب للتعديلات الحاصلة على المتطلبات. بالإضافة إلى إمكانية استعمال النظام للتعرف على ال(Objects) و ال(Attributes) لكل (Class).

يتميز إطار العمل المقترح بإمكانية تزويده بإضافات مما يتيح إمكانية تطويره إلى نظام متكامل.