

**A Methodology for Semi-Automatic Assistance in  
Elicitation and Analysis of Textual User Requirements**

**Shadi Moradi Seresht**

A Thesis  
in  
The Department  
of  
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements for  
the Degree of Master of Computer Science at  
Concordia University  
Montreal, Quebec, Canada

September 2008

© Shadi Moradi Seresht, 2008



Library and  
Archives Canada

Published Heritage  
Branch

395 Wellington Street  
Ottawa ON K1A 0N4  
Canada

Bibliothèque et  
Archives Canada

Direction du  
Patrimoine de l'édition

395, rue Wellington  
Ottawa ON K1A 0N4  
Canada

*Your file    Votre référence*

*ISBN: 978-0-494-45325-4*

*Our file    Notre référence*

*ISBN: 978-0-494-45325-4*

**NOTICE:**

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

**AVIS:**

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

---

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

  
**Canada**

## **ABSTRACT**

# **A Methodology for Semi-Automatic Assistance in Elicitation and Analysis of Textual User Requirements**

**Shadi Moradi Seresht**

Requirements Engineering (RE) is a sub-discipline within Software Engineering increasingly recognized as a critical component in the success of a software development project. With the escalating complexity of software requirements, problems of traditional requirements engineering techniques, including the use of natural language text, are becoming increasingly apparent.

This research aims to assist software analysts in dealing with the challenges that exist in correctly understanding user requirements during the interactive process of requirement elicitation and analysis. It proposes a methodology related to visualization of textual requirements and ways of making them shared, reviewed and debated by the stakeholders. The proposed methodology serves as a basis for a semi-automated process aimed at capturing the conceptual model of the software system under development and its high-level services from user requirements text. The extracted information can be used by analysts in their in-depth study of the requirements text and in avoiding the risks associated with specifying poor or invalid requirements.

The approach is based on a syntactic analysis and formalization of the text written in natural language and enriched with domain-related information extracted from reusable domain-specific data models. The applicability of this research is illustrated with a case study. A prototype implementing our methodology is developed as a proof-of-concept. The results of controlled experiments designed to evaluate our approach prove the validity of the methodology. The thesis discusses future work, issues, problems, and priorities. Furthermore, it proposes recommendations for textual requirements comprehension research.

## Acknowledgements

Initially, I would like to thank my supervisor: Dr. Olga Ormandjieva for her guidance and support, for being a great technical advisor and finally for being such a nice person. Your aptitude is appreciated.

Many thanks to Kaveh for inspiring me to continue my studies, for his love and patience; Moreover, for his review and feedback on the thesis. Invaluable and always remembered.

Thanks to my father for supporting me, and my mother for encouraging me.

I would also like to thank Dr. Yong Zeng (Institute for Information Systems Engineering, Concordia University) for providing information and references for my thesis.

Special thanks to everyone who helped me in preparing this thesis: All my friends who made the evaluations for this thesis possible, TROMLAB research group for their support, Sam, Chen, Ishrar, and Ayrin for their valuable contributions to the implementation.

Thanks to all other friends for scintillating talks and delightful coffee breaks on working days at 5:00.

# Contents

<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>xi</b>
<b>List of Abbreviations</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Statement . . . . .	4
1.2 Goal of the Research . . . . .	5
1.3 Contributions . . . . .	6
1.4 Organization of the Thesis . . . . .	7
<b>2 Background</b>	<b>8</b>
2.1 Requirements Engineering . . . . .	8
2.1.1 Feasibility Study . . . . .	9
2.1.2 Requirements Elicitation and Analysis . . . . .	9
2.1.3 Requirements Validation . . . . .	10
2.1.4 Requirements Management . . . . .	10

2.1.5	Scope of the Thesis . . . . .	10
2.2	Linguistics and Language Processing . . . . .	11
2.3	Applying Linguistic Analysis to RE . . . . .	12
2.4	READ Project . . . . .	13
2.4.1	Automatic Ambiguity Detection in User Requirements Documents . . . . .	13
2.4.2	Formal Graphical Presentation of the Requirement Text . . . .	14
<b>3</b>	<b>Related Work</b>	<b>18</b>
3.1	Introduction . . . . .	18
3.2	General Review . . . . .	18
3.3	Overview of Most Important Related Work . . . . .	24
3.3.1	NL-OOPS . . . . .	24
3.3.2	LIDA . . . . .	25
3.3.3	CM-Builder . . . . .	26
<b>4</b>	<b>Proposal</b>	<b>28</b>
4.1	Objective . . . . .	28
4.2	Summary of the Research Methodology . . . . .	29
<b>5</b>	<b>Research Methodology</b>	<b>32</b>
5.1	Hypothesis . . . . .	32
5.2	Methodology . . . . .	32
5.2.1	Preprocessing of the Text . . . . .	33

5.2.2	Structural Analysis . . . . .	33
5.2.3	Capturing High-Level System Services . . . . .	58
<b>6</b>	<b>Illustration</b>	<b>69</b>
6.1	User Requirements . . . . .	69
6.2	Preprocessing of the Requirement Text . . . . .	70
6.3	Structural Analysis . . . . .	72
6.4	Identification of the System Services . . . . .	77
<b>7</b>	<b>Prototype</b>	<b>86</b>
7.1	Overview . . . . .	86
7.2	Development Platform . . . . .	88
7.3	EVP Modules . . . . .	88
7.3.1	First-Cut Structural Visualizer . . . . .	88
7.3.2	Improved Structural Visualizer . . . . .	91
7.3.3	Context Use Case Visualizer . . . . .	91
7.3.4	Distance Calculator-Partitioner (DC-P) . . . . .	94
<b>8</b>	<b>Experimental Work</b>	<b>100</b>
8.1	Experiment . . . . .	100
8.1.1	Procedure . . . . .	101
8.1.2	Discussion . . . . .	102
<b>9</b>	<b>Discussion</b>	<b>107</b>
9.1	Intentions and Contributions . . . . .	108



9.2 Limitations . . . . .	110
<b>10 Conclusion and Future Work</b>	<b>112</b>

# List of Figures

1.1	NLP-driven quality assessment in RE. . . . .	3
2.1	Object. . . . .	15
2.2	Compound object. . . . .	15
2.3	Constraint relation. . . . .	15
2.4	Predicate relation. . . . .	16
2.5	Connection relation. . . . .	16
4.1	Summary of the research methodology. . . . .	30
5.1	ROM diagram. . . . .	44
5.2	SM for the sample sentence. . . . .	48
5.3	ECC for invoicing. . . . .	54
6.1	Sample ROM diagram. . . . .	72
6.2	Overall ROM presentation. . . . .	73
6.3	FCSM. . . . .	74
6.4	ISM. . . . .	78
6.5	CUCM diagram. . . . .	85

7.1	Architecture of READ. . . . .	87
7.2	EVP context menu. . . . .	98
7.3	EVP's logging through Eclipse error log view. . . . .	99
7.4	SM visualized as class diagram. . . . .	99
9.1	Supported sentence patterns. . . . .	111

# List of Tables

5.1	Graphical presentation of Rule 1. . . . .	34
5.2	Graphical presentation of Rule 2.1. . . . .	35
5.3	Attribute keywords. . . . .	36
5.4	Graphical presentation of Rules 2.2 and 2.3. . . . .	36
5.5	Graphical presentation of Rule 3. . . . .	38
5.6	Graphical presentation of Rule 4. . . . .	39
5.7	Graphical presentation of <i>Is kind of</i> . . . . .	40
5.8	Graphical presentation of Rule 6. . . . .	41
5.9	Graphical presentation of Rule 7.1. . . . .	41
5.10	Graphical presentation of Rule 7.2. . . . .	42
5.11	Graphical presentation of Rule 8. . . . .	43
5.12	Graphical presentation of Rule 9. . . . .	43
5.13	Syntactical analysis for the sample sentence. . . . .	44
5.14	ECC concept. . . . .	46
5.15	ECC generalization. . . . .	47
5.16	ECC generalization with all combination. . . . .	48

5.17	Conventions used for mutually inclusive and mutually exclusive concepts.	48
5.18	ECC attributes.	49
5.19	ECC aggregation/composition.	49
5.20	ECC association with * multiplicity.	50
5.21	ECC association with <i>OR</i> constraint.	51
5.22	ECC self association.	52
5.23	Calculating similarity.	66
5.24	Calculating dissimilarity.	66
6.1	Parts of speech for the sample sentence.	71
6.2	Syntactical analysis for the sample sentence.	71
6.3	Finding use cases from a formal presentation (system perspective).	80
6.4	Finding use cases from a formal presentation (actor perspective).	80
6.5	Extracted actors and use cases.	81
6.6	Distance calculations for S1.	82
6.7	Distance calculations for S11.	83
6.8	Distance calculations for S13.	84
8.1	Validation result (extracted information perspective).	103
8.2	Validation result (missing information perspective).	104
8.3	Validation result (extracted Concept perspective).	105
8.4	Validation result (missing Concept perspective).	106

# List of Abbreviations

CASE .....	Computer Aided Software Engineering
CUCM .....	Context Use Case Model
CUCV .....	Context Use Case Visualizer
DC-P .....	Distance Calculator-Partitioner
ECC .....	Expert-Comparable Contextual
EMF .....	Eclipse Modeling Framework
EVP .....	Eclipse Visualisation Plugin
FCSM .....	First-Cut Structural Model
FCSV .....	First-Cut Structural Visualizer
GMF .....	Graphical Modeling Framework
ISM .....	Improved Structural Model
ISMA .....	Improved Structural Model Algorithm
ISV .....	Improved Structural Visualizer
MDT .....	Model Development Tools
NLP .....	Natural Language Processing
NLU .....	Natural Language Understanding

OO ..... Object Oriented  
OOA ..... Object Oriented Analysis  
RE ..... Requirement Engineering  
READ ..... Requirements Engineering Assistance Diagnostic  
ROM ..... Recursive Object Model  
SM ..... Structural Model  
SUD ..... System Under Development  
SV ..... Structural Visualizer  
UCM ..... Use Case Model  
UML ..... Unified Modeling Language  
VDM ..... Vienna Development Method  
XML ..... Extensible Markup Language

# Chapter 1

## Introduction

Software Engineering is a systematic, disciplined, quantifiable approach which is adopted for the development, operation, and maintenance of software [18]. The focus of this thesis is on the area of Requirements Engineering (RE) - a sub-discipline which appeared within software engineering about twenty years ago to address specific challenges in the effort to gain an understanding of the nature of the engineering problem arising from stakeholders' real-world needs and the real expectations from the final software system by the stakeholders [45]. RE may be broken down into activities mainly concerned with "requirements elicitation (gathering the requirements from stakeholders), analysis (checking for consistency and completeness), specification (documenting the requirements) and validation (making sure the specified requirements meet stakeholders' expectations)" [49]. In the context of RE, comprehension of the requirements text describing a problem and its domain can typically be divided into two broad levels: the literal meaning (or surface understanding) and the interpretation (or conceptual understanding). In the context of our work, we consider



surface understanding and conceptual understanding to be the two main factors on which the quality of a text depends.

We use the term *conceptual understanding* to represent how much a developer would gain in designing or implementing a system by carefully reading/examining its problem statement only. The conceptual level involves interpretation of the document or, in other words, understanding what is meant or implied, rather than what is stated. This includes making logical links between facts or events, drawing inferences and trying to represent the content more formally. The above activities often take considerable time to perform manually, as the length of a real-life requirements document can range from a few to hundreds of pages containing numerous words, phrases and sentences, where each can potentially be wrongly interpreted. Consequently, inspecting the user requirements documents manually, in spite of being the most common way of doing so, is also one of the costliest phases of RE. A lack in domain knowledge is another important factor contributing to the difficulty of these activities because in reality, stakeholders do not usually understand the software design and development process well enough to write a comprehensive problem statement, and, at the same time, software analysts often do not understand the stakeholders' problem and field of endeavor well enough to model requirements satisfying their real needs. For this reason, different approaches have been deployed during the recent years in providing assistance to the RE process (see Chapter 3).

In this research, we provide a methodology for assisting the conceptual understanding of textual user requirements by both stakeholders and analysts. Such semi-automatic assistance can reduce the time needed for requirements elicitation and

analysis, and will help requirements engineers correctly understand the problem and, therefore, develop the right solution for it. Our approach combines two technologies: a formal graphical language called the Recursive Object Model (ROM) [50], [52] and an Expert-Comparable Contextual (ECC) model extracted from the universal data models. ROM provides a formal graphical model of the text and the knowledge it carries, and ECC is used to extract stakeholder role analogies and to investigate the completeness of the original requirements text.

The work reported in this thesis is part of a bigger project, Requirements Engineering Assistance Diagnostic (READ), aimed at improving the quality of RE documentation by applying Natural Language Processing (NLP) techniques to the RE process (see Fig.1.1).

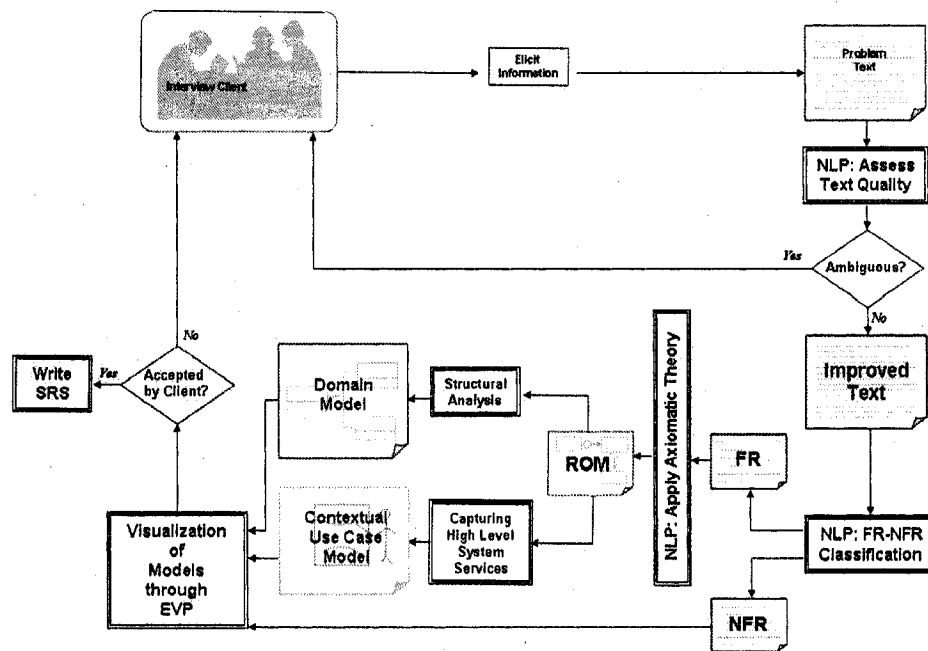


Figure 1.1: NLP-driven quality assessment in RE.

The objectives of the overall project can be expressed in terms of two main goals:

- (G1) Automatic NLP-driven quality assessment of the textual requirements in the requirements elicitation and analysis phase.
- (G2) Extraction and graphical visualization of the conceptual knowledge extracted from the requirements text for the stakeholders' validation and feedback.

This thesis is concerned with the challenges inherent in capturing the high-level conceptual view on the software system structure (Structural Model), i.e., concepts and relations between them as well as the high-level contextual view on the software system actors and services (Context Use Case Model) from the textual user requirements. The goal is to help software analysts meet the challenges that exist in correctly understanding user requirements during the interactive process of requirement elicitation and analysis, where a full and proper understanding of the application domain is considered as a fundamental success factor in the software development process.

## 1.1 Problem Statement

The problem statement of this thesis mainly stems from the gap between the stakeholders' perception of their needs and how these needs are described textually, whereby the description can be misinterpreted by developers due to the inherent ambiguity of the natural language itself. We found virtually no conceptual characteristic that would be extractable automatically from text for highlighting problems of conceptual understanding by currently available NLP tools. This may be due to the subjective nature of the conceptual understanding process, which requires expertise in software modeling. The ability to detect serious conceptual problems in the requirements text

at a very early stage of requirements analysis could help avoid very costly requirements misinterpretations.

## 1.2 Goal of the Research

The proposed methodology forms the basis of the automated process designed to capture the high-level conceptual view on the system structure (SM) and high-level contextual view on the software system's actors and services (Context Use Case Model - CUCM) from the textual user requirements. The SM of the system which shows the structure of the system in terms of the concepts and the relations between them. The CUCM is intended to serve as a basis for software Use Case Model development, and can be used by analysts in their in-depth study of requirements text (The Use Case Model is recognized as one of the models most often used in coming to an agreement on the final set of requirements [9], as well as being well known as a conventional analysis method in RE [20]).

The approach is rooted in the syntactical analysis and formalization of text written in natural language, and it is enriched with domain-related information provided by the ECC models that are extracted from reusable domain-specific data models.

Such visualization would save development time, serve as a mean to proofread the requirements, and facilitate communication between the stakeholders who provide the requirements and the software development team who have to implement them. In conclusion, the semi-automatic assistance proposed in this thesis aims at reducing the effort needed for improving requirements understanding and clarity, and

for analyzing their consistency and completeness, ultimately increasing the quality of the RE documentation.

## 1.3 Contributions

This section summarizes the expected contributions in response to the problem statement in this research.

- Devise rules for extracting CUCM and SM elements, i.e., concepts, attributes, relations, actors, summary-level use cases, and making associations between them.
- Define rules for Creating ECC models from the universal data models.
- Apply the knowledge included in the ECC to identify the potential actors for CUCM.
- Identify the key sentences characterizing high-level system services and assign the remaining sentences from the requirements text to the corresponding use case they describe, with the help of a metrics-based text partitioning algorithm.
- Investigate the completeness of the original requirements text with the aid of ECC modeling and improved their completeness by injecting the domain related missing information conveyed in ECC model.
- Develop a prototype tool in support of the methodology as a proof of concept.

## 1.4 Organization of the Thesis

The thesis is organized as follows: The context of the research and the background required to understand the methodology are covered in Chapter 2. In Chapter 3 a critique of the research and a comparison with the related work are presented. Objectives and research proposal are outlined in Chapter 4. The methodology is explained in Chapter 5 and is illustrated with a case study in Chapter 6. The architecture of the tool implementing our approach is described in Chapter 7. Chapter 8 covers the results of the validation of the methodology. In Chapter 9 we present a critical discussion on the significance and limitations of our work. Finally, in Chapter 10, conclusions are presented and directions for future work are outlined.

# Chapter 2

## Background

In this chapter, we introduce the areas of RE and NLP. We will also briefly describe the work that has been done in ROM and the previous phases of the READ project as a background knowledge on which the work in this thesis is based.

### 2.1 Requirements Engineering

We start our discussion with defining the term RE as used in literature.

Zave [19] has defined RE as “the branch of software engineering concerned with the real world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families.” As stated above, the RE is mostly concerned with real world goals which can indeed be considered as the motivation behind the development of the system. Another interesting point in the previous description is the term “precise specification” which can be

considered as a basis for analysis and validations of stakeholders' needs.

Sommerville [45] defines RE as a process that includes four high level sub-processes, namely, Feasibility Study, Requirements Elicitation and Analysis, Requirements Validation, and Requirements Management with the final goal of creating and maintaining a system requirements document. Each of these sub processes is described in greater detail in the following.

### **2.1.1 Feasibility Study**

The goal of this stage is to determine whether the stakeholders' needs are achievable considering the current software and hardware technologies and budget. Based on the final report of this stage, a decision is made whether or not to continue with the development of the system.

### **2.1.2 Requirements Elicitation and Analysis**

Once a decision is made to continue with the development of the system, the next step is the elicitation and analysis of the requirements with the purpose of clarifying what is expected from the system. This stage encompasses elicitation activities which deal with gathering requirements through discussion with the potential stakeholders, interview mock ups, prototyping, etc. Requirements elicitation and analysis can be a long and arduous process which plays a crucial role in the success of the project [2].



### **2.1.3 Requirements Validation**

Allowing errors, inconsistencies and conflicts in the user requirements document will cause them to propagate to the stages of system design and implementation, which will increase the cost of fixing and resolving them. Making sure that requirements describe what really stakeholders need is the aim of the requirement validation phase. Requirements should be validated from different aspects such as consistency, completeness, and realism. Different validation techniques such as reviews, test case generation, etc. can be used for this process.

### **2.1.4 Requirements Management**

Requirement management aims at understanding and controlling change. The stakeholders' needs may change and new requirements may appear also existing requirement may be modified, or dropped. Therefore, there should be a mechanism for managing the requirements, keeping track of the newly added or changed requirements as well as capturing and maintaining the interdependencies between the requirements. Planning for requirements management can start as early as the requirement elicitation phase.

### **2.1.5 Scope of the Thesis**

The focus of this thesis is on the requirements elicitation and analysis and to some extent validation. Other sub-processes are out of the scope of this thesis.

## 2.2 Linguistics and Language Processing

Linguistics is mainly concerned with studying human languages whereas Computational Linguistics, as a subset of both Linguistics and Computer Science, emphasizes the formalization of linguistics structures while automatically processing languages. Alternatively, NLP or Natural Language Understanding (NLU), involves building the information processing systems that are able to recognize and understand humans language in various forms and interact with humans accordingly [33, 47].

Some typical applications of the NLP include:

- Information Retrieval.
- Spelling and grammar checking.
- Automatic text summarization.
- Machine Translation.

There also exist different levels of NLP, namely:

Phonetics deals with the physical characteristics of the language like constants and vowels, and it is used in speech recognition.

Lexicon and Morphology examine different forms of the words such as singular and plural forms.

Syntax works on the possible order and arrangement of the words in a sentence or, in other words, the structure of the sentences.

Semantics studies the meaning of the words either in isolation or when combined in sentences.

Pragmatics addresses how the real world knowledge can affect the literal meaning of the words and sentences [25].

## 2.3 Applying Linguistic Analysis to RE

As pointed out previously, having a precise requirement statement is playing a key role in the success of the software project. Although great progress has been made in the area of formal methods yet it does not sound tenable to ask the stakeholders to learn and express their needs using formal methods. Therefore, majority of the requirements are still written in natural language [30] which tends to be ambiguous by nature. According to [3], writing requirements is a cooperative work between analysts and stakeholders. The stakeholders best know their needs. On the other hand, the analysts should translate the needs to the world of computing and establish the communication between these two worlds. Being capable of extracting the concepts and their relations from the requirements document is an important factor and, as systems are expanding and becoming more complex, the necessity to extract the information from large requirement documents written in natural language has emerged as a great motivation for research on the applications of NLP in RE [34]. As Ryan has concluded in [37], reviewing the history of NLP in RE makes it clear that building a system which automatically understands the stakeholders' needs is unrealistic. However, considering that conformance of the requirements to stakeholders' needs is

a dynamic and social process, NLP can be used as support for this process (not a replacement), which will assign the proper role to NLP in RE.

## 2.4 READ Project

In this section, we will briefly explain what has been done previously in the context of the READ project (see Fig. 1.1).

### 2.4.1 Automatic Ambiguity Detection in User Requirements Documents

The first phase of the READ project aimed at detecting the presence of possible ambiguities in software requirements documents during the requirements elicitation process. The objective was to identify textual ambiguities in the text before the conceptual modeling of the requirements begins. This approach relied heavily on the NLP, namely, on the usage of a text classifier that emulates human reading comprehension considering a number of quality characteristics to differentiate between ambiguous and unambiguous requirements. Using the text classifier yielded optimum results with an accuracy of 86.67% in detecting ambiguities at the level of surface understanding (as a reminder, we use the term *surface understanding* to denote how easy or how difficult it is to understand the facts stated in the document, without judging its design or implementation concerns in terms of any software engineering concept) [16].

Having the improved text, meaning the text from which the ambiguities were

removed, the formal graphical presentation of the requirement text (ROM diagram) is created. Generating the ROM diagram from the requirement text forms the second phase of the READ project.

### **2.4.2 Formal Graphical Presentation of the Requirement Text**

The purpose of this section is to introduce briefly the formal representation of the syntactic structure of text with ROM. Our choice of formal representation is justified by the fact that ROM is proved sufficient to represent technical English text used in the software engineering documents, where only statements are involved. The ROM was initially developed in [50] and further refined in [51].

#### **Mathematical Foundation**

The formal representation of the linguistic structure has been confronted with fundamental mathematical and philosophical challenges. The axiomatic theory of design modeling provides a solution on the basis of the rigorous mathematical concepts for this problem based on which ROM is developed for representing the text [5, 50, 51].

#### **Graphic Representation of Linguistic Structure with ROM**

The following graphical symbols are defined in correspondence with the axiomatic theory of design modeling to represent English verbs and nouns [51].

The ROM uses only five basic symbols to represent an object, a composite object, constraint, connection, and predicate relations.

The basic unit in the ROM is an object represented by a word surrounded by a

solid line box. This object represents a concrete entity that equals a concrete noun, an object that can be measured, or a proper noun Fig. 2.1 shows the object in ROM.



Figure 2.1: Object.

The concept of compound object ( $\oplus O$ ) is used to represent a more complex object, which is an object that includes at least two objects in it. A compound object is shown in Fig. 2.2.



Figure 2.2: Compound object.

A constraint relation ( $\xi$ ), an arrow with a solid line attached by a circle at one end as shown in Fig. 2.3, represents a descriptive, limiting or particularizing relation of one object to another. The arrow always points to the object to be constrained.



Figure 2.3: Constraint relation.

The mathematical definition for such a relation can be represented as an interaction from the constraining object  $O_i$  to the constrained object  $O_j$ :

$$\xi \subset O_i \otimes O_j$$

A predicate relation ( $\rho$ ), an arrow with a solid line (see Fig. 2.4) represents an act of an object on another object or the state of an object. A predicate relation includes

different forms, such as an action, and it corresponds to transitive or intransitive verbs in English.



Figure 2.4: Predicate relation.

The mathematical definition for such a relation can be represented as an interaction from the one object  $O_i$  to another object  $O_j$ :

$$\rho \subset O_i \otimes O_j$$

A connection relation ( $\iota$ ), a dashed arrow, shown in Fig. 2.5, is used to connect two objects that do not constrain each other. The arrow is optional, depending on the semantics of the relation.



Figure 2.5: Connection relation.

The mathematical definition for such a relation can be shown as an interaction between one object,  $O_i$ , and another object,  $O_j$ :

$$\iota \subset O_i \otimes O_j$$

Connections can be further classified into spatial, temporal, and logical. Examples of connection relations include and, or, onto, to, from-to, if, then, etc.

The Recursive Object Model Analysis (ROMA) tool transforms a text in natural language into a ROM diagram. This diagram is also stored internally in the XRD

(an extension of the Extensible Markup Language - XML) format, and can be used by various applications which are based on the formal syntactical structure of a text.

The formal ROM model of the requirements text represents its linguistic structure, and carries knowledge on the structured relations between language entities. It does not, however, offer the means to discover relations, concepts and attributes, that is, the conceptual context and high-level services of the system.

In our methodology, the formal model is employed to formalize the textual user requirements, and serves as an input to the generation of SM and CUCM.

The next chapter presents an overview of some of the most noted work in the field of applying linguistic techniques to static and dynamic analysis of the requirements text.



# **Chapter 3**

## **Related Work**

### **3.1 Introduction**

Over the past few decades, extensive studies have been conducted in the area of applying linguistic techniques to static and dynamic analysis of a requirements text. Although a wide variety of radically different case tools have been developed, common to most of these approaches are a couple of basic concepts, such as relating nouns to classes, and relating adjectives to attributes [1, 4]. In this chapter, we will review a few noteworthy studies in this area in a chronological order.

### **3.2 General Review**

The earliest study in this field is done by Abbott [1] (1983). His idea is that the terms used in the original problem description are the best candidates that can be used to write a program that solves the problem. He proposes a technique for creating

programs which is extracting different data types from proper nouns or common nouns and operators from verbs or predicates that match the informal but precise English descriptions. His major contribution is identifying the relationship between data types and common nouns. His target programming language is Ada. His idea is further developed by Booch who introduces an Object Oriented (OO) design model [4] (1986). In his proposed model nouns suggest classes and verbs suggest operation.

Saeki *et al.* (1989) [41] propose a process of evolving a formal specification from an informal specification in natural language. Their methodology is consisted of two activities: “design” and “elaborate”. In the “design” activities, nouns and verbs are extracted from the requirements document and assigned to software concepts and relations. In their approach nouns and verbs are divided into different categories such as class nouns or value nouns, and action verbs or relational verbs. They emphasized more on verb patterns that occur more often to make the approach more applicable to dynamic systems. In the “elaborate” activities, they are rewriting the informal sentences more precisely. They do not intend to develop a tool to be used for transforming natural language text to formal specification automatically using NLP. Their aim is to create software module design documents using OO concepts that are extracted from the natural language requirements documents. Although the tool is able to extract verbs and nouns automatically, it cannot detect the importance of the words and, as stated clearly in their paper, from this activity emerges the need of participation of a knowledgeable human being.

Meziane *et al.* (1986) [28] introduce the system that obtains a set of logical form expressions from translating an informal natural language requirements text. These

expressions are then used as a basis for producing the entity-relationship models and from there the system can extract the formal data types and generate the specification in the Vienna Development Method (VDM). The major drawback of this method is that not only it needed great amount of involvement from analysts but it needs analysts who are very knowledgeable in formal methods for it to be able to produce acceptable results.

Mich L. (1996) [29] propose a CASE tool called NL-OOPS which is capable of analyzing requirements and creating OO models. The architecture of the system is divided into three parts. The first part contains all NLP activities and is based on system called Large scale Object based Linguistic Interactor Translator Analyzer (LOLITA). The second part deals with defining coherent requirements set, and the last part is concerned with the OO analysis itself. The OO module implements an algorithm that extracts the objects and their associations. Two main issues are remaining unsolved in this approach. The first problem is detecting ambiguities, inconsistencies, and omissions from the requirement text and the second problem is the size of the requirement document. In the larger documents the selection of the relevant objects becomes a concern.

Moreno *et al.* (1997) [31] propose a methodology for formalizing the Object Oriented Analysis (OOA) process in order to reduce its immaturity. The aim of this method is to analyze the informal specifications, extract the components of the OO system such as classes, present the models (OO class diagrams) graphically, and make this process as independent as possible for analysts' domain knowledge. The major drawback of the method is that it cannot deal with incoherence, ambiguity, inconsis-

tency, and other deficiencies in the requirements specification. Therefore, it assumes that the requirements document is correct.

Recently, many researches have attempted to deal with unrestricted natural language using the progress that has been made in NLP. Their major goal is generating static and dynamic view of the System Under Development (SUD). One interesting tool, that of Harmain *et al.* 2000 [14], called CM-Builder, is a CASE tool which performs domain independent OO analysis. The input to the system is a software requirement document and the output is a set of candidate classes and their relationships. The output is in a format called CDIF (CASE Data Interchange Format) which can be directly fed to a graphical CASE tool like Select Enterprise Modeler for further refinements. The shortcomings of this approach are: not being integrated to the powerful graphical CASE tool designed specifically to support software analysis, not supporting any kind of dynamic diagrams as well as being useful for small corpus of case studies.

Another tool, LIDA, is introduced by Overmyer (2001) [35] with the goal of helping analysts in transition from natural language text to OO notation. LIDA is a prototype tool that provides linguistic assistance in development process. It helps analysts develop OO models of a domain, using a subset of the Unified Modeling Language (UML). The input to the tool could be textual domain descriptions, operational concept documents and use cases, and the output is a model consisting of elements, their relations and attributes that would be finally mapped to a UML model by an analyst. However, the text analysis and generation of the models remains in good part a manual process that can be cumbersome with complex texts. LIDA does not

support dynamic diagrams and it requires considerable user interventions.

B. S. Lee *et al.* (2002) [22] develop a system in which there is a mapping from a natural language requirement document to the OO formal specification language called VDM++, an OO extension of the Vienna Development Method. The aim of this tool is to provide the mapping from natural language to OO formal specification language. The drawback of this tool is being limited to only a two-level grammar.

For their part, H.G. Perez-Gonzalez *et al.* (2003), present GOOAL (Graphical Object Oriented Analysis Laboratory) [36]. The goal is to produce static and dynamic object models from natural language document automatically. Original problem statements are automatically translated to semi-natural language called 4W. The produced sentences then are analyzed with role POSets which are based on the mathematical concept of partially ordered set which are used widely in many linguistics formalisms to construct the necessary structures that can generate static and dynamic views of the system from a problem written in 4W. After that the interpretation in a 4W language is shown by the system and validated by the user. GOOAL can produce a dynamic behavioral diagram. The drawback is that the system has been tested with problems described in no more than eight sentences and 100 words on an average.

Lui *et al.* (2003) [24] propose a tool for automatic objects/class identification. The goal of this tool is to automate the transition from requirements to the detail design. The case tool is developed as one of the add-ins of rational rose. This tool has two main functionalities: 1) use case realization and 2) class diagram generation. However, there are no further information regarding the performance of the tool and the maximum size of the corpus they have used for testing their system. For their part

Drazanet *al.* [10], suggest a method for processing textual use cases and extracting their behavioral aspects based on linguistic techniques namely parts of speech of the words.

Finally, Some [44] presents a tool called UCed with the aim of providing the framework for use case editing, clarification and finally developing the “executable specification integrating the partial behaviors of the use cases” in the form of state machines. UCed provides means for capturing the use case descriptions but does not assist the developer in the use case model’s elicitation process.

The work reported in this thesis differs considerably from the related work in that our methodology is founded on a formal syntactical representation of the text and not on the parts-of-speech technique [10, 24], which allows for an automatic and thus objective elicitation of SM and CUCM elicitation from text. This approach would allow us to reduce the number of costly human errors in the RE process. Moreover, we bring an automatic and objective expert knowledge into the elicitation process by introducing the ECC models extracted from domain-specific data models.

In the next section we discuss in detail three of the research publications that represent the most important work in this area and are relevant to the research proposed in this thesis.

## **3.3 Overview of Most Important Related Work**

### **3.3.1 NL-OOPS**

#### **Summary**

The goal of the NL-OOPS tool [29] is to generate an OO model from the requirement document written in natural language. The research on the NL-OOPS started from the Semantic Net (SemNet) produced by the LOLITA, an NLP-based CASE tool which is the core of NL-OOPS. The SemNet serves as a knowledge base of the LOLITA storing knowledge that can be accessed, modified or extended. The OO analysis algorithm is implemented using database primitives for refining the nodes contained in the SemNet and identifying the classes, attributes, and associations required to construct the OO model. The final Object Model is visualized using the existing package Graphed [15].

#### **Critical Discussion on NL-OOPS**

- NL-OOPS ignores possible ambiguities and inconsistencies in the text.
- As the size of the text increases, the semantic net becomes very complicated and thus its readability decreases.
- It filters the potential concepts according to their frequencies, which can cause loss of information.

### 3.3.2 LIDA

#### Summary

Overmyer *et al.* [35] develop a prototype tool which addresses linguistic assistance in conceptual model development. They propose a methodology in which a tagger is used to assign parts of speech to the words in the requirement text and classify them as nouns, adjectives, and verbs. The list is used by analysts iteratively for identifying the potential classes, attributes, and relationships between the concepts and other unnecessary information that are omitted from the list. Normally, the words with highest frequencies are chosen as potential classes. New classes, attributes, and adjectives are associated to each other graphically using the editor called Modeler.

#### Critical Discussion on LIDA

- LIDA does not generate the models automatically. It classifies the words in the sentences as nouns, verbs, and adjectives and it is the responsibility of analysts to associate these nouns to classes, attributes, and possible relationships. The tool provides a modeling environment that can be used by analysts to graphically visualize these models.
- Analysts have to identify the synonym nouns that are used in the text as well as the proper adjectives that can be used as the attributes.
- LIDA can not hold the accurate interrelation between the extracted information and show the related element in the Modeler. For example, the only criteria to decide whether adjective X should be an attribute of concept A is their



proximity in the text.

- The models should be exported to Visio (a diagramming software) in order for the analysts to edit them.
- The authors validate their models using the texts that are generated from the model using LIDA Modeler.
- The system relies heavily on the knowledge of the analyst.
- LIDA does not support any kind of dynamic models.

### **3.3.3 CM-Builder**

#### **Summary**

The Class Model (CM) Builder is a CASE tool which can generate the conceptual models from the requirement text through linguistic analysis. The NLP-modules of the tool called LaSIE (Large Scale Information Extraction) form the core of the CM-Builder and the input text will go through lexical processing, morphological analysis and semantic interpretation sentence by sentence and, finally, the discourse model are produced. The next step is centered on OO analysis and an algorithm is implemented to extract the classes, attributes, and relationships. The final conceptual model is represented in the CDIF format and is fed to the graphical case tool called Select Enterprise Modeler for further refinement.

## Critical Discussion on CM-Builder

- The system itself was incapable of visualizing the model graphically. The final output of the system was a CDIF file that should be exported to a graphical case tool.
- The CM-Builder did not support any kind of dynamic models.
- Identified nouns with low frequencies were discarded which could cause loss of important information.
- The generated models had some degree of over-specification.
- The discourse model was built and never used because their analysis was limited to the domain independent analysis.

The details of our proposed methodology are described in the next chapter.

# Chapter 4

## Proposal

### 4.1 Objective

The objective of this research is to generate semi-automatically from the textual user requirements and graphically visualize a SM of the software system domain concepts and their relations, as well as a high-level contextual view on the software system's actors and services (CUCM). The importance and benefits of such models of the system to be developed include saving development time, providing the means to proofread requirements and facilitate communication between the stakeholders who provide the requirements and the software analysts who are responsible for detecting omissions and/or conceptual errors in the text. Furthermore, we intend to develop a tool as a proof of concept in support of our methodology.

## 4.2 Summary of the Research Methodology

As described in Chapter 2, Section 2.4, in the first phase of the READ project ambiguities are detected and removed from the textual user requirements. In the second phase of the READ project, the formal ROM model is generated from the improved requirements text. In our methodology, the formal ROM model serves as an input to the generation of SM and CUCM.

Fig. 4.1 summarizes the proposed methodology, which consists of 3 phases:

- (1) Generating the First-Cut Structural Model (FCSM) (see Chapter 5, Section 5.2.2, First-Cut Structural Model)
- (2) Generating the Improved Structural Model (ISM) (see Chapter 5, Section 5.2.2, Improved Structural Model)
- (3) Capturing high-level system services and generating Context Use Case Model (CUCM) (see Chapter 5, Section 5.2.3)

In the first phase, concepts, attributes, and relations are extracted from the ROM presentation of the text by applying set of heuristics developed to map the ROM presentation to the SM. The output of this phase is FCSM which is the early sketch of the conceptualization of the System Under Development (SUD).

Second phase starts with constructing the Expert Comparable Contextual (ECC) model which is a visual representation of the noteworthy concepts of the domain, their attributes and the relation between them. ECC is created from the pre-built domain specific data model according to the predefined set of heuristics. Once FCSM

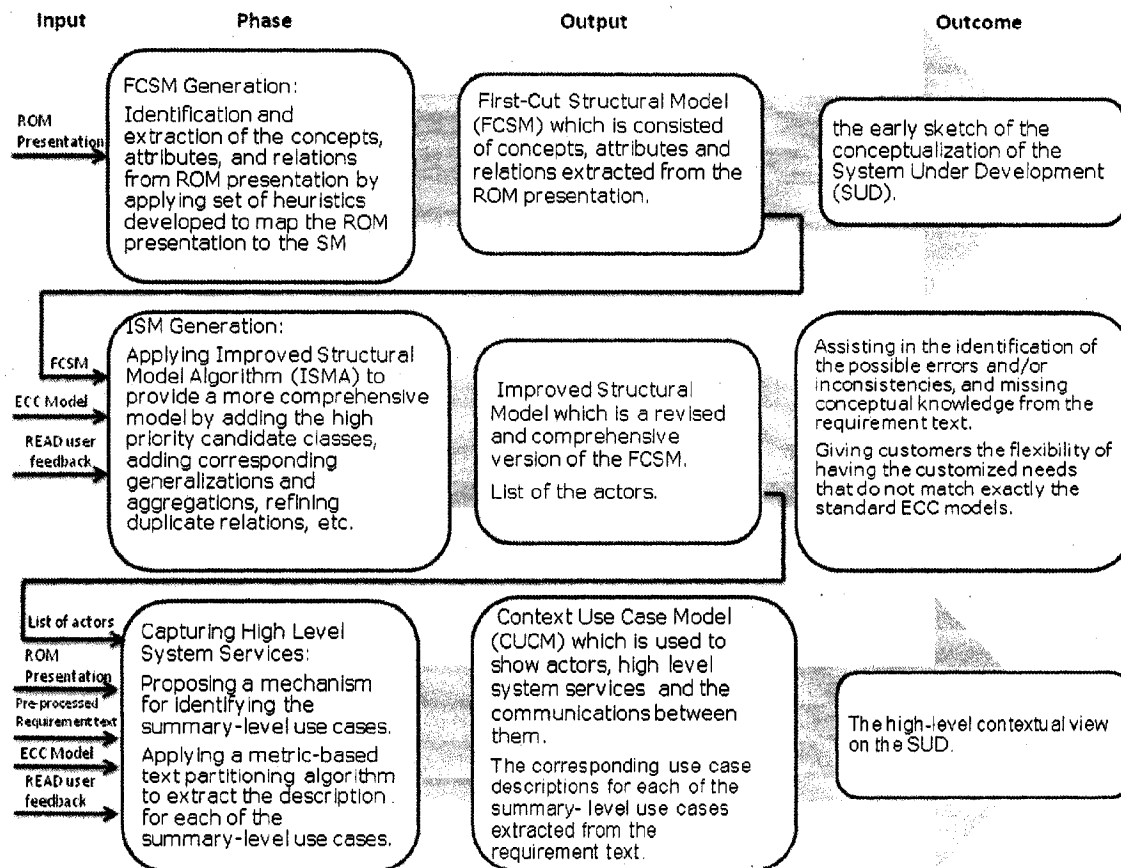


Figure 4.1: Summary of the research methodology.

and the ECC model are created, the next step is to employ them in the Improved Structural Model Algorithm (ISMA) for generating a more comprehensive model by adding the high priority candidate classes, adding corresponding generalizations and aggregations, refining duplicate relations, etc. The output of this phase is ISM which is a revised and comprehensive version of the FCSM.

The rationale behind the ISM hinges on providing both a more comprehensive model, assisting in the identification of the possible errors/inconsistencies and missing conceptual knowledge in the text and giving stakeholders the flexibility of having the customized needs that do not match exactly the standard domain models (ECC models).

Finally, the last phase of the methodology focuses on capturing summary-level use cases which is defined as high-level services provided by the system to the user, and applying a metric-based text partitioning algorithm to extract the description for each of the summary-level use cases from the preprocessed requirement text. The output of this phase is the CUCM which is used to show the actors, summary-level use case and the communications between them as well as the corresponding summary-level use case descriptions extracted from the requirements text.

Having presented the objective of the research and the proposal, in the next chapter we describe the details of our methodology.

# Chapter 5

## Research Methodology

### 5.1 Hypothesis

Our hypothesis states that a formal presentation of the requirements text, in addition to the Expert Comparable Contextual (ECC) models which are considered as a standard models of domain and are extracted from reusable universal data models, would provide sufficient information to automatically mine the conceptual knowledge required for the generation of the SM and CUCM from the text.

### 5.2 Methodology

This section describes an elaborate methodology that constitutes a proof of concept for the idea that a conceptual knowledge on the software to be developed can be acquired through a semi-automated process with textual requirement documents as input and diagrams representing SM and CUCM as outputs.

Our methodology is divided into two parts. Using the formal presentation of the text as a basis, the first part concentrates on the structural analysis and the second part focuses on capturing the high-level system services from the user requirements text.

### **5.2.1 Preprocessing of the Text**

The preliminary step of the methodology consists of preprocessing the textual input which means removing ambiguities from it with the help of the stakeholder [16] and representing its structure formally and graphically using ROM methodology [51].

### **5.2.2 Structural Analysis**

The main goal of this stage is to identify the essential elements for visualizing the text from the conceptual point of view summarized as candidate concepts, their attributes, and relationships between them.


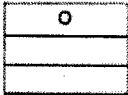
Structural analysis is performed at two different levels which are described in details in the following paragraphs:

#### **First-Cut Structural Model**

In order to map the text to its equivalent SM, with the aim of making this transformation performed semi-automatically, there is a need to develop a comprehensive set of heuristics that should be applied to the formal ROM presentation. The output of this step is called First-Cut Structural Model (FCSM) which is in fact one early sketch of the conceptualization of the System Under Development (SUD).



Table 5.1: Graphical presentation of Rule 1.

Formal presentation	Equivalent SM
	

The patterns that are likely to happen in the formal presentation of text, their equivalent FCSM and the corresponding rules are described in detail in the following paragraphs. These rules are flexible and new rules consistent with the extracted rules can be added when necessary.


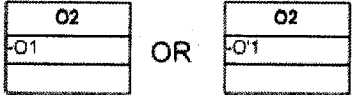
### Mapping Rules

**Rule 1** Each solid box can be considered either as a concept or an attribute for another concept in the SM. However, not all boxes are valid concepts in the domain so later they will be refined according to the defined rules. Table 5.1 is used to show the object in ROM and its equivalent in SM.

**Rule 2.1** The constraint relation (O1+O2) as shown in Fig. 2.3: If O1 is an adverb or an auxiliary verb, it will be ignored but if it is an adjective, it can be a candidate attribute for O2 or imply that there exist an attribute for O2 which corresponds to the adjective O1. In FCSM all adjectives are highlighted for the READ user in order to make sure that the attributes that should be tracked will not be lost. Later they can be substituted by the appropriate attributes in consultation with WordNet [13]. It should be noted that substitution should be done manually.

Example: In the phrase '*the large (O1) book (O2).*' the adjective *large* may

Table 5.2: Graphical presentation of Rule 2.1.

Formal presentation	Equivalent SM
	

indicate that *Book* may have an attribute called *size* or *large* itself can be an attribute for the *Book*.

**Rule 2.2** If  $O1 = \text{noun}$  and  $O2 = \text{noun}$ ,  $O2$  can be considered as an attribute for  $O1$  if  $O2$  is found in the predefined list of keywords. This list contains the words that are usually used as attributes in the specific domains and are extracted using [21], [43], and other available academic projects. Table 5.3 shows the current list. This list can be updated and expanded.

Example: In the phrase '*customer (O1) name (O2)*', the word *name* can be considered as an attribute for *Customer*.


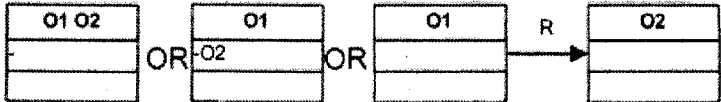
**Rule 2.3** If  $O1 = \text{noun}$  and  $O2 = \text{noun}$  and  $O2$  is not in the list of predefined keywords, we calculate the frequency of the appearance of the  $O1+O2$  in the requirement text.

In the case where  $O1+O2$  occurs once, if  $O1$  is already identified as a concept from the text (implying that it also appears without  $O2$  in the text) or if both  $O1$  and  $O2$  have been identified as concepts in the text, then  $O1$  will be associated to  $O2$  as a *Has* relation. The proposed association will be shown to the READ user for final approval.

Table 5.3: Attribute keywords.

Attribute Keywords		
Username	password	serial number
ID	reference number	sequential ID
time	name	comment
message	date	picture
description	address	phone number
image	Quantity	amount
information	detail	pin number
price	status	

Table 5.4: Graphical presentation of Rules 2.2 and 2.3.

Formal presentation	Equivalent SM
	

Example: In the phrase '*customer(O1) receipt(O2)*', where only *Customer* or both *Customer* and *Receipt* have been identified as concepts (from the other sentences of the text), these two concepts will be associated to each other with a *Has* relation.

If O2 is already identified as a concept from the text, O1O2 will be presented to the READ user as a potential single concept.

Example: In the phrase '*purchase(O1) order(O2)*', where *Order* is already identified as a concept, the phrase *Purchase(O1) Order(O2)* will be presented to the READ user as a representative of a single concept.

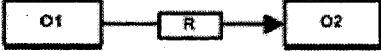
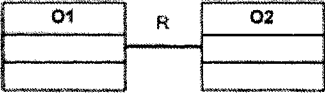
If neither O1 nor O2 are identified as a concept from the text, then all the options (attribute, *Has* relation, single concept) will be provided so that the READ user can make a proper choice.

In the case where O1+O2 is repeated more than once in the requirement text or if O1+O2 is a proper noun (e.g., *Invoicing System*), then O1O2 can be considered as a single concept.

**Note 1.** We may encounter the cases where more than two nouns ( $N_1, \dots, N_n$ ), where  $n > 2$ , are in the sequence. In that case:

- (1) If the sequence is identified as a proper noun then it represents a single concept (e.g., *Invoicing Order System*).
- (2) If  $N_1, \dots, N_{n-1}$  are identified as a single concept then the whole sequence is constraining the last noun  $N_n$  (e.g., *purchase order details* in a text where *Purchase Order* is identified as a single concept).
- (3) In all other cases each of the nouns is constraining the last noun in a sequence

Table 5.5: Graphical presentation of Rule 3.

Formal presentation	Equivalent SM
	

$(N_n)$ .

Tables 5.4 is used to present different cases of the constraint relation and their equivalents in SM.

**Rule 3** The sentence pattern  $O1+R+O2$ : If  $R$ =transitive verb or a verbal (e.g., *conform to*), then  $O1$  is in relation with  $O2$  and the relation between them will be  $R$ :  $R(O1, O2)$  (see Table 5.5).

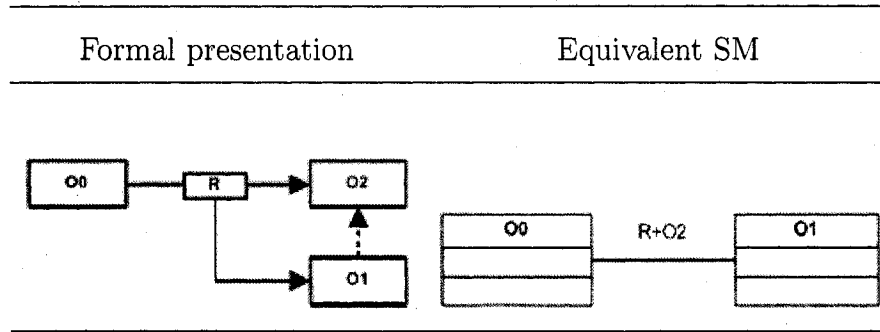
Example: In the sentence '*CBMSys(O1) creates(R) the order(O2)*.' *CBMSys* is in relation with *Order* and the relation between them is *Create*: *Create(CBMSys, Order)*.

**Rule 4** The sentence pattern  $O0+R+O1+O2$ : If  $R$ =transitive verb, then  $O0$  is in relation with  $O1$  and the relation between them is  $(R+O2)$ :  $R+O2(O0, O1)$  (see Table 5.6).

Example: In the sentence ('*System(O0) send(R) publisher(O1) an invoice(O2)*'), *System* is in relation with *Publisher* and the relation between them is (*Send Invoice*): *Send Invoice (System, Publisher)*.

**Rule 5** The sentence pattern  $O1+R+O2$ : If  $R$ =linking verb, according to the patterns described in [51], the syntactic rule of  $O2$  will be the subject complement

Table 5.6: Graphical presentation of Rule 4.



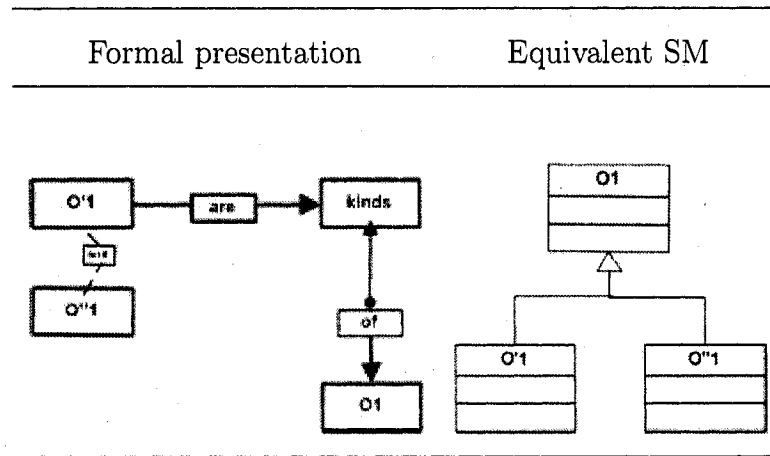
of the sentence. If the subject complement (O2) is an adjective, then the sentence is used to identify the corresponding concepts only and if the subject complement (O2) is a noun, then O1 is in relation with O2 and the relation is R: R (O1, O2).

Example: In the sentence ('connection (O1) is(R) closed (O2).'), because *closed* is an adjective, only the concept *Connection* is identified. However, in the sentence ('user (O1) is(R) an engineer (O2).'), the concepts *User* and *Engineer* and the relation *Is* between them are identified: *Is (User, Engineer)*.

*Note 1.* ROM is also capable of identifying the if clauses in the requirement text. If the if clause pattern is O1+R+O2 in which O2 is an adjective, the if clause is used to identify the corresponding concepts and/or attributes and the sentence in the main clause is used to extract concepts, relations, and attributes. This approach will help to illustrate the relations that exist only under certain circumstances or conditions.

Example: 'if customer's credit record is good, then the system accepts the order.' From the if clause we extract the *Customer* and *Credit Record* concepts and from the main clause we extract *System*, *Order*, and the relation *Accept*: *Accept (System, Order)*. In other cases, both if and main clauses are used to extract concepts, relations,

Table 5.7: Graphical presentation of *Is kind of*.



and attributes. We also use the if clause as a note for the relation identified from the main clause relation. Therefore, we can track the condition under which the relation exists between the concepts.

*Note 2.* For the sentence pattern described above, if  $R = \text{Is kind of/are kinds of}$ , then  $O2$  is the parent of  $O1$  and the relation between them will be generalization (see Table 5.7).

**Example:** In the sentence '*Saving Account and Checking Account are kinds of accounts.*', *Account* is a general concept (parent) for *Saving Account* and *Checking Account*.

**Rule 6** The sentence pattern  $O1 + R$ : If  $R = \text{Intransitive verb}$ , then the concept  $O1$  has a self association,  $R$  (see Table 5.8).

**Example:** In the sentence ('*System( $O1$ ) stops( $R$ ).*'), the concept *System* has a self association *Stop*.

**Rule 7** The prepositional relation to, for, from: The pattern of the original sentence as it appears in the requirement text should be checked. There are three

Table 5.8: Graphical presentation of Rule 6.


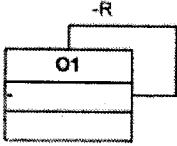
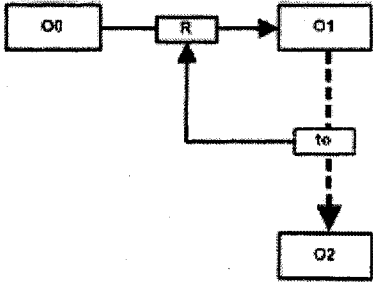
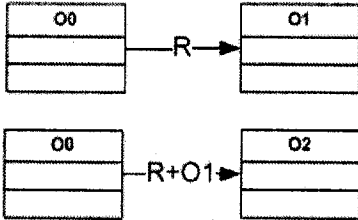
Formal presentation	Equivalent SM
	

Table 5.9: Graphical presentation of Rule 7.1.

Formal presentation	Equivalent SM
	

cases to consider:

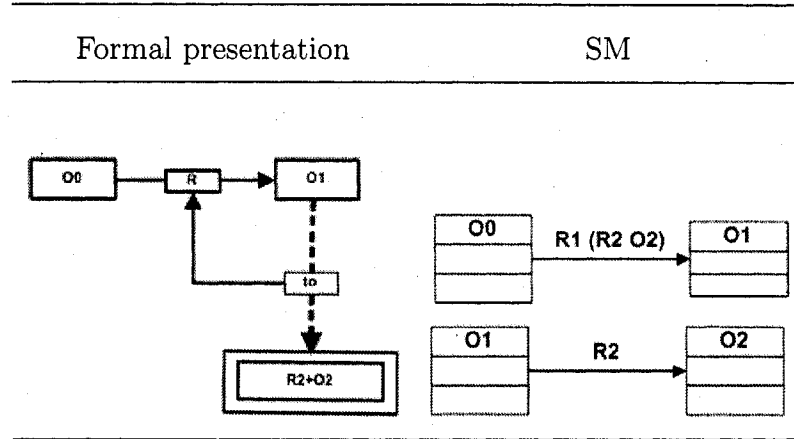
**Rule 7.1** The Sentence pattern  $O0+R+O1+to/for/from+O2$ : If  $R=transitive$  verb, then  $O0$  is in relation with  $O2$  and the relation is  $(R+O1)$  and  $O0$  is in relation with  $O1$  and the relation will be  $R$ :  $R+O1 (O0, O2); R(O0, O1)$  (see Table 5.9).

Example 1: In the sentence '*System (O0) sends(R) invoice (O1) to the customer (O2).*', there is a relation between the *System* and the *Invoice* concepts and that relation is *Send*: *Send (System, Invoice)*. Also there is a relation between the *System* and the *Customer* which is *Send Invoice*: *Send Invoice (System, Customer)*.

Example 2: In the sentence '*Customer(O0) withdraws(R) cash(O1) from ATM(O2)*',



Table 5.10: Graphical presentation of Rule 7.2.




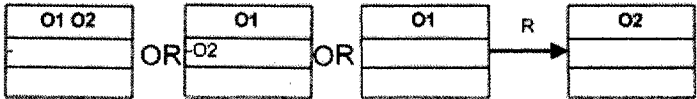
there is a relation between the *Customer* and the *ATM* and that relation is *Withdraw Cash*: *Withdraw Cash (Customer, ATM)*. Also there is a relation between the *Customer* and *Cash* which is *Withdraw*: *Withdraw (Customer, Cash)*.

**Note 1.** For the same sentence, if O1 is defined as one of the attributes in the predefined list of keywords (see Table 5.3) and it is being constrained by another object such as O3 then we only consider the relation between O0 and O2, and the relation will be (R+O3O1): R+O3O1 (O0, O2).

Example: In the sentence '*system(O0) shows(R) order (O3) status(O1) to the customer(O2).*', where *Order (O3)* is constraining *status (O0)* (see Fig. 2.3), there is a relation between the *System* and the *Customer*, and that relation is *Show Order Status*: *Show Order Status (System, Customer)*.

**Rule 7.2** The Sentence pattern O0+R1+O1+to/for+R2+O2: If R=transitive verb, then O1 is in relation with O2, and the relation is R2: R2 (O1, O2) and also O0 is in relation with O1, and the relation is R1(R2-O2): R1+R2O2 (O0, O1) (see Table 5.10).

Table 5.11: Graphical presentation of Rule 8.

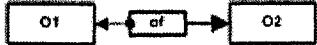
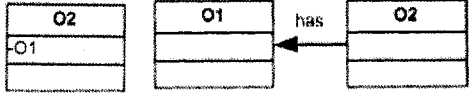
Formal presentation	Equivalent SM
	

Example: In the sentence '*Customer(O0) request(R1) the system(O1) to update(R2) the stock(O2).*' There is a relation between the *Customer* and the *System* and that relation is *Request (Update Stock): Request Update Stock (Customer, System)* and there is another relation between the *System* and *Stock* which is *Update: Update (System, Stock)*.

**Rule 8** O1=noun, O2=noun, and O1's O2: If O2 is among the keywords, then O2 is the attribute for O1, else there exist a *Has* relation between O1 and O2 (see Table 5.11).

**Rule 9** Prepositional relation of in the phrase O1+of+O2. If O1 is in the predefined list of keywords (see Table 5.3), then O1 is the attribute for O2, otherwise, O2 and O1 are related to each other and the relation is *Has* (see Table 5.12).

Table 5.12: Graphical presentation of Rule 9.

Formal Presentation	Equivalent SM
	

Next, the formalization process and a sample of the rules explained above are illustrated on a very simple example.

Given the following sentence in plain English for which the parts of speech have been assigned to each word, after applying the axiomatic theory of design modeling we will obtain the ROM diagram shown in Fig. 5.1 and by applying Rule 6.1 we obtain the corresponding SM shown in Fig. 5.2.

System (NN) sends (VBZ) invoice (NN) to (PP) the (DT) customer (NN).

Tables 5.13 demonstrates the result of the syntactic analysis for the above sentence.

Table 5.13: Syntactical analysis for the sample sentence.

nsubj(sends-2, System-1)
dobj(sends-2, invoice-3)
prep(sends-2, to-4)
det(customer-6, the-5)
pobj(to-4, customer-6)

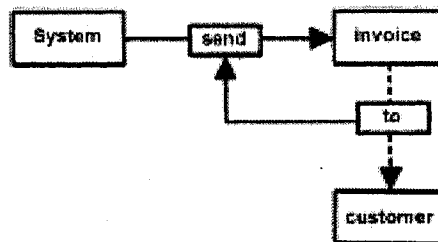


Figure 5.1: ROM diagram.

Our next goal is improving the FCSM. Here, an improvement stands for adding

domain-related expert knowledge to the FCSM as well as detecting and resolving some of the possible incompleteness or inconsistencies that may exist in the model. The following section introduces the ECC models (the related background knowledge) and further describes the process of obtaining the Improved-Structural Model (ISM) using both the ECC and the FCSM.

### **Expert Comparable Contextual (ECC) Model**


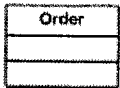
Generally, models are developed to achieve a particular goal and to highlight the important features of something, considering that specific goal [48]. While reviewing the literature, we came across interesting evidence which supports the resemblance between the entities in the data models and the conceptual classes in domain models. For example, in [21], reuse of the universal data models has been stated as one of the strategies for identifying conceptual classes. Data Models describe the structure of the data as well as their meaning, and data modeling is recognized as a standard for designing databases [43].

Our analysis revealed that there is great potential for using data models as resources for identifying the necessary elements of the conceptual models due to their resemblance.

We intend to construct a comprehensive model for specific domains and further investigate their applicability to improve the structural model developed in the previous step. Furthermore, we want to contemplate the potential of these models for assistance in automatic generation of the CUCM of the system.

We have named these models ECC models and their derivation process is outlined

Table 5.14: ECC concept.

Entity	Concept
	

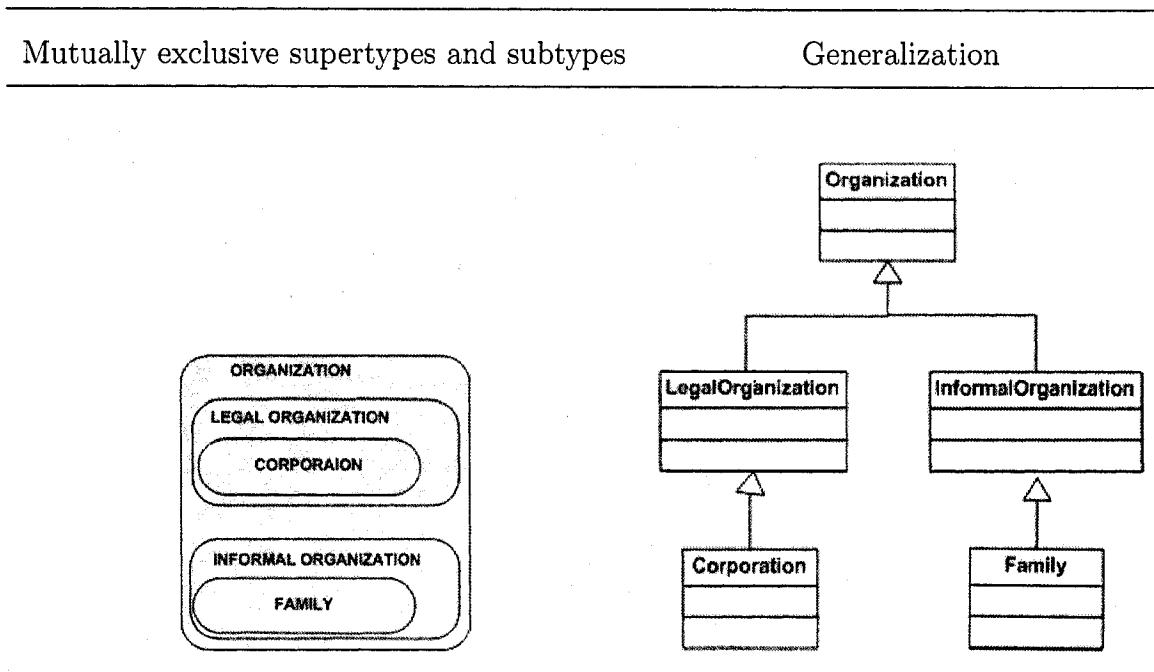
below:

- Acquisition of the standards and conventions used in data modeling.
- Defining the list of heuristics that can be used for mapping data models to the ECC model.
- Developing the representative domain concepts for each of the entities in the data models.
- Developing a comprehensive model that defines the interrelation between the concepts.

Following is the description of the transformation rules which are used for mapping data models to ECC models. Each table is composed of two columns. The left column presents one of the naming standards and diagramming conventions used in the data models and the right column shows the corresponding name and diagram in the ECC model.

**Rule 1** Each entity in the data modeling can be simply replaced by the corresponding concept in the SM (see Table 5.14).

Table 5.15: ECC generalization.



**Rule 2** Mutually exclusive supertypes and subtypes are mapped to the generalization (see Table 5.15).

For example: An organization can be either legal or informal. Furthermore, a legal organization can be a corporation.

**Rule 3** Non-mutually exclusive supertypes and subtypes are mapped to the specific generalization that includes conventions for illustrating all possible combinations (see Table 5.16).

For example: A requirement can be customer, product or work requirement. From another perspective, the same customer requirement can also be product or work requirement.

Table 5.17 shows the conventions used to illustrate the mutually exclusive and mutually inclusive concepts.

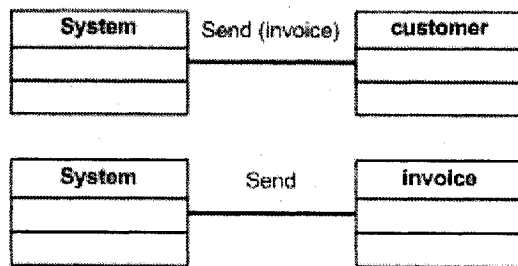


Figure 5.2: SM for the sample sentence.

Table 5.16: ECC generalization with all combination.

Non mutually exclusive super types	Generalization with all combinations and subtypes
------------------------------------	---

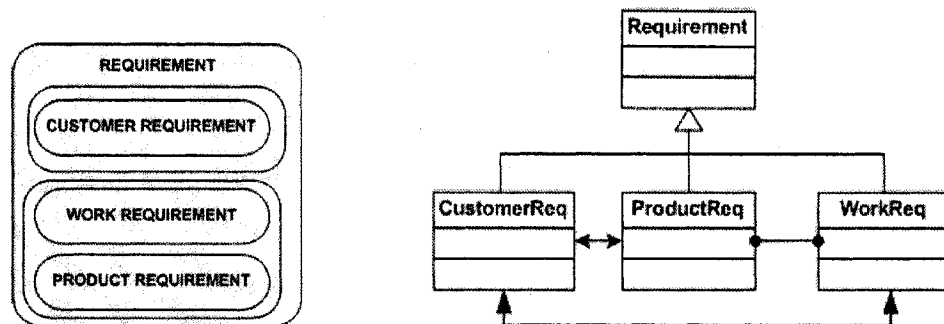


Table 5.17: Conventions used for mutually inclusive and mutually exclusive concepts.

Mutually exclusive concepts	Mutually inclusive concepts
-----------------------------	-----------------------------

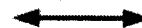
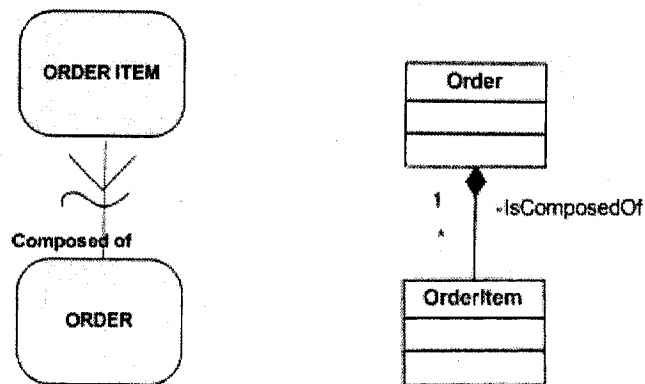


Table 5.18: ECC attributes.

Attribute(s)
<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; border-radius: 10px; padding: 5px; text-align: center;"> <b>ORDER</b>  ORDER ID  ORDER DATE  ENTRY DATE </div> <div style="border: 1px solid black; padding: 5px;"> <b>Order</b>  Order ID  Order Date  Entry Date </div> </div>

Table 5.19: ECC aggregation/composition.

One-to-Many (Composed of)	Aggregation or Composition
---------------------------	----------------------------



**Rule 4** Each Attribute in the data modeling can be directly transferred to the ECC (see Table 5.18).

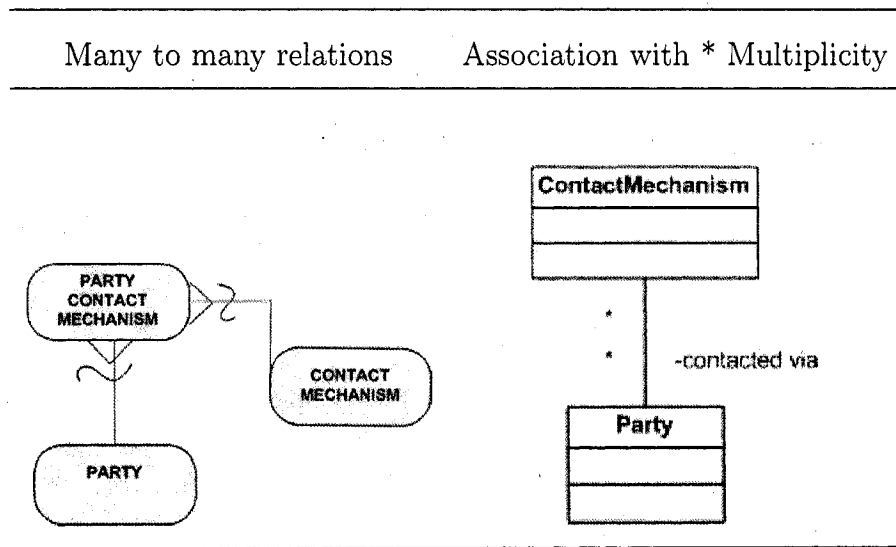
**Rule 5** A one to many relation (composed of, consisted of) can simply be replaced by aggregation or composition (see Table 5.19).

For example: *Each order consists of few order items.* Order can be considered as an aggregation of order items.

**Rule 6** Many-to-many relations can simply be replaced by association with \*(Many)



Table 5.20: ECC association with \* multiplicity.



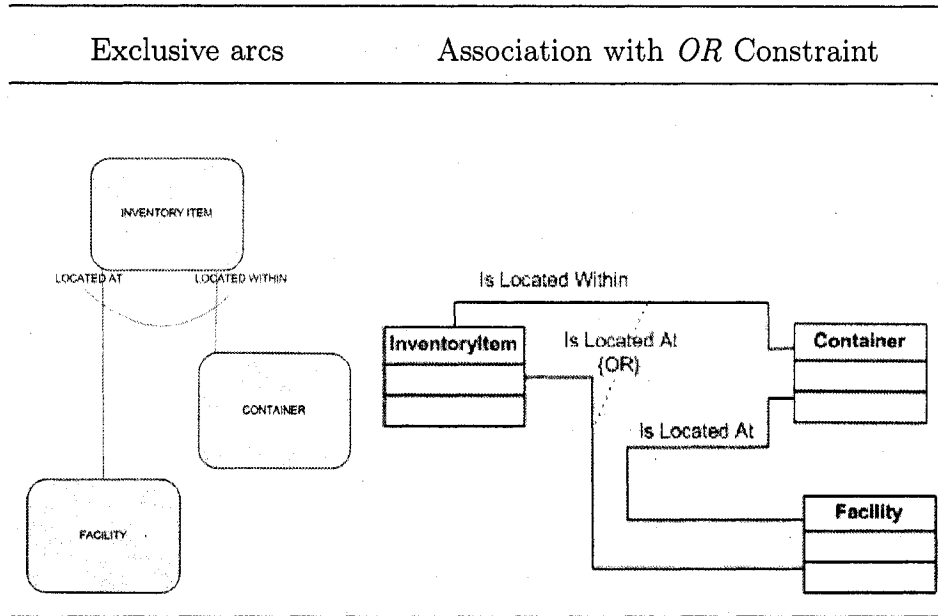
multiplicity. The intersection entity can be removed because the intersection entity is an entity that is created to breakdown the many-to-many relationships when they are normalized (see Table 5.20).

For example: *'A party may have more than one contact mechanism such as postal and telecommunication and, on the other hand, many people may have the same contact mechanism (e.g., the same working address).'* We can remove the party contact mechanism entity and use an association directly between party and contact mechanism.

It should be noted that assigning multiplicities to the relations is out of the scope of this thesis.

**Rule 7** Exclusive arcs are used to represent group of relations that cannot exist together and at each time only one of them exists. In the ECC model they are replaced by associations with *OR Constraint* (see Table 5.21), for example:

Table 5.21: ECC association with *OR* constraint.



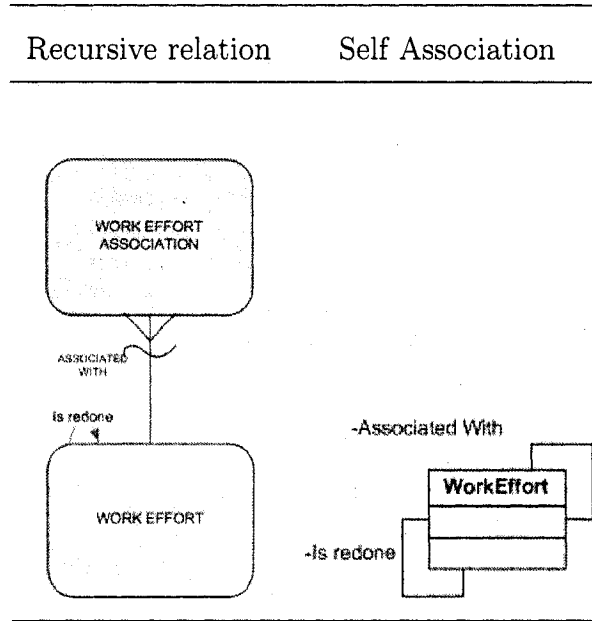
*'Each inventory item can be either located at the facility or located within container.'*

**Rule 8** Recursive relations show that the entity is in relation with itself. There are two kinds of recursive relations: i) one-to-many which is pointed from the concept to itself, and ii) many-to-many relation which is showed with the help of another entity used as an intersection. Both of the recursive relations are mapped to the self association in the ECC model (see Table 5.22)

For example: *'Each work effort is related to itself to show that the work effort should be redone. Also it is related to another entity, work effort association meaning each work effort may be dependent to another or broken down into several lower-level work efforts.'*

In our approach, we use a concise form of these models by extracting the fewest concepts necessary to form the basis of the specific domain. This provides a domain

Table 5.22: ECC self association.



model which can serve the stakeholders' various needs at different times. By this we mean that it is essential to extract only the expert knowledge required for the conceptual modeling of the domain and to reduce the number of details pertaining to the physical data model instances and the persistent storage.

As suggested in [43], some of the constructs applicable to most organizations are: people and organizations, products, ordering, shipping, work effort, invoicing, accounting and budgeting, and human resources. Currently we have investigated the Invoicing Orders domain and the derivation process of ECC is manual. We leave the rest of the constructs to be explored in future. Fig. 5.3 shows the ECC model for the invoicing orders domain.

While building the ECC model for each of the mentioned constructs requires certain effort, once these models are built they can be reused and their real benefit

will become evident in the long term.

The influence of these models on the improvement and completeness of the SM, such as useful information added about the relationships between concepts (e.g., generalization and composition), will be discussed in the following section. In section 5.2.3, we focus on the use of ECC model in capturing the high-level system services from the requirement text.

### Improved Structural Model

Once the FCSM and the ECC model are created, the next step is to employ them in the Improved Structural Model Algorithm (ISMA) for generating the ISM.

The steps of the ISMA are outlined below:

**Step 1** The concepts that exist in both models are the best candidates to be added to the ISM.

**Step 1.1** Selected attributes of the FCSM and the ECC model are added to the ISM. Here, selection means that:

- (i) We have to find the proper attribute that addresses a certain adjective such as the attribute size that addresses the adjectives large or small.
- (ii) If the identified attribute is one of the keywords *detail* or *information*, it will be highlighted and READ user is asked to specify all the attributes that should be tracked. For example, the phrase *Payment Information* will be highlighted and the READ user has to specify clearly the attributes that should be tracked about *Payment* such as *date*, *amount*, etc.



**Step 2** If any FCSM concept partially matches the concepts in the ECC model, they are both added to the model and highlighted with the same color to show that they might be identical. The READ user will make the appropriate choice.

**Step 2.1** If any two concepts in the FCSM that are related to each other with relation labeled *has*, partially match a single concept in the ECC model, they will be highlighted as a candidate for representing one single concept. The READ user will make the appropriate choice.

Example: Suppose that because of the bad style of writing the phrase *purchase order* is not identified as single concept in the FCSM which may result in having concepts *order* related to the 'purchase' with the relation *has*. Then ISMA will identify *purchase order* as one of the partial matches for both of these concepts. Therefore, the concept *purchase order* will be suggested to the READ user as a potential single concept.

**Step 3** In order to make sure that we have extracted all the necessary elements for the SUD for our final model (ISM) and nothing is missing, we ask the READ user to answer four basic necessary questions extracted from the Larman's category list [14] as shown below:

- (1) What is the transaction?
- (2) Where is the transaction recorded?
- (3) What are the roles of the people and the organization?
- (4) What are the other collaborating systems?

In case one of the concepts corresponding to the answers of the previous questions is missing from the model, it will be added and the model will be updated correspondingly.

**Step 4** If there is a concept in the FCSM that does not exist in the ECC model as either an attribute or a concept, and on the other side it is not found in the answers to the previous questions, we flag the concept and the READ user will make a final decision on keeping or excluding the concept from the ISM.

**Step 5.1** If attribute A that exists in the FCSM or in the newly created ISM appears as a concept in the ECC model, it will be highlighted and shown to the READ user as a potential concept and the READ user will make a final decision regarding keeping it as an attribute or changing it to the concept.

**Step 5.2** For an attribute of the concept B in the ECC model that appears as a concept A in the FCSM and it is not captured as an attribute by the Rule 2.2 defined in section 5.2.2, if concept B already exists in the newly created ISM, only A is placed properly as its attribute. Otherwise, concept B is selected from the ECC model and added to the ISM while placing A properly as its attribute. If A is the attribute of more than one concept in ECC, all the possible concepts from the ECC will be highlighted and the READ user will make a proper selection.

For example, in the sentence '*the customer enters ID to the system.*', since it is not specified to which concept *ID* is related, it will not be captured as an attribute, instead, it will be considered as a concept in FCSM. However, *ID* is the attribute of the *Invoice* in ECC. Therefore, if the concept *Invoice* is already added to the ISM, *ID* will be placed as its attribute. Otherwise, first we add *Invoice* to the ISM and

then *ID* will be placed as its attribute.

**Step 6** If, at any time, concept A in the FCSM or newly created ISM is participating in a composition/aggregation relationship with a concept B in the ECC model, then both A and B concepts should be added to the ISM. Aggregation is used to show the relationship between the part and the whole so that we can have a better view over the relationships and attributes that go to the whole and the ones that go to the part. Composition is the strong relation between the whole and the part in which the parts are destroyed when the whole is destroyed.

**Step 7** If, at any time, concept A in the FCSM or newly created ISM is participating in a generalization relationship in the ECC model, both the general concept (parent) and its child are added to the ISM so that we can have clearer understanding of the general attributes and relations that apply to the general type as well as the ones applied to subtypes.

The major goal of Steps 6 and 7 is to provide better understanding and knowledge of the SUD.

**Step 8** With regard to relationships between each pair of concepts, should the relationship in the FCSM named differently from that of the ECC model, the ECC relation will be suggested to the READ user as an inclusive alternative but the READ user will make the final decision. This step is also useful for refining any duplicate relations that may exist between concepts.

**Step8.1** Considering the relationships between each pair of concepts, if the relation and one of the concepts are the same in FCSM and ECC, then the other remaining concepts from FCSM and ECC model are highlighted and shown to the



READ user as candidates for being synonyms. The READ user will make a final decision whether to keep them as synonyms or not.

**Step 9** Finally, a list of other remaining FCSM and ECC model concepts, their attributes and their relationships are provided to the READ user who has the option of adding any of them to the model as necessary.

The rationale behind the ISM hinges on providing both a more comprehensive model, assisting in the identification of the possible errors/inconsistencies in the text and giving stakeholders the flexibility of having the customized needs that do not match exactly the standard domain models (ECC models).

It should be noted that currently the FCSM can be generated for different requirement texts and it is independent of the domain. However, ISM is tied to the Invoicing domain but ISMA can be applied to other constructs as well when the corresponding ECC models are developed.

### **5.2.3 Capturing High-Level System Services**

The primary goal of this step is to propose a mechanism for identifying the high-level system services (potential summary-level use cases [6]) from the user requirements text, visualize the corresponding CUCM diagram representing the actors and services (use cases) and extract the use cases textual descriptions from the original text.

Use case identification can be done at different levels, such as business/interaction [20], or with different scopes, such as functional/design [6]. In the requirements elicitation phase, software system services are initially captured at a higher abstraction

level as “summary-level use cases” [6]. These are further refined into functional or design user-goal use cases [8]. In this work, a UML use case diagram is employed to graphically synthesize the content of the CUCM. Similar to a Use Case Model (UCM), actors, use cases, and the communications between them are the three elements that constitute a CUCM. Actors are divided into two categories: primary actors, which initiate an interaction with the system to achieve a goal, and supporting actors, which provide a service for the system [6, 21]. Our goal is to identify the software system summary-level use cases which are defined as the services provided by the system to the user. The steps of our methodology for semi-automatically generating CUCM from the textual user requirements and extracting a brief description of the summary-level use cases are summarized below:

**Step 1:** Identify the actors with ECC model assistance.

**Step 2:** Identify the high-level system services, called ‘summary-level use cases’ and the key sentences in the user requirements text characterizing each service.

**Step 3:** Extract a brief textual description of the summary-level use cases using a metrics-based text-partitioning algorithm.

**Step 4:** Identify the supporting actors associations.

**Step 5:** Draw a Context Use Case Diagram which depicts graphically the CUCM.

These steps are described in detail in the corresponding subsections.

The remaining issue is how to deal with structures such as while, go to, and if. Because our focus is on the summary-level use cases rather than detailed use case scenario descriptions, we are excluding the appearance of the keywords go to and while in our text. If clauses are normally used to express the conditions, status, or

state under which a certain relation is established or an activity is performed. In our methodology, the if clauses are visualized in the form of UML notes and may later be refined to user-goal use cases.

## **Actors**

Discovering and finalizing the existence of the actors is accomplished separately for each type of actors (primary and supporting).

**Primary Actor.** Each ECC model accommodates the possible roles played in that specific domain (e.g., Customer and Supplier in Invoicing Orders) by the various users of the system to achieve their requests. Therefore, the list of possible roles for each specific domain is generated automatically in terms of the potential primary actors in the system.

**Supporting Actor.** There are two main approaches to producing software systems: building them individually or developing them from the perspective that systems can collaborate (in other words, a holistic approach to system development) [43]. We are interested in the second approach. Modeling the interrelationship between software systems makes it possible to automatically elicit the supporting systems associated with the SUD as potential actors to support its services.

In order to provide the flexibility for the stakeholders and to take into account their customized needs which do not exactly match the standard domain models (ECC models), both lists will be shown to the READ user. Final modifications are done with the help of the READ user and the list is approved.

## Identification of System Services

SUD should provide certain services to the primary actors with the purpose of fulfilling their needs. In order to identify those system services (potential summary-level use cases), we search for and analyze two kinds of patterns in the ROM presentation of the text: i) relations directed from the SUD toward another entity, and ii) the relations that are directed from the primary actors toward another entity.

Whenever a relation is directed from the SUD to another entity, the combination of the relation and the entity can be a use case. Yet, not all these combinations are valid, and further analysis is required to reveal those that are. Relations stemming from the system can be divided into: (a) internal actions of the system; (b) the services of the system or high-level use cases; (c) any interaction between the system and supporting actors, such as forwarding a result, waiting for data [10], etc.; and (d) any interaction between the system and primary actors, such as asking for information or confirmation. In order to identify valid use cases, each <primary actor, trigger,SUD,relation directed from the SUD,entity toward which the relation is directed> tuple will be checked with the original sentences in the user requirements. The primary actor's relationship to the SUD can be considered as the triggering event for accomplishing a certain service (summary-level use case). This triggering event is normally stated in the requirements text using verbs such as request, ask, want, choose from, select from. If in the requirement text, a sentence with all the keywords in the tuple exists, then the use case and the communication between the actor and the use case are considered to be valid. Otherwise, they are invalid and will therefore

be ignored.

In completing the use case identification process, we study the relations that originate from the primary actors and are directed toward another entity. If the relation that originates from the primary actor is one of the reserved verbs *enter* or *type* and an entity toward which the relation is directed is found in the predefined list of attribute keywords (see Table 5.3), the relation and the entity will be ignored because this list contains some of the keywords such as ID, user name, password, etc., which in combination with the verbs *enter* or *type* are normally used in response to type (d) interactions. If the entity is not in the list and the relation is not one of the reserved verbs, we scan the original text sentence by sentence, looking for tuples of the type <primary actor, relation directed from the actor, entity toward which the relation is terminated, to/from, SUD>. We are seeking the relations that are directed toward a system entity with the prepositional relations *from* and *to*. If there is a sentence containing all the keywords, then a combination of the entity and the relation is considered to be a valid use case. The above two patterns were revealed by our studies of the user requirement documents in academic and course projects.

The result of this procedure is a set of sentences SS, each containing a primary actor and the verb indicating a particular use case of the SUD. There is one sentence in an SS per high-level system service. We remind the reader that high-level system services are referred to as “summary-level use cases” and are usually described in a narrative style.

## Summary-Level Use Case Briefs

Summary-level use cases are high-level descriptions of the services provided by the SUD. Our goal is to partition the original problem statement description around the sentences chosen in an SS into summary-level use case descriptions, one partition per sentence  $S_i \in SS$ , where each partition groups the sentences related to one service (use case). The equivalence criterion is the rule for evaluating the closeness of a sentence to  $S_i$  based on a distance metric. Such grouping increases the visibility of the text describing a service, which is possibly scattered among the paragraphs or pages of the original text. The increased visibility will facilitate the job of analysts in ensuring the completeness of the use case descriptions and in inspecting the text for possible inconsistencies between otherwise scattered statements.

**Metric-based text partitioning algorithm.** The set of sentences in the user requirements is represented as a metric space where the space points are abstractions of sentences. The word *metric* here means distance between two points (that is, abstractions of two sentences) in a metric space, where the distance is a measure of functional similarity/dissimilarity between two sentences.

Let  $RR$  be the set of all sentences in the user requirements, excluding those already chosen in the SS. The metric-based text-partitioning algorithm takes as input the sets  $RR$  and  $SS$ . It breaks down the set of sentences of the original problem statement into partition  $P_i$ , one partition per each sentence  $S_i \in SS$  (that is, for each summary-level use case). The number of partitions is equal to the number of sentences in  $SS$ . A sentence belongs to partition  $P_i$  if it is the closest to the corresponding  $S_i \in SS$ .

The distance between two sentences  $S_1$  and  $S_2$  ( $S_1 \in SS$ ,  $S_2 \in RR$ ) is calculated as follows:

$$sd(S_1, S_2) = \text{similarity}(S_1, S_2) * \text{dissimilarity}(S_1, S_2) \quad (5.1)$$

It should be noted that a similar approach was originally proposed in [46] for a metric-based test case partitioning algorithm. The similarity ( $S_1, S_2$ ) was redefined to adapt the formula to the software analysts' use case elicitation process. The details of the distance calculation are as follows:

First, sets  $WS_1$  and  $WS_2$  are generated for each sentence  $S_1$  and  $S_2$ , each of which contains the significant words in the corresponding sentences meaning that modal, auxiliary verbs, determiners, etc. are ignored and the remaining words are stemmed. Two tables in which each row corresponds to a sentence and each column corresponds to a different word in  $S_1$  and  $S_2$  are then generated. Each cell has a value 1 if the word corresponding to that column belongs to the sentence, and 0, otherwise. Thus, the sentences are converted into binary strings (rows are binary strings representing the sentences) forming a metric space on which the distance  $sd$  between two sentences  $S_1$  and  $S_2$  is defined. The first table (see, for example, Table 5.23) contains actors and actions, and the second table (see Table 5.24) contains the remaining words in the sentence. The first table is used for calculating the similarity ( $S_1, S_2$ ), while the second table provides the necessary information for calculating dissimilarity ( $S_1, S_2$ ). Similarity and dissimilarity are calculated from the above binary strings as follows:

$$\text{Similarity}(S_1, S_2) = 2^{-C(S_1, S_2)} \quad (5.2)$$

where  $C(S_1, S_2)$  is the number of common actors and actions in  $S_1$  and  $S_2$  sentences.

This definition is justified by the fact that the use cases might include common behavior started off by the same request from a primary actor (input action). In the analysis phase the above mentioned common behavior will be refined into a set of scenarios defining the use case. The range of the similarity measure is between 0 and 1, where values closer to 0 indicate greater similarity and 1 indicates no similarity. When identifying the actions (verbs) for the similarity table, if the sentence contains one of the triggering verbs (request, ask, want, choose from, select from), both the trigger and the main action are placed in the similarity table. This is due to the fact that triggering verbs do not represent the main action of the sentences and they just present the trigger that causes the action.

The dissimilarity measure between two binary strings representing  $S_1$  and  $S_2$  is calculated as the number of elementary transformations or, equivalently, number of words that should be changed in order to transform string  $S_1$  into string  $S_2$ , excluding the words that are already counted in similarity (see Table 5.24). The more the set of words manipulated in one sentence differs from another sentence, the more dissimilarity is between them. The distance formula  $sd(S_1, S_2)$  indicates that the more distance there is between two sentences, the more they will differ in content, and thus the less likely they will be to characterize the same use case. It is assumed in the distance formula that dissimilarity will never get the value of 0 unless two sentences are identical which will result in having the distance equal to 0.

For instance, let  $S_1 =$  *'The customer requests the CBMSys to place an order.'* and  $S_2 =$  *'If the customer's credit record is good, then the CBMSys places the order.'* The distance between the sentences  $S_1$  and  $S_2$ , using the information shown in Tables



5.23, 5.24 is calculated as  $sd(S_1, S_2) = 2^{-2} * 2 = 0.5$ .

Table 5.23: Calculating similarity.

	Customer	Request	Place
S1	1	1	1
S2	1	0	1

Table 5.24: Calculating dissimilarity.

	CBMsys	Credit	Record	Order
S1	1	0	0	1
S2	1	1	1	1

The distances between all the sentences in RR and each of the sentences in SS are calculated using the suggested above formula. Each sentence is placed in one of the partition from which its distance is minimal. If the shortest distances to different partitions are equal, the algorithm calculates the distance between that specific sentence and the rest of the sentences in each chosen partition; the sentence is finally added to the partition  $P_i$  from which its distance is minimal. The sentences corresponding to the set of binary strings in the  $P_i$  are then recovered and the use case summary is generated and shown to the READ user.

## **Supporting Actor Associations**

The communication links between the use cases and the supporting actors are then extracted based on the appearance of the supporting actor names in the use case summary description.

## **Use Case Context Diagram**

The Context Use Case Diagram is generated from the CUCM elements (actors, use cases, and their communications) extracted in the steps outlined above.

## **Discussion**

As mentioned earlier, user requirements text is normally written in natural language. The writing style and the terminology used for describing the problem are highly dependent on the individuals who record them [6]. In this context, applying ECC models may give rise to the question of how certain we are that the same vocabulary will be used by the authors of the user requirements.

The terms used in the ECC models are standard in each domain and are applicable to different organizations with different needs [43]. They form the dictionary of terms which can be used in writing the user requirements, and authors of the user requirement texts are greatly encouraged to use them. It should be noted that using unpopular terminologies or different terms for a single concept may give rise to inconsistencies and uncertainty in the later stages of development of the SUD. It is worth noting that using homogenized terminology has previously been suggested by [6, 20] as a pattern for specifying the requirement text and use cases.

The methodology we describe here is illustrated in a case study in the next chapter.

# Chapter 6

## Illustration

In this section, the proposed methodology in Chapter 5 is illustrated on the Invoicing Orders case study. The problem statement is inspired by [42] and the Invoicing Orders System requirements for a fictitious company, CBM Corp [32].

This example has been chosen because, despite its simplicity, it gives us the opportunity to clearly explain the details of our methodology. The sentences are numbered to simplify the illustration process.

### 6.1 User Requirements

*1. The customer requests the CBMSys to place an order. 2. CBMSys retrieves customer's credit record from the Customer Persistent Storage (CPS). 3. If the customer's credit record is good, then the CBMSys places the order. 4. CBMSys creates and sends the order to the publisher. 5. The CBMSys receives the invoice of the order from the publisher. 6. If the purchase invoice's details are correct, then the CBMSys*

*accepts it and sends it to the Account Payable System (APS). 7. The CBMSys prepares publisher payment and sends the check to the publisher. 8. The CBMSys assigns shipment to the orders. 9. CBMSys generates a sale invoice for the customer. 10. The customer provides payment information to the CBMSys, and the CBMSys sends it to the Accounts Receivable System (ARS). 11. The customer shall view the order status from the CBMSys. 12. The customer enters the order ID to the CBMSys, and CBMSys shows the order status to the customer. 13. Customer and publisher shall be able to update their Personal Profile from the CBMSys.'*

## 6.2 Preprocessing of the Requirement Text

In our methodology it is assumed that the ambiguities have been removed from the text with the help of the stakeholder [16]. In our sample text, after ambiguity detection (see section 2.4.1), pronouns will be substituted by the corresponding nouns to which they are referring to and few minor changes are also made to the text such as removing parenthesis. The modified text will be the input that is used in other steps.

**Formal Presentation.** When ambiguities are removed from the text, the next step is generating its ROM diagram (see section 5). We will illustrate the process on one of the sentences of the text as an example.

Given the following sentence in plain English, after applying the axiomatic theory of design modeling we will obtain the ROM diagram as shown in Fig. 6.1:

'The customer enters the order ID to the CBMSys, and CBMSys shows the order status to the customer.'

Tables 6.1 and 6.2 show the parts of speech and syntactic analysis for the above sentence.

Table 6.1: Parts of speech for the sample sentence.

```
(ROOT(S (S
(NP (DT The) (NN customer)), (VP (VBZ enters),
(NP (DT the) (NN order) (NN ID))
(PP (TO to), (NP (DT the) (NNPS CBMSys))))), (, ,)
(CC and), (S (NP (NNP CBMSys)), (VP (VBZ shows),
(NP (DT the) (NN order) (NN status))
(PP (TO to), (NP (DT the) (NN customer))))), (. .)))
```

Table 6.2: Syntactical analysis for the sample sentence.

```
Det (customer-2, The-1), nsubj (enters-3, customer-2), det (ID-6, the-4),
nn (ID-6, order-5), dobj (enters-3, ID-6), det (CBMSys-9, the-8),
prep-to (enters-3, CBMSys-9), nsubj (shows-13, CBMSys-12),
conj-and (enters-3, shows-13), det (status-16, the-14), nn (status-16, order-15),
dobj (shows-13, status-16), det (customer-19, the-18), prep-to (shows-13, customer-19)
```

According to the sentence pattern that is shown in (Rule 7.1, 5.2.2) we will obtain the following ROM diagram (see Fig. 5.1).

The detailed information on the process of transforming the sentences in natural language to the ROM presentation can be found in [5], [50], and [51].

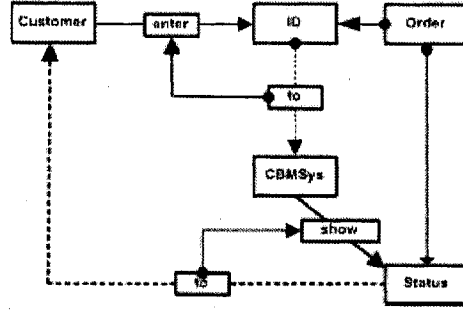


Figure 6.1: Sample ROM diagram.

As mentioned before, our methodology is based on the ROM presentation. The overall ROM presentation for the text is shown in Fig. 6.2. The diagram is the simplified view of the ROM output, created manually, based on the descriptions in [51]. It is shown with the purpose of simplifying the understanding of our methodology.

## 6.3 Structural Analysis

**First Cut Structural Model (FCSM).** Having the previous sentence in mind, in this section, we will explain how the transformation rules described in section 5.2.2 of Chapter 5 can be used to extract the necessary elements for generating the FCSM:

- (i) Applying rule 1 will result in having Customer, Order, and CBMSys as concepts.
- (ii) Applying rule 2.2 will result in having status and ID as the attributes for Order.
- (iii) Applying rule 7.1, Note 1 will result in having a relationships *Enter Order ID* and *View Order status* between the customer and CBMSys.

Fig. 6.3 shows the overall FCSM for the text. This figure is generated directly from the formal ROM presentation by applying the rules developed for mapping ROM

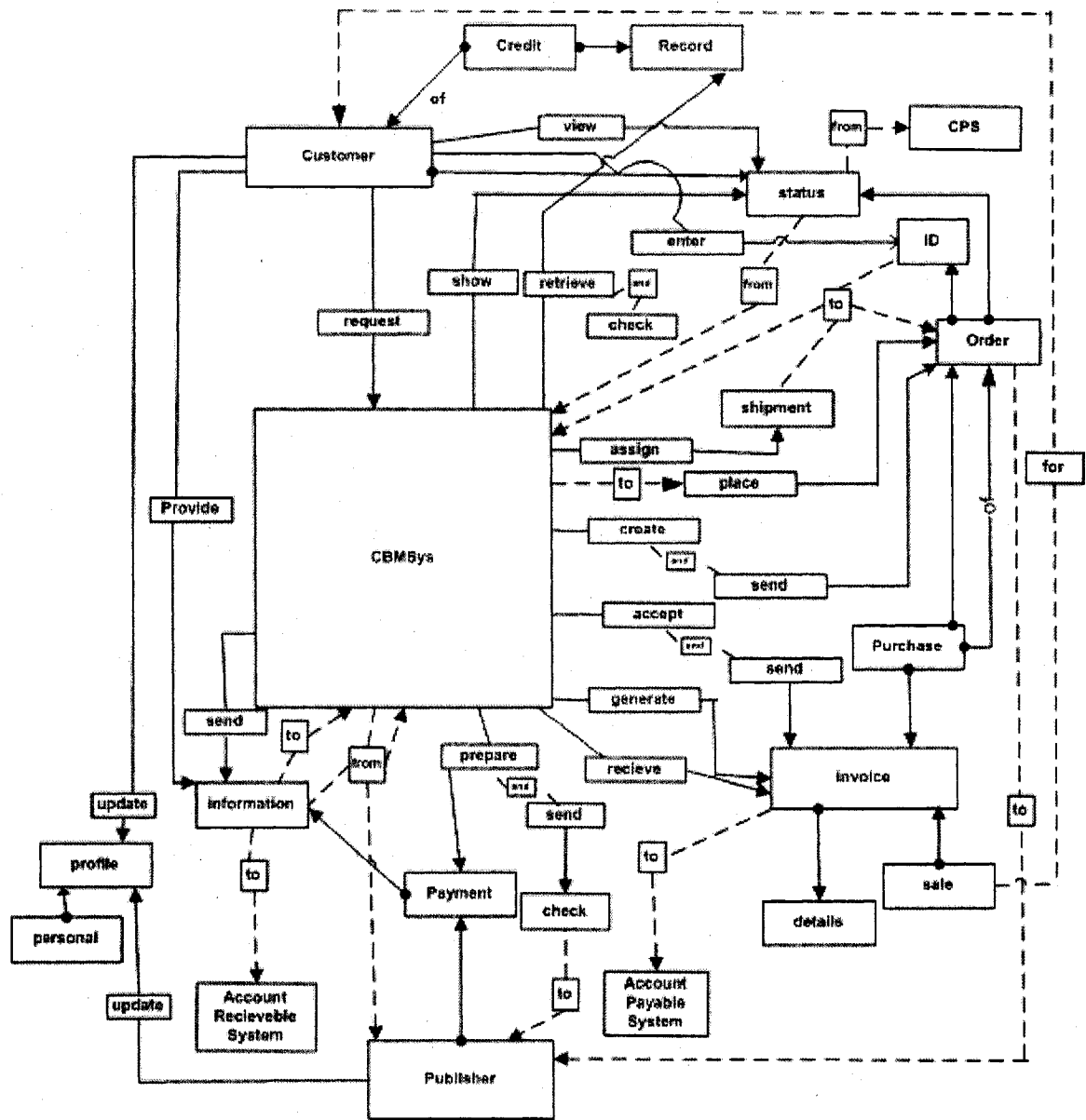


Figure 6.2: Overall ROM presentation.



to the SM to the whole text.

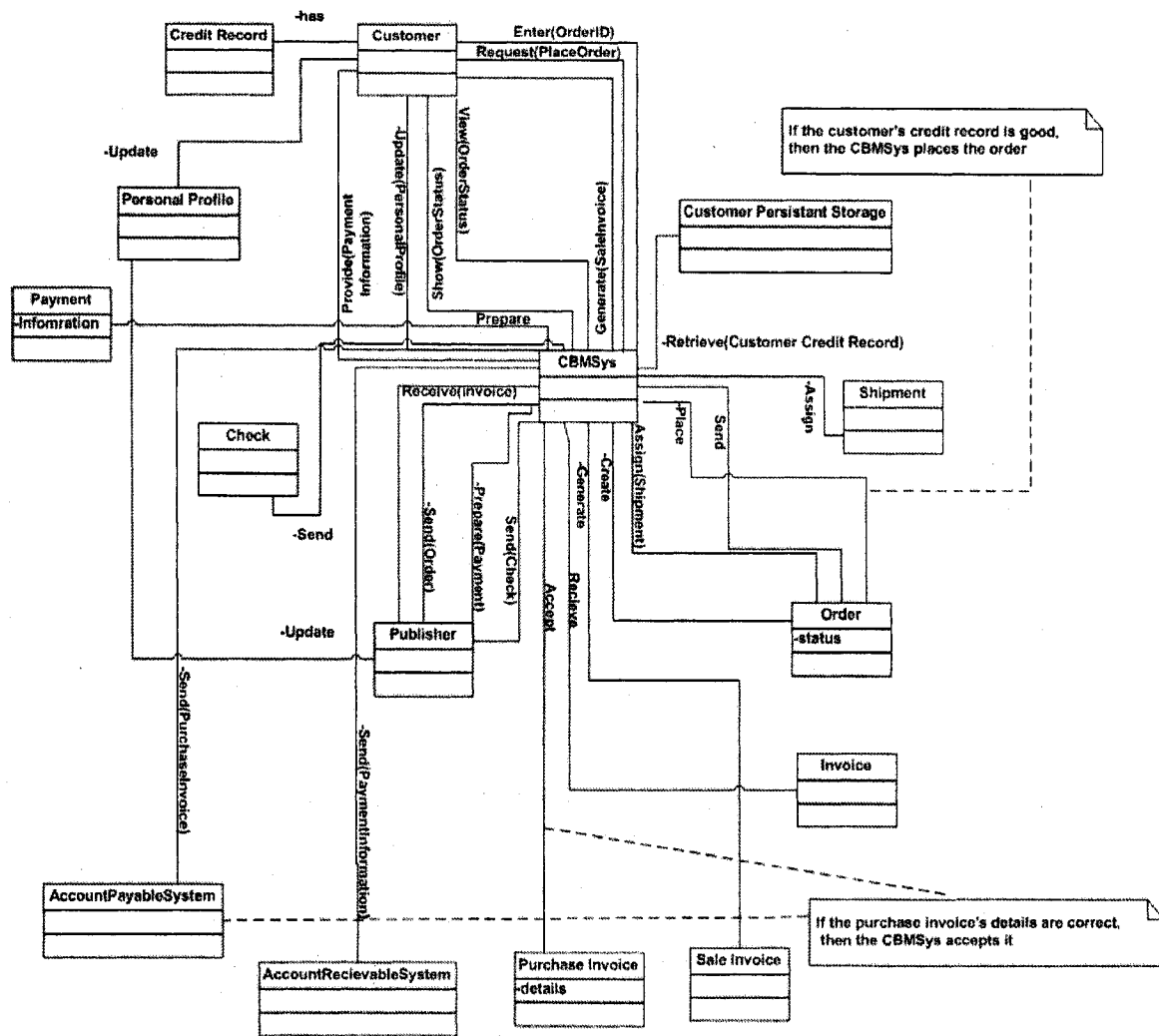


Figure 6.3: FCSM.

**ISM.** An improved (revised) version of the SM is generated from the FCSM using the ISMA. The steps of the ISMA applicable to the sample requirement text are described in the following:

Step 1. Customer, Payment, Purchase Invoice, Sale Invoice, Invoice, and Shipment concepts are added to the ISM because they exist in both the FCSM and the ECC model.

Step 1.1. Regarding attributes for Payment and Purchase Invoice concepts, attributes from the ECC model are added to the ISM and because of the appearance of the word *information* in the phrase Payment Information and *details* in phrase Purchase Invoice details, READ user's validation is needed for finalizing the list of the attributes for these two concepts.

Step 2. Partially matching concepts from the FCSM and the ECC model respectively (such as Order with Ordering, Account payable and Account Receivable with Financial Accounts, Check with Certified Check, Personal Check, etc.) are added to the ISM and highlighted with the same color in order to show to the analyst that they might be interchangeable. The analyst will make the appropriate choice:

Step 3. The concepts representing the answers to the following questions will be added to the model (in case they are missing from the model) along with their attributes and any relationships they are participating in:

- (1) What is the transaction? Invoice.
- (2) Where is the transaction recorded? CBM System (CBMSys).
- (3) What are the roles of the people and the organization? Customer, Publisher.
- (4) What are the other collaborating systems? Account Payable System, Account Receivable System, and Customer Persistent Storage.

In our example this step will result in adding the concepts CBMSys and Publisher to the ISM. Customer has already been added in Step 1. Account Payable System, Account Receivable System, and Customer Persistent Storage have been added to the model in Step 2.

Step 4. The concepts Credit Record, Personal Profile appearing in the FCSM which do not exist in the ECC model, are flagged to the analyst before being omitted with the purpose of ensuring that we are not losing any important information extracted from the user requirements.

Step 5. The attribute *status* in FSCM is defined as a concept in ECC, therefore, it will be highlighted and shown to the READ user as a potential concept and the READ user will make a final decision regarding keeping it as an attribute or changing it to the concept.

Step 6. According to this step of ISMA, the concept Invoice Item is added to the model because invoice is composed of invoice items.

Step 7. The concept Invoice generalizes the concepts Sale Invoice and Purchase Invoice. The concept Payment Method is added because it generalizes the Check concept. These concepts come together with some of the relations that will be shown to the READ user; they are added to the ISM if approved by the READ user.

Step 8. We did not encounter any cases where the relationship in the FCSM was named differently from that of the ECC model, therefore, this step is not ap-

plicable to our example.

Step 9. The list of the remaining concepts from the ECC model (e.g., Receipt, Contact Mechanism, etc.) are presented to the READ user. In case there is a need to add a missing concept, it will be added to the model.

Fig. 6.4 shows the ISM generated from the user requirements text provided at the beginning of this chapter.

## 6.4 Identification of the System Services

In this section, we will illustrate the details of our methodology for generating CUCM from the user requirements where (i) the system services provided to the primary actors are extracted from the user requirements text, and (ii) the original text is partitioned into groups where each partition represents a collection of sentences describing one service. It is to be noted that the number of partitions is equal to the number of services identified and the partitions are mutually exclusive.

**Step 1: Primary and Supporting Actors.** The primitive list of actors generated for the READ user using the *Party Role-Invoice Specific* concept of the ECC model contains: (i) Customer, Supplier, and general Organization as the primary actors, and (ii) Shipment, Ordering, Financial Account, and Party which are shown as interfaces (other constructs) communicating with the invoicing domain as supporting actors. With the help of the READ user, this list is refined to Customer and Publisher as primary actors, and Accounts Receivable, Accounts Payable, and Customer persistent Storage as supporting actors.



**Step 2: System Services.** From the overall formal presentation of the text (ROM) shown in Fig. 6.2 we extract the services of the system (considered as potential summary use cases) according to the patterns described in section 5.2.3.

We are interested in the relations that are rooted in the system entity and the entity toward which the relations are directed. Sample candidate use cases from this point of view are displayed in Table 6.3.

We are also looking for all the relations that are rooted in primary actors and the entities toward which the relation is directed. Tables 6.4 shows sample potential use cases from this prospect.

For the former group of potential use cases, each <primary actor, trigger,SUD,relation directed from the SUD,entity toward which the relation is directed> tuple will be checked against the original sentences in the invoicing description. If a sentence with all the keywords in the tuple exists, then the actor, use case, and communication are considered valid. Otherwise, they will be ignored. For example, <customer, request, place, order> is valid because the sentence that contains all these keywords exists in the original problem statement as '*The customer requests the CBMSys to place an order.*' In contrast, <customer, request, prepare, payment> is not valid because none of the sentences in the original problem statement contain all these keywords.

As for the latter group of potential use cases, every tuple consisting of <primary actor, relation directed from the actor, entity toward which the relation is terminated, to/from, SUD> is aligned with the problem statement. As a result, *Update Profile* is identified as a valid use case associated with both the Customer and the Publisher, as well as *View Order status* which is associated with the Customer. On the other hand,

*Enter ID* is ignored because *enter* is one of the reserved verbs (see type(d) relations in section 5.2.3) and *ID* is one of the keywords on the predefined list of attributes in Table 5.3. Therefore, the phrase *Enter ID* is identified as a representative for the data or information provided in response to the system's request.

Table 6.3: Finding use cases from a formal presentation (system perspective).

Actor	Trigger	Sample potential use case
customer	request	show status
		prepare payment
		(to) place order

Table 6.4: Finding use cases from a formal presentation (actor perspective).

Actor	Sample potential use case
customer	update profile
publisher	enter ID
	view status

As a result of the above procedure, the main sentences of the requirement text characterizing the system services provided to the primary actors will be:

1. The customer requests the CBMSys to place an order.
11. The customer shall view the order status from the CBMSys.
13. Customer and publisher shall be able to update their personal profile from the CBMSys.

Therefore  $SS=\{1, 11, 13\}$ ,  $RR=\{2, 3, 4, 5, 6, 7, 8, 9, 10, 12\}$ , and we have three partitions.

Tables 6.5 summarize the information that we have extracted up until this part.

Table 6.5: Extracted actors and use cases.

Actor	use cases
customer-publisher	Update Profile
customer	Place Order
customer	View Order Status

**Step 3: Briefs.** After applying the text partitioning algorithm, each of the sentences in the  $RR$  set will be assigned to the corresponding partition where the partitions are mutually exclusive and their union will be exactly the set  $RR$ .

- (i) Sentences are divided into tokens (e.g., prepositions, articles are omitted and the words are stemmed.)
- (ii) The sentences in set  $SS$  are always considered as the main sentences because it is assumed that the brief description of the summary use cases is about them and it starts with them.

As described before, each sentence belongs to one partition only for which the distance between the given sentence and the partition's root is minimal. In case there is no minimum distance, that is, the smallest distances between the given sentence and two or more partitions' roots happen to be equal, then we have to find the distance of



the given sentence to the sets of sentences belonging to each partition. The sentence will be finally placed into the partition from which it has the minimum distance.

According to the same procedure described and shown in section 5.2.3, distances between the sentences in set SS and RR are calculated. The results of the calculations are summarized in the tables 6.6, 6.7, and 6.8 .

Table 6.6: Distance calculations for S1.

$(S_i, S_j)$	Distance
$(S_1, S_2)$	2.5
$(S_1, S_3)$	0.5
$(S_1, S_4)$	2
$(S_1, S_5)$	2
$(S_1, S_6)$	7
$(S_1, S_7)$	5
$(S_1, S_8)$	2
$(S_1, S_9)$	2
$(S_1, S_{10})$	2.5
$(S_1, S_{12})$	1.5

As it can be seen from the calculations, the first step of the metric-based text-partitioning algorithm results in three  $P_j$  sets, namely  $P_1 = \{1, 2, 3, 4, 5, 6, 8, 9, 10\}$ ,  $P_2 = \{11, 12\}$  and  $P_3 = \{13, 7\}$  corresponding to the sentences 1, 11 and 13 from the SS and the summary-level use cases *Place Order*, *Check Status*, and *Update Informa-*

Table 6.7: Distance calculations for S11.

$(S_i, S_j)$	Distance
$(S_{11}, S_2)$	3
$(S_{11}, S_3)$	2
$(S_{11}, S_4)$	3
$(S_{11}, S_5)$	3
$(S_{11}, S_6)$	8
$(S_{11}, S_7)$	6
$(S_{11}, S_8)$	4
$(S_{11}, S_9)$	2.5
$(S_{11}, S_{10})$	3
$(S_{11}, S_{12})$	1

tion.

All the sentences are correctly assigned to their corresponding partitions except sentence 7 which is wrongly assigned to  $P_3$ ; in reality, it belongs to  $P_1$ . The reason might be the ambiguous formulation of the sentence. Further analysis is required to clarify the issues related to the wrongly classified sentences. The refinement of the distance metric definition will be tackled in our future work.

**Step 4: Supporting Actor Communications.** The supporting actors identified from the  $P_1$  are Customer Persistent Storage and the Accounts Receivable System. This shows that there exists a communication between these two supporting actors

Table 6.8: Distance calculations for S13.

$(S_i, S_j)$	Distance
$(S_{13}, S_2)$	3
$(S_{13}, S_3)$	3
$(S_{13}, S_4)$	2.5
$(S_{13}, S_5)$	2.5
$(S_{13}, S_6)$	8
$(S_{13}, S_7)$	3
$(S_{13}, S_8)$	5
$(S_{13}, S_9)$	2.5
$(S_{13}, S_{10})$	3
$(S_{13}, S_{12})$	3

and the corresponding use case *Place Order*. No supporting actors are identified in  $P_2$  or  $P_3$ , therefore no communication exist between these use cases and any of the supporting actors.

**Step 5: Use Case Context Diagram.** The graphical model consolidating the above information is shown in Fig. 6.5.

Having illustrated the methodology, in the next chapter we introduce the prototype tool implementing our methodology.

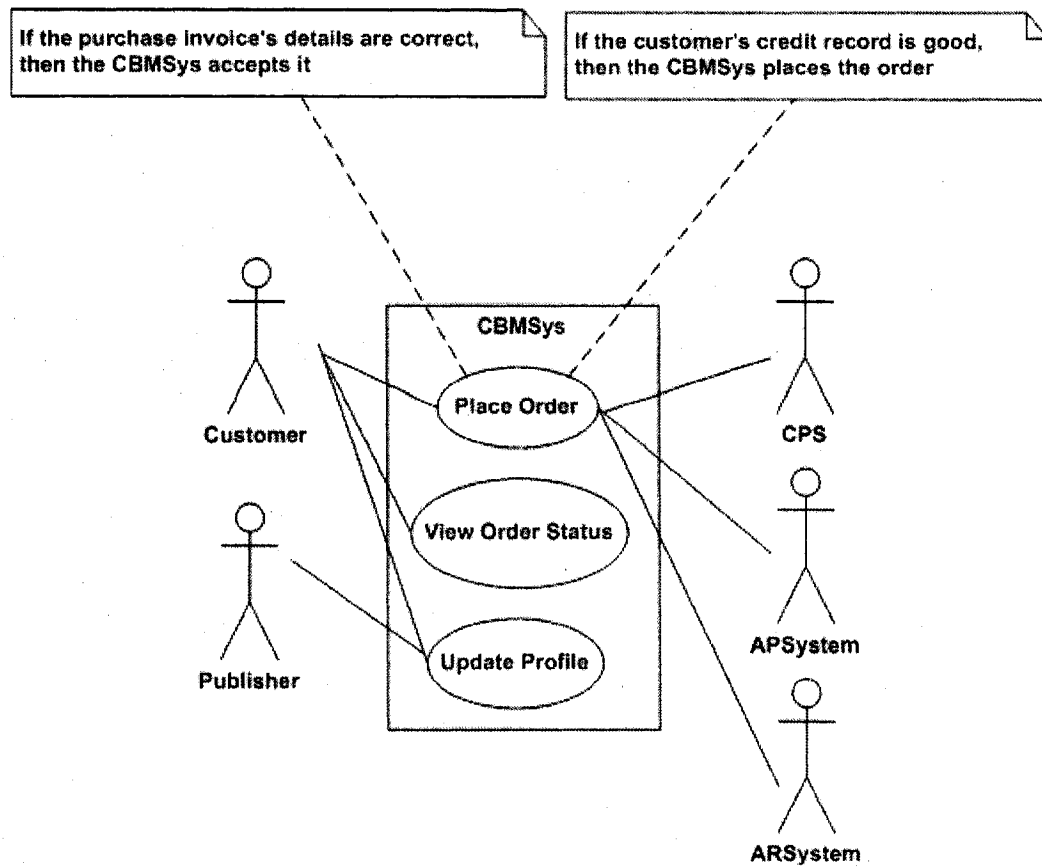


Figure 6.5: CUCM diagram.

# Chapter 7

## Prototype

### 7.1 Overview

This chapter is intended to present the details related to the proof-of-concept tool that implements our approach.

READ is a prototype tool that operates on a requirement text written in natural language. This tool is being developed with the aim of assisting the software analyst in the requirement elicitation process where correctly understanding the stakeholder's need is considered an important factor for the success of the project. Upon choosing a requirement text as an input, the requirement text is processed by ReqSAC tool [16] which is the ambiguity checker and then it goes through the ROMA tool from which the formal syntactical presentation of the text is generated and stored in XRD (an extension of the XML). ROMA-EVP Mapper is then used to transform the output of ROMA to the internal formal presentation of Eclipse Visualisation Plugin (EVP) in XML format. Finally, READ generates the SM as a UML domain model diagram and the context use case model CUCM as a UML Use Case diagram, both extracted from the formal representation of the text (ROM). The work of [16] and [5] provided

valuable input to this thesis. The logical architecture of the READ is shown in Fig.

7.1.

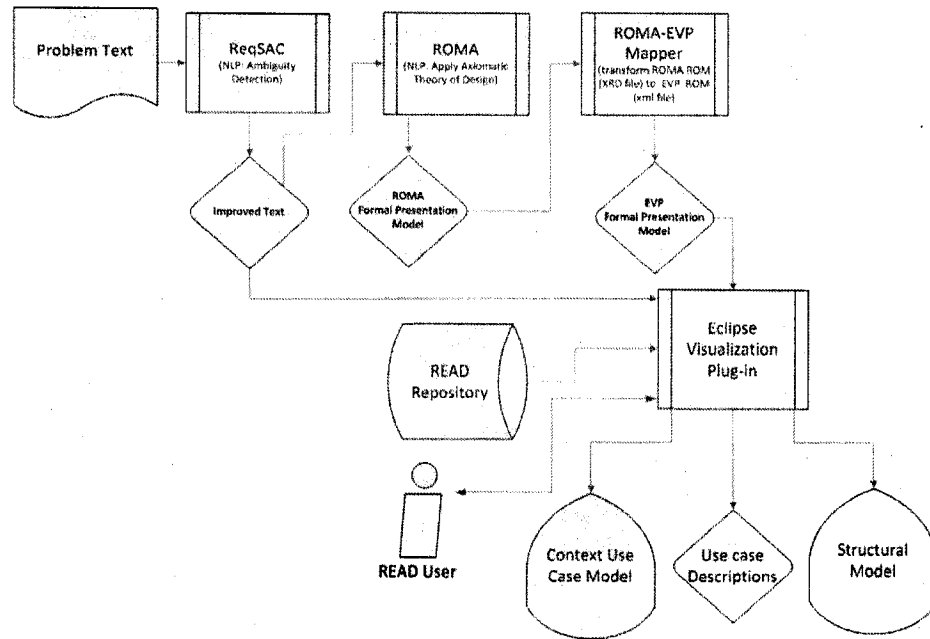


Figure 7.1: Architecture of READ.

As mentioned above, the first two components of READ had been developed during the previous phases of the project. One of the objectives of this thesis was to develop a prototype EVP, which was particularly concerned with generation and visualization of the domain model and context use case diagram from the ROM presentation of the text, as well as partitioning the original problem statement description around the sentences identified as the main sentences expressing high-level system services.

The process of EVP development was divided into two sub-projects: The first project addressed visualization, and the second was concerned with partitioning. In section 7.3, we provide detailed information about the components of the EVP.

## 7.2 Development Platform

EVP is a Java prototype that leverages the open-source Eclipse framework and related projects [11, 12]. In particular, EVP depends on the UML2 component and UML2 Tools project. UML2, a component of the Model Development Tools (MDT) project, is an implementation of the UML 2.x OMG metamodel using the Eclipse Modeling Framework (EMF) [11]. The UML2 Tools project is a set of UML diagram editors developed using the Graphical Modeling Framework (GMF) for viewing and editing UML models [12].

## 7.3 EVP Modules

EVP consists of three main modules: the Structural Visualizer (SV), which displays the SM diagram, the Context Use Case Visualizer (CUCV), which displays the CUCM diagram, and, finally, the Partitioner that calculates the distances between sentences and uses these distances to partition the whole text around the main sentences expressing system services. The SV consists of two sub-components: the First-Cut Structural Visualizer (FCSV) and the Improved Structural Visualizer (ISV). Currently, CUCV and Partitioner modules are dependent on each other but Structural Visualizer can work independently.

Each of EVP modules is described in more detail in the next section.

### 7.3.1 First-Cut Structural Visualizer

The main functionality of this component is to generate the first sketch of the domain for the SUD. This model is composed of the concepts mentioned in the text, relations

between them and their attributes. As an input, this modeler reads the XML file that holds the ROM formal presentation of the text. Then, it processes the file according to the following pseudo code.



---

**Algorithm 1:** Pseudo code for FCSV.

---

```
input : ROM XML files
output: First-Cut Structural View XML file

1 Get the XML root element name and create a model with this name ;
2 foreach <Entity> element in the XML File do
3   Get the XML <Ent-Name> element value ;
4   if XML <Ent-Name> value is not in the predefined list of attributes then
5     Check for duplicate class;
6     Create a class with XML <Ent-Name> value;
7     Get the XML <Ent-Attr> value and add it as an attribute to the class;
8   end
9   foreach XML <Relation> element do
10    Get relation type;
11    if relation type is association-1 or modification then
12      if the target is in the predefined list of attributes then
13        Check for duplicate attribute in class;
14        Add attribute to the class defined by <Ent-Name>;
15      end
16      if the target is not in the predefined list of attributes then
17        Check for duplicate class in model;
18        Create a class with name defined by the XML target attribute;
19        if modification then set relation label to 'has';
20      else
21        set relation label to XML relation element value;
22        Create association between <Ent-Name> class and target class using The name
        label;
23      end
24    end
25  end
26  if relation type is association-2 then
27    Check for duplicate class in model;
28    Create a class named by the XML i-target value;
29    Get XML Relation element value;
30    Set label to XML Relation element value concatenated with XML d-target;
31    element value and XML refAttr value;
32    Create association between <Ent-Name> class and i-target class using the name label;
33  end
34  if relation type is association-3 then
35    Check for duplicate class in model;
36    Create a class named by the XML d-target value;
37    Set label to XML Relation element value concatenated with XML refRel;
38    attribute value and i target attribute value;
39    Create association between <Ent-Name> class and d-target class using the name label;
40  end
41  if relation type is generalization then
42    Check for duplicate class in model;
43    Create a class named by the Relation element value;
44    Create association between <Ent-Name> class and the relation element value class;
45  end
46  if relation type is of-preposition then
47    if <Ent-Name> element value is in predefined list of attributes then
48      Check for duplicate class in model;
49      Create a class named by the target attribute value;
50      Check for duplicate attribute;
51      Add <Ent-Name> element value as attribute to target class;
52    end
53    if <Ent-Name> element value is not in predefined list of attributes then
54      Check for duplicate class in model;
55      Create a class named by the target attribute value;
56      Check for duplicate class in model;
57      Create a class named by the <Ent-Name> element value;
58      Create an association between these two classes labeled 'has';
59    end
60  end
61 end
62 end
```

---

As an output, the algorithm generates an XML file that represents the First-Cut Structural view. The model described in this XML file is implemented as a UML 2.x (currently at version 2.2.1) metamodel provided by the Eclipse UML2 component, which is stored as an XMI file. The UML2 Tools project class diagram editor is then used to display the SM diagram.

### **7.3.2 Improved Structural Visualizer**

This module offers the functionality of generating an improved SM for the SUD. It aims at creating a more mature model in accordance with the ISV algorithm demonstrated in 5.2.2. For this purpose, it considers both the output of the FCSV and it also interacts with the READ Repository that stores the ECC model.

Basically, the modeler reads a file that represents the SM as well as a second file that represents the ECC model and is stored in the READ Repository. It implements the ISV algorithm and then it generates the final file that holds the information for the Improved Structural View. ISV pseudo code is used to explain the processing steps of this module.

As explained before, the model described in the XML file generated by this component is then implemented as a UML 2.x metamodel provided by the Eclipse UML2 component, which is stored as XMI files. The Eclipse UML2 Tools project class diagram editor is then used to display the Improved Conceptual Domain Model.

### **7.3.3 Context Use Case Visualizer**

This component of EVP is responsible for gathering all the necessary elements for the CUCM (i.e., actors, use cases, and the communications between them) through

---

**Algorithm 2'. ISV.**

---

```
input : First Cut and Category-List XML files
output: Improved XML file

1  Get the FC-XML root element name and create a model with this name ;
2  foreach element class of the FC - XMLFile do
3      Get the FC-XML class name ;
4      foreach element class of the ECC - XMLFile do
5          Get the ECC-XML class name ;
6          if FC-XML class name is equal to the ECC-XML class name or FC-XML class name contains ECC-XML class name
7              or ECC-XML class name contains FC-XML class name then
8              The same process as of line 4 of the pseudo code for FCSV ;
9              (Please,note that in the pseudo code provided below the notion of class is equivalent to the notion of element
10             in pseudo code for FCSV ;
11             )
12         end
13     end
14     foreach element class of the Category - List - XMLFile do
15         Get the Category-List-XML class name ;
16         The same process as of line 4 of the pseudo code for FCSV;
17     end
18     foreach element class of the FC - XMLFile do
19         Get the FC-XML class name ;
20         foreach element class of the ECC - XMLFile do
21             Get the ECC-XML class name;
22             if there exist no ECC-XML class name that is equal to FC-XML class name then
23                 Show that FC-XML class name to the user;
24             end
25         end
26     end
27     foreach element class of the FC - XMLFile do
28         Get the FC-XML class name ;
29         foreach element class of the ECC - XMLFile do
30             Get the ECC-XML property;
31             if the FC-XML class name is equal to ECC-XML property and FC-XML class name exists in the FCC model then
32                 The same process as of line 7 of the pseudo code for FCSV else
33                 Get the ECC-XML class name ;
34                 The same process as of line 4 of the pseudo code for FCSV;
35             end
36         end
37     end
38     foreach element class of the FC - XMLFile and the newly created model(IMP - XML) do
39         Get the FC-XML class name;
40         foreach element class of the ECC - XMLFile do
41             Get the ECC-XML class name;
42             if ECC-XML class name is equal to FC-XML class name then
43                 Get the ECC-XML class relation (association);
44             end
45             foreach element association of the ECC - XMLFile do
46                 Get the association type;
47                 if the ECC-XML is equal to 'Aggregation' or 'Composition' then
48                     Get the MemberEnd and OwnedEnd for that association;
49                     Check for duplicate class in model;
50                     Create a class with name defined by the class that is not already in the model ;
51                 end
52                 if the ECC-XML is equal to 'Generalization' then
53                     Get the MemberEnd and OwnedEnd for that association;
54                     Check for duplicate class in model;
55                     Create a super class/sub class which is not already in the model ;
56                 end
57             end
58         end
59     end
60     foreach element class of the FC - XMLFile do
61         Get the FC-XML class name;
62         foreach element class of the ECC - XMLFile do
63             Get the ECC-XML class name;
64             if the FC-XML class name is equal to ECC-XML class name then
65                 Get the FC-XML Association ;
66                 Get the ECC-XML Association;
67                 if memberEnd and ownedEnd of the associations are equal and names of the associations are not equal then
68                     Show the ECC-XML Association name as an alternative to the user;
69                 end
70             end
71         end
72     end
73     foreach element class of the ECC - XMLFile do
74         Get the ECC-XML class name ;
75         foreach element class of the FC - XMLFile do
76             Get the FC-XML class name;
77             if FC-XML class name is not equal to FC-XML class name then
78                 Show that ECC-XML class name to the user as concept that can be added to the model;
79             end
80         end
81     end
82 end
```

---

couple of different processes and finally displaying the diagram graphically using UML2 notation.

*Identifying Actors.* In this process, a simple interactive interface is used to finalize the set of primary and supporting actors for the system. At first, the system suggests the preliminary list of actors to the analyst as described in 5.2.3. The analyst modifies the list according to his/her expectations. The final list of actors is saved in an XML file called CUCM.xml. The root element of this xml file is the 'system boundary' and its value is equal to the name of the SUD.

*Removing Unwanted Words from the Preprocessed Requirement Text. Tokenization.* Using ReqSAC [16], each of the sentences in the original requirement text are divided into tokens<sup>1</sup>. *Stemming.* Next, using ReqSac morphological analyzer the identified tokens are stemmed and the result is saved<sup>2</sup>. Next, the unnecessary words such as determiners and modals are removed from the sentences because we believe that they will introduce too much unwanted noise in the data. It should be noted that the prepositions are not removed from the text. They will be used later for identification of the use cases. The result is saved in an XML file called Preprocessed.xml. The CUCV reads this XML file and for each sentence, it saves the tokens.

CUCV reads four XML files: formal ROM presentation of the text for extracting the candidate use cases, Preprocessed.xml for comparisons, CUCM.xml which contains approved list of actors and, finally, the last XML file called typed.xml which contains the keyword related to the type (d) relations as described in 5.2.3. In CUCV

---

<sup>1</sup>The aim of the Tokenization is to divide the text into units, called tokens. A token can be a word but it can also be a number, a punctuation mark, an abbreviation, etc.

<sup>2</sup>In Stemming affixes are removed from the tokens and they are turned into their inflectional forms.

pseudo code, the corresponding processing steps are presented.

In order for this model to be considered complete the communication between the supporting actors and the use cases should be added to the model. Identifying the correct communications is handled by the text partitioning algorithm as described in 5.2.3. At this stage, this model will be saved as an XML file and, when the communications are identified, each of them will be added to the model and associated to the corresponding summary use cases. Then, the XML file is ready for visualization. Once again, the model described in this XML file is implemented as a UML 2.x metamodel provided by the Eclipse UML2 component, which is stored as XMI files. The UML2 Tools project use case diagram editor is then used to display the Context Use Case model.

#### **7.3.4 Distance Calculator-Partitioner (DC-P)**

Finally, this component of EVP is assigned the responsibility of extracting the use case briefs by partitioning the requirement text. Each of the partitions contains the main sentence that represents the service of the system and is identified with the CUCV module (see section 7.3.3) in addition to the other sentences describing briefly that specific service as described in section 5.2.3 of the methodology.

As an input, it reads an XML file with the list of the main sentences and creates the corresponding partition for the sentences. It also reads both the XML file that contains the original requirements text and preprocessed requirement text in which each sentence is stemmed and unwanted words are stripped off of it 7.3.3. One more XML file is also read by this component, which contains the remaining sentences

---

**Algorithm 3:** Pseudo code for Context Use Case Visualizer.

---

**input :** ROM,Preprocessed,typed and CUCM XML files  
**output:** Context Use Case Diagram XML file, Main services XML file

```
1 Get the CUCM-XML/or ROM-XML root element and create a model with this name ;
2 foreach <Entity> element in the ROM-XML File do
3   Get the ROM-XML <Ent-Name> element value;
4   if ROM-XML <Ent-Name> element value is equal to the 'name of the SUD' then
5     foreach ROM-XML <Relation> element do
6       Get the ROM-XML <Relation> element value;
7       Get the ROM-XML <Relation> d-target attribute value;
8       Create a new array i ;
9       Save <Relation> element value and <Relation> d-target attribute value in that array
10    end
11  end
12  if ROM-XML <Ent-Name> element value is equal to the name of the actors then
13    foreach ROM-XML <Relation> element do
14      Get the ROM-XML <Relation> element value;
15      if <Relation> element value is equal to one of the triggering verbs then
16        Create a new array j ;
17        Save <Relation> element value and <Ent-Name> element value in the array;
18        foreach array i created previously do
19          Add the content of this array i to array j ;
20          Compare the content of the merged arrays with each of the preprocessed array;
21          if they are equal then
22            Check for duplicate Actor element name;
23            Create an element Actor ;
24            Set the name of the Actor to<Ent-Name> element value;
25            Check for duplicate UseCase element name;
26            Create an element UseCase;
27            Set the element name to <Relation> element value + <Ent-Name> element value;
28            Create an association between them;
29            Save the sentence in a Main services XML file;
30          end
31        end
32      end
33      Get the 'type','i-target' value and 'd-target' value for the <Relation> element;
34      if the 'type' value is equal to 'association-2' and 'i-target' is equal to the name of the SUD and the
        d-target is not equal to one of the keywords in typed.xml then
35        Create an array k;
36        Save <Relation> element value and <Ent-Name> element value in the array;
37        Get the ROM-XML <Relation> d-target attribute value;
38        Add the attribute value to the content of the created array K;
39        foreach array k do
40          Compare the content of the merged array with each of the preprocessed sentences ;
41          if preprocessed sentence content is equal to array K and strings 'of/to' and the 'name of
            the SUD' then
42            Check for duplicate Actor element name ;
43            Create an element Actor ;
44            Set the name of the Actor to<Ent-Name> element value;
45            Check for duplicate UseCase element name;
46            Create an element UseCase;
47            Set the value of the element to <Relation> element value + 'd-target' value ;
48            Create as association between them;
49            Save the sentence in a Main services XML file;
50          end
51        end
52      end
53    end
54  end
55 end
```

---

of the text after they have been pre-processed. Next, all the actors' names as they appear in the input file are flagged in the main sentences and the remaining sentences files. Partitioner then implements the Metric-based text partitioning algorithm as a result of which the use case summary description is generated and shown to the analyst. Pseudo code for DC-P is describing the processing steps of this module in detail. Fig. 7.2, 7.3 and 7.4 show snapshots of the EVP.

---

**Algorithm 4:** Pseudo code for DC-P.

---

```

input  : Main sentences, Actors, Remaining sentences, and original requirement text XML files
output: Use Case Briefs

1 Read the main sentences(SS) and remaining sentences(RR) files;
2 foreach ( $S_i \in SS$ ) do
3   foreach ( $S_j \in RR$ ) do
4     Create a similarity table in which each row corresponds to the sentences  $S_i$  and  $R_i$  and each cell corresponds
      to the Actors and Actions (verbs) in the sentences ;
5     foreach column in a table do
6       Get the word corresponding to that column;
7       scan the sentence;
8       if the word exists in the sentence then the value of the cell is equal to '1' else
9         the value of the cell is equal to '0';
10      end
11    end
12    Create a dissimilarity table in which each row corresponds to the sentences  $S_i$  and  $R_i$  and each cell
      corresponds to the remaining words in the sentences ;
13    repeat the lines 5 to 9 ;
14    calculate similarity and dissimilarity according to the formula;
15    calculate the distance and save it in a table;
16  end
17 end
18 foreach ( $S_j \in RR$ ) do
19   Compare the distance of the  $S_i$  to each ( $S_i \in SS$ ) ;
20   Get the minimum distance;
21   Retrieve the sentence(s) that correspond to that minimum distance;
22   if one ( $S_i \in SS$ ) is retrieved then Add ( $S_j \in RR$ ) in the partition of the ( $S_i \in SS$ ) else
23     Add ( $S_j \in RR$ ) to all the partition ( $S_i \in SS$ ) with minimum distance ;
24     Put ( $S_i \in RR$ ) in an array called 'undecided' ;
25   end
26 end
27 foreach  $S_i$  in 'undecided' array do
28   Retrieve the partition to which it belongs ;
29   Repeat steps 2-15 to calculate the distance between the  $S_i$  and the remaining sentences in that partition ;
30   Repeat step 18 to 24 to add the  $S_i$  to the appropriate partition;
31   if one ( $S_i \in SS$ ) is retrieved then Add ( $S_j \in RR$ ) in the partition of the ( $S_i \in SS$ ) else
32     print the message: 'The status of this sentence is not identified.';
33   end
34 end
35 Retrieves the sentences corresponding to each partition from the original requirement text ;

```

---

It should be noted that the implementation of the READ tool is still in progress.

As a proof-of-concept, few of the rules (Rules 1, 2 (partially), 3, 4, 5 (Note 2), 7.1, 7.2,

8, and 9) described in section 5.2.2 were implemented, and the logical architecture of the tool containing the components and the relations between them had been designed. As described earlier in this chapter, the corresponding pseudo codes of the main algorithms were developed. The integration of EVP with the previously developed READ components and the implementation of the remaining rules will be tackled in our future work.

After having proved the concept by developing a prototype tool implementing our methodology, we have validated the methodology as explained in the next chapter.



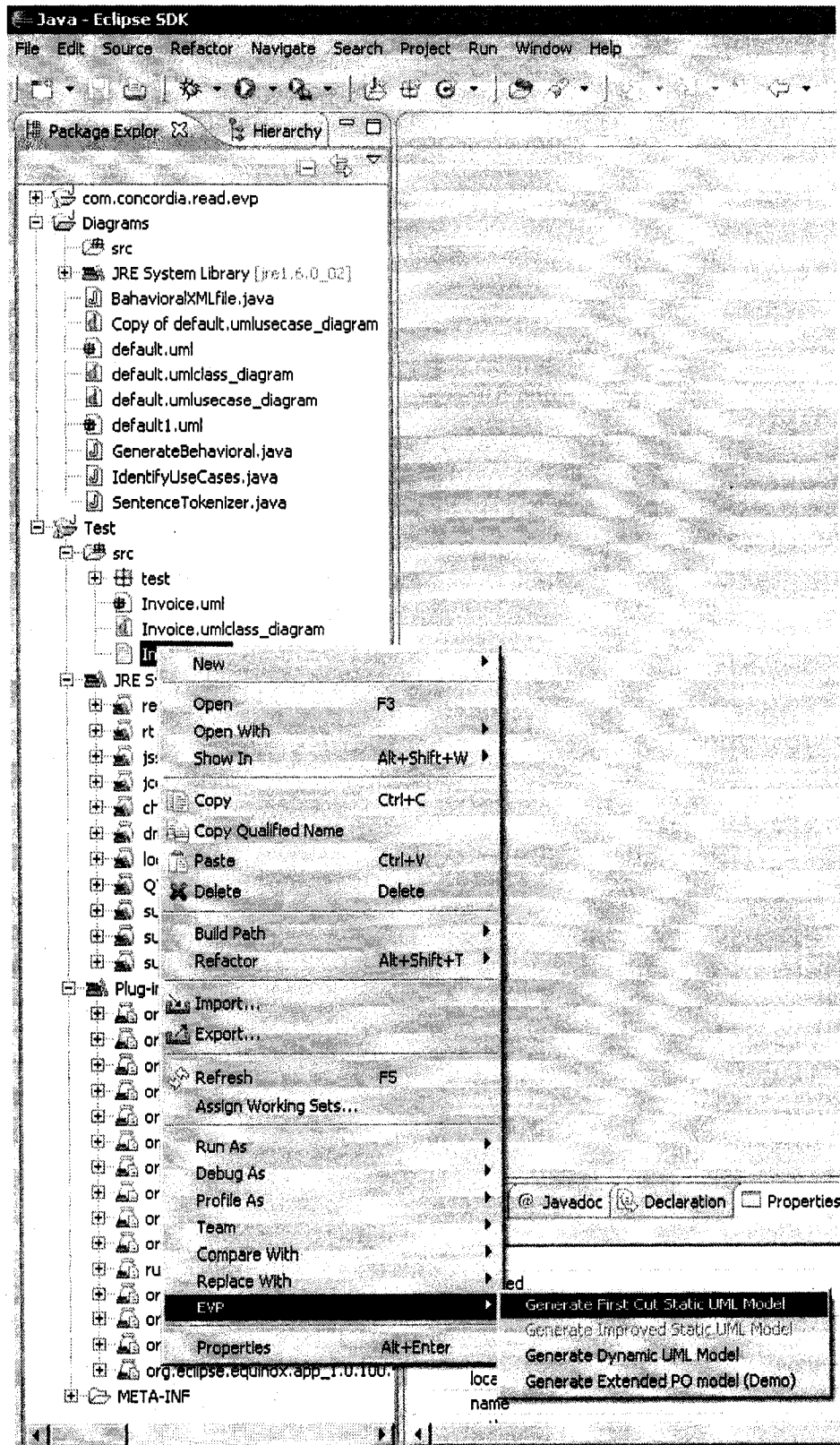


Figure 7.2: EVP context menu.

Message	Plugin	Date
Creating model...	com.concordia.read.evp	2008-05-01 10:44:1.600
Model 'OrderInvoicing' created.	com.concordia.read.evp	2008-05-01 10:44:1.605
Creating primitive types...	com.concordia.read.evp	2008-05-01 10:44:1.605
Primitive type 'OrderInvoicing::int' created.	com.concordia.read.evp	2008-05-01 10:44:1.605
Primitive type 'OrderInvoicing::string' created.	com.concordia.read.evp	2008-05-01 10:44:1.605
Class 'OrderInvoicing::Customer' created.	com.concordia.read.evp	2008-05-01 10:44:1.605
Class 'OrderInvoicing::InvoiceOrderSystem' created.	com.concordia.read.evp	2008-05-01 10:44:1.605
Association 'OrderInvoicing::Customer::dst' [0..1] --- 'OrderInvoicing::InvoiceOrderSystem::src' [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.605
Association 'OrderInvoicing::Customer::dst' [0..1] --- 'OrderInvoicing::InvoiceOrderSystem::src' [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.605
Class 'OrderInvoicing::Supplier' created.	com.concordia.read.evp	2008-05-01 10:44:1.605
Association 'OrderInvoicing::Supplier::dst' [0..1] --- 'OrderInvoicing::InvoiceOrderSystem::src' [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.605
Association 'OrderInvoicing::Supplier::dst' [0..1] --- 'OrderInvoicing::InvoiceOrderSystem::src' [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.605
Class 'OrderInvoicing::Order' created.	com.concordia.read.evp	2008-05-01 10:44:1.625
Association 'OrderInvoicing::InvoiceOrderSystem::dst' [0..1] --- 'OrderInvoicing::Order::src' [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.625
Association 'OrderInvoicing::InvoiceOrderSystem::dst' [0..1] --- 'OrderInvoicing::Order::src' [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.625
Class 'OrderInvoicing::Stock' created.	com.concordia.read.evp	2008-05-01 10:44:1.625
Association 'OrderInvoicing::InvoiceOrderSystem::dst' [0..1] --- 'OrderInvoicing::Stock::src' [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.625
Attribute 'OrderInvoicing::Order::ID' : OrderInvoicing::int [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.625
Class 'OrderInvoicing::Product' created.	com.concordia.read.evp	2008-05-01 10:44:1.625
Attribute 'OrderInvoicing::Product::Ordered' : OrderInvoicing::String [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.625
Attribute 'OrderInvoicing::Product::ID' : OrderInvoicing::int [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.625
Attribute 'OrderInvoicing::Product::Quantity' : OrderInvoicing::int [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.625
Class 'OrderInvoicing::Integer' created.	com.concordia.read.evp	2008-05-01 10:44:1.625
Association 'OrderInvoicing::Product::dst' [0..1] --- 'OrderInvoicing::Integer::src' [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.625
Association 'OrderInvoicing::Product::dst' [0..1] --- 'OrderInvoicing::Integer::src' [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.640
Class 'OrderInvoicing::Result' created.	com.concordia.read.evp	2008-05-01 10:44:1.640
Class 'OrderInvoicing::StockManagementSystem' created.	com.concordia.read.evp	2008-05-01 10:44:1.640
Association 'OrderInvoicing::StockManagementSystem::dst' [0..1] --- 'OrderInvoicing::Stock::src' [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.640
Association 'OrderInvoicing::StockManagementSystem::dst' [0..1] --- 'OrderInvoicing::Result::src' [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.640
Attribute 'OrderInvoicing::Stock::Class' : OrderInvoicing::String [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.640
Attribute 'OrderInvoicing::Stock::Quantity' : OrderInvoicing::int [0..1] created.	com.concordia.read.evp	2008-05-01 10:44:1.640
Class 'OrderInvoicing::ShipmentManagementSystem' created.	com.concordia.read.evp	2008-05-01 10:44:1.640

Figure 7.3: EVP's logging through Eclipse error log view.

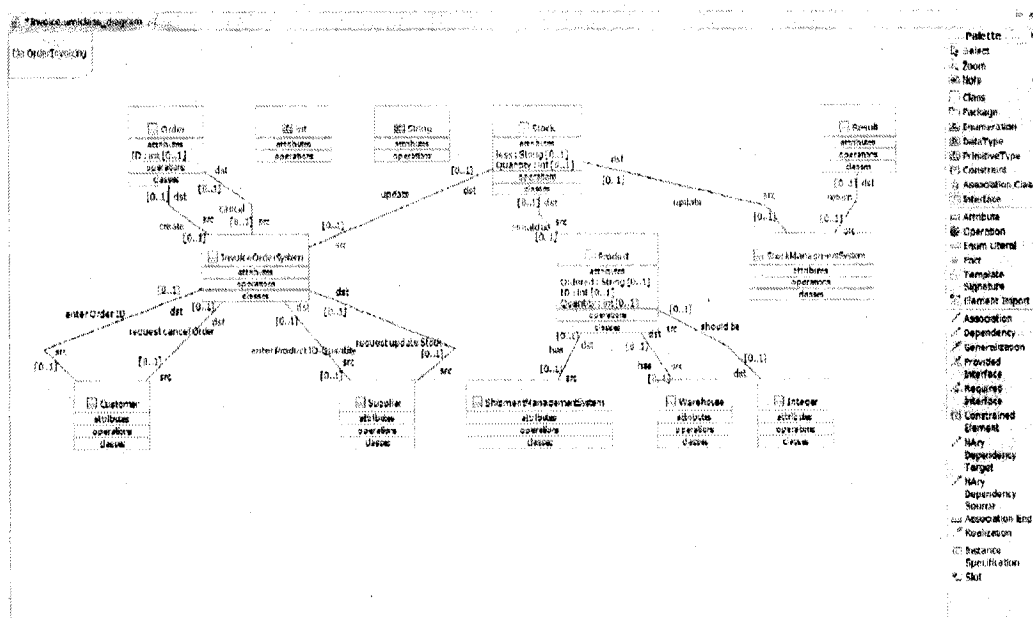


Figure 7.4: SM visualized as class diagram.

# Chapter 8

## Experimental Work

### 8.1 Experiment

The mapping rules described in chapter 5 were mainly extracted through the direct observation of sample requirement texts, therefore we decided to devise a strategy to evaluate our methodology properly on a controlled experiment. Our approach is similar to the one in [8]. The main goal of the controlled experiment was to find out how close the conceptual models developed using our methodology are to those developed by RE experts in comparison with students' models, all developed from the same textual user requirements. Another goal of this experiment was to establish the limitations of our methodology with the purpose of improving it and outlining the future work directions of this research.

### 8.1.1 Procedure

In our experiment, the same invoicing system description (see section 6.1) was distributed among experts and students. Two experts created the corresponding context use case diagrams. They were asked to focus on the summary-level use cases. The intersection of these models served as a benchmark for validation purposes. One expert was assigned the responsibility of creating a domain model for the same text. Our experts had in-depth knowledge of modeling the requirement as well as industrial experience.

The case study was also given to five graduate students in software engineering program at Concordia University with a good knowledge of use-case modeling, and four graduate students with acceptable level of knowledge on domain modeling. As a result, use-case diagrams and domain models for our experiment were developed. The students' diagrams were analyzed carefully and summarized based on the average number of correct and incorrect choices of actors, use cases and their communications in the use case diagrams. In case of domain model, only the concepts were counted but not the relations. This is justified by the fact that the relations between the concepts cannot be strictly defined and categorized and the way they are defined is highly dependent upon the human analyst who models them. It should be noted that the difficulties related to the evaluation of the models had also been presented in [14].

Next, the students' diagrams and the diagrams developed using our methodology were compared with the experts' models. In the case of use case diagrams actors, use cases, and communications were considered equal if they were playing the same

role, achieving the same goal and establishing the same relationship between the same actors and use cases respectively, and their names matched exactly. They were considered equivalent if everything was equal, as defined above, but the names of the roles or use cases were different (e.g., publisher (in the case study text) played the role of the supplier (proposed by the ECC model) in this system, however they did not have the same name). Finally, the actors, the use cases, and their communications were considered different if they were either incorrect or added extra (but valid) information. An element was *incorrect* if it did not exist in the expert model, it was considered wrong based on our common sense, and it was classified as *extra* if it was correct but not stated in the expert model. Once again, the decision was based on our own judgment. The same categories existed for the concepts in the domain model.

### 8.1.2 Discussion

***Context Use Case Model Validation Result and Conclusion.*** The validation results are summarized in Table 8.1.

For instance, 16.66% of the actors automatically identified by the READ methodology were equal, 33.33 % were equivalent, and the other 50 % was extra but valid when compared to the expert model. As for the students, 86.95% of the actors identified were correct and the other 13.04 % was incorrect.

We concluded from this experiment that interaction with the analyst is definitely needed in order to permit acceptance and modification of the actors once the primitive list has been automatically proposed by the READ using the ECC model.

Table 8.1: Validation result (extracted information perspective).

Actor				
	Equal	Equivalent	Incorrect	Extra
READ	16.66%	33.33%	0	50%
AVG-Students	86.95%	0	13.04%	0
Use Case				
READ	100%	0	0	0
AVG-Students	31.11%	0	68.88%	0
Communication				
READ	100%	0	0	0
AVG-Students	38.98%	0	61.01%	0

The READ methodology was better at identifying the summary-level use cases and communications, whereas human analysts tend to extract incorrect use cases which are actually considered as steps for other use cases.

It was also important to know the percentage of information that was missing from the READ result as compared to the expert models. The results of this comparison are shown in Table 8.2. For example, 6.66% of the use cases identified by the experts were missing from the average student' models, and 20% of the actors extracted by the experts from the text were missing from the READ result.

We concluded that READ was better than the human analysts at identifying the use cases and the communication links between the actors and the use cases.

In none of the cases were the READ results incorrect. Moreover, READ helped identify extra information undetected by the analysts. Missing information was reported in the list of actors, but the pre-approval of the actor list by the analyst would eliminate this deficiency.

Table 8.2: Validation result (missing information perspective).

Missing Actor	
READ	20%
AVG-Students	8%
Missing Use Case	
READ	0
AVG-Students	6.66%
Missing Communication	
READ	0
AVG-Students	28.57%

***Domain Model Validation Result and Conclusion.*** The validation results are summarized in Table 8.3.

For instance, 83.3% of the concepts automatically identified by the READ methodology were equal, 5.5% were equivalent, and the other 11.11% was extra but valid when compared to the expert model. As for the students, 100% of the concepts identified were correct.

Regarding the identified concepts, we concluded from this experiment that READ

Table 8.3: Validation result (extracted Concept perspective).

	Concepts			
	Equal	Equivalent	Incorrect	Extra
READ	83.3%	5.5%	0	11.11%
AVG-Students	100%	0	0	0

and students almost perform the same except that there existed over-specification in the models developed by READ. It should be noted that over-specification in this context is used to refer to the extra correct information that is not in the expert model [14]. As stated in [14, 21, 26, 27], over-specification is more preferable than losing information in the requirement analysis of the software development life cycle.

It was also important to know the percentage of information that was missing from the READ and students results as compared to the expert models. The results of this comparison are shown in Table 8.4. For example, 53.125% of the concepts identified by the experts were missing from the average student's models, and none of the concepts were missing from the READ model. We concluded that READ model is closer in quality to the experts model in terms of completeness of the identified concepts because none of the concepts are missing. Moreover, READ helped identifying extra information undetected by the analysts.

The results of the controlled experiments designed to evaluate the approach proved the validity and feasibility of the methodology.

In the next chapter, our achievements in this research as well as the limitations



Table 8.4: Validation result (missing Concept perspective).

Missing Concept	
READ	0
AVG-Students	53.125%

of our approach are discussed.

# Chapter 9

## Discussion

Our research addressed specific challenges which existed in the effort to gain an understanding of the stakeholders' real-world needs and desires with the purpose of providing the correct software solution. The solid formal basis of the linguistic structure of ROM makes it possible to automatically build the SM and CUCM of the system which is considered as an important advantage. The outcomes of our work can be used by software analysts in their in-depth study of requirements text. Our achievements not only constituted a proof of concept for practical use, but also introduced a basis for future research in this area. We will summarize our contributions in this chapter.

In our work, we also faced many challenges that we had to overcome; we improved and evolved our methodology with new ideas and enhancements to endure these obstacles and, finally, satisfied our objectives. However, we are aware that the proposed methodology is not free from limitations. The limitations of our approach are also discussed in this chapter.

## 9.1 Intentions and Contributions

The aim of this research was to propose an elaborate methodology which constituted a proof of concept for the idea that a conceptual knowledge on the software to be developed can be acquired through a semi-automated process, with textual requirements document as input and UML diagrams representing its structure (SM) and high-level contextual view on the software system's actors and services CUCM, as outputs. Another objective was to build a prototype tool for visualizing these models and validate the models generated automatically against the ones generated by software engineers (students and experts in the area). The following is the list of outcomes we achieved and the outline of our contributions.

As specified before, our problem statement mainly stemmed from the gap between the stakeholders' perception of their needs and how these needs are described textually, whereby the descriptions can be misinterpreted by analysts due to the inherent ambiguity of the natural language itself.

- (i) We proposed a solution for reducing this gap from an innovative point of view, namely the use of automatic conceptual knowledge extraction from user requirements text and its visualization, and injection of domain-related missing information provided by ECC models where necessary.
- (ii) We devised rules for transforming the formal internal presentation of the text (ROM) to two structural views (FCSM and ISM), each with different level of details [39].

- (iii) We designed rules for extracting CUCM elements, such as actors and use cases, and associations between them from the requirement text [38,40].
- (iv) We defined rules for constructing the comprehensive model for specific domain (ECC models) from the standard pre-built Data Models. The ECC models can be reused and improved over time [39,40]. Using them reduces the cost of development because they work as an expert.
- (v) We applied the knowledge included in the ECC model for automatically identifying the actors for the CUCM and improving the SM generated directly from the text [38,40]. An improved SM model of the system represents its structure with more details.
- (vi) We developed a prototype tool in support of the methodology [40]. The importance and benefits of such a tool are obvious. It would save interview time, serve as a means to proofread the requirements, and facilitate communication between the stakeholders who provide the requirements and the software analysts who have to understand them correctly and clearly. This will help analysts to avoid errors in RE phase and their subsequent propagation to the design and implementation phases. As it is mentioned in [23], the cost of fixing an error/or mistake during Implementation phase is 100% to 200 % more than fixing it during the Requirement phase.
- (vii) We designed controlled experiments to evaluate our approach. The outcomes of our experiments proved the validity and feasibility of our methodology because

in none of the cases the READ results were incorrect. Moreover, the READ helped identifying extra information undetected by the analysts.

## 9.2 Limitations

The following is a list of the limitations we encountered during our work:

- (i) Creating the ECC model for each of the specific domains and constructs (e.g., Invoicing, Ordering, etc.) requires certain effort. However, once these models are built they can be reused and their real benefit will become evident in the long term. Furthermore, reusing the existing data models is expressed as one of the classical strategies for identifying conceptual classes [21].
- (ii) The proposed methodology is using the subset of the ROM functionalities thus it is bounded and limited to the set of patterns which are presented in Fig. 9.1. It should be noted that ROM is a work in progress and further information can be acquired from [7].
- (iii) Our conceptual analysis for the ISM is bounded by the invoicing system. However, FCSM can be developed for every domain.
- (iv) Currently, our experience with the approach has mainly been in academic projects' context. Therefore, the scalability of the approach is yet to be determined through larger, real-world case studies.

Finally, we would like to point out that, as Ryan concluded in [37], it is clear from a review of the history of NLP in RE that building a system which will automatically

Pattern#	Sentence Structure	ROM Representation
Pattern 1	Subject + intransitive verb	
Pattern 2	Subject + linking verb + subject complement	
Pattern 3	Subject + transitive verb + direct object	
Pattern 4	Subject + transitive verb + indirect object + direct object	
Pattern 5	Subject + transitive verb + direct object + object complement	<p>Note: The connection 1 can be 'to', 'for', or nothing.</p>

Figure 9.1: Supported sentence patterns.

explain stakeholders' needs is an unrealistic objective. However, considering that the elicitation of user requirements is a dynamic and social process, NLP techniques can assist the analyst in this process, but tools such as READ would not replace the analyst's role in RE.

The conclusions and guidelines for our future research work are outlined in the next chapter.

## Chapter 10

# Conclusion and Future Work

The success of a software project largely depends on the quality of the user requirements documentation, which serves as an input to the design, coding, and testing phases. Software requirements are documented mostly in natural language. Thus, the textual user requirements should be well written and completely understood by both stakeholders and software analysts.

This thesis addressed the problem of semi-automatically assisting the software analysts in the requirement elicitation and analysis activities. First, it proposed a methodology for conceptual knowledge mining semi-automatically from the requirements text and for providing domain expertise semi-automatically by using pre-defined domain-oriented data models. A conceptual SM was then generated with the help of a set of pre-defined rules. Then, an ISM was generated by injecting domain-related information provided by ECC models, which are extracted from reusable domain-specific data models. In the second part of the methodology, the focus was in semi-automatically assisting the software analyst in the early stages of the use case

model elicitation process. The same ECC was used to help identify the actors that may exist in each domain for generating a CUCM. The elements of the CUCM were extracted from the text according to the devised rules and the corresponding CUCM diagram was generated to graphically visualize the identified actors and system services provided to those actors. Finally, each sentence from the requirements text was assigned to exactly one use case description with the help of a metrics-based text partitioning algorithm. In our approach, whenever the information seems incorrect, doubtful or unclear, it would be highlighted and flagged to the analyst for further feedback.

The work presented in this thesis was motivated by the industrial importance of correctly documenting requirements and of reducing the costs inherent in identifying the underlying requirements problems and potential system solutions.

We illustrated the approach on a case study. We also briefly described a tool which implements the proposed methodology and discussed the results of the evaluation.

For our future work in the context of the READ project we will mainly focus on extending the predefined rules for identifying concepts, relations, actors, and use cases from the formal ROM presentation to the point where we can possibly handle less structured sentences and more unrestricted natural language. We also intend to test and improve the current rules using statistical NLP techniques. Preferably, the components of the READ tool should not only work in stand-alone mode, but be capable of incorporation with each other which would increase the applicability of the tool.

We will analyze the possibility of creating the ECC model for other constructs ap-



plicable to the wider ranges of application and organizations. More robust evaluation of the methodology in terms of the performance of the system and completeness and accuracy of the generated models (Precision and Recall) is also required; this should be ideally evaluated on systems already developed by different kinds of organizations and evolving studies with large number of requirements.

We are currently investigating means for visualizing the NFRs that are automatically extracted and classified from text [17]. Specifically, NFRs will be highlighted to increase their visibility and will be explicitly linked to the corresponding functional requirements. NFRs will be integrated within the SM and CUCM.

We believe that such automated assistance would be very beneficial to RE.

# References

- [1] R. J. Abbott, "Program design by informal english descriptions," *Communications of the ACM*, vol. 26, no. 11, pp. 882–894, 1983.
- [2] A. Abran, J. W. Moore, P. Bourque, and R. Dupuis. (2004) Guide to the Software Engineering Body of Knowledge (SWEBOK). Online HTML format. [Online]. Available: <http://www.swebok.org/htmlformat.html>
- [3] V. Ambriola and V. Gervasi, "Processing natural language requirements," in *Proc. 12th IEEE International Conference on Automated Software Engineering (formerly: KBSE) (ASE'97)*, Incline Village, Nevada, USA, November 1997, pp. 36–45.
- [4] G. Booch, "Object-oriented development," *IEEE Transactions on Software Engineering*, vol. 12, no. 2, pp. 211–221, 1986.
- [5] Z. Chen, "Formalization and classification of product requirements using axiomatic theory of design modeling," Master's thesis, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, 2006.
- [6] A. Cockburn, *Writing Effective Use Cases*. Boston: Addison- Wesley, 2001.

- [7] (2008) Design lab website. [Online]. Available: <http://users.encs.concordia.ca/-zeng/research.html>
- [8] I. Diaz, L. Francisca, A. Matteo, and O. Pastor, "Integrating natural language techniques in oo method," in *Proc. 6th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing'05)*. New York, NY, USA: Springer Berlin/Heidelberg, 2005, pp. 560–571.
- [9] I. Diaz, F. Losavio, A. Matteo, and O. Pastor, "A specification pattern for use cases," *Information and Management*, vol. 41, no. 8, pp. 961–975, 2004.
- [10] J. Drazan and V. Mencl, "Improved processing of textual use cases: Deriving behavior specifications," in *Proc. 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'07)*, Harrachov, Czech Republic, January 2007, pp. 856–868.
- [11] Eclipse Modeling Framework. Eclipse Foundation. [Online]. Available: <http://www.matthewwest.org.uk/Documents/princ03.pdf>.
- [12] Graphical Modeling Framework. Eclipse Foundation. [Online]. Available: <http://wiki.eclipse.org/MDT-UML2Tools>
- [13] C. Fellbaum. (1998) Wordnet: An electronic lexical database. [Online]. Available: <http://wordnet.princeton.edu/perl/webwn>
- [14] H. Harmain and R. Gaizauskas, "CM-Builder: An automated NL-based case tool," in *Proc. 15th IEEE International Conference on Automated Software Engineering (ASE'00)*, Grenoble, France, September 2000, pp. 45–53.

- [15] M. Himsolt, "Graphed.: An interactive graph editor," in *Proc. 6th Annual Symposium on Theoretical Aspects of Computer Science (STACS'89)*. New York, NY, USA: Springer-Verlag New York, Inc., February 1989, pp. 532–533.
- [16] I. Hussain, O. Ormandjieva, and L. Kosseim, "Automatic quality assessment of srs text by means of a decision-tree-based text classifier," in *Proc. 7th International Conference on Quality Software(QSIC'07)*, Portland, Oregon, USA, October 2007, pp. 209–218.
- [17] I. Hussain, O. Ormandjieva and L. Kosseim, "Using linguistic knowledge to improve the detection of non-functional requirements specifications," in *Proc. 13th International Conference on Applications of Natural Language to Information Systems (NLDB'08)*, London, UK, June 2008, pp. 287–298.
- [18] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std. 610.12, 1990.
- [19] M. Jackson and P. Zave, "Domain descriptions," in *Proc. IEEE International Symposium on Requirements Engineering (RE'93)*, San Diego, CA, U.S.A, January 1993, pp. 56–64.
- [20] J. Kim, S. Park, and V. Sugumaran, "Improving use case driven analysis using goal and scenario authoring: A linguistics-based approach," *Data and Knowledge Engineering*, vol. 58, no. 1, pp. 21–46, 2006.

- [21] C. Larmann, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, 3rd ed. Upper Saddle River, N.J.: Prentice Hall PTR, 2005.
- [22] B. Lee and B. R. Bryant, "Automated conversion from requirements documentation to an object-oriented formal specification language," in *Proc. ACM symposium on Applied computing (SAC'02)*, 2002, pp. 932–936.
- [23] D. Leffingwell and D. Widrig, *Managing Software Requirements: A Unified Approach*. Boston, MA, USA: Addison-Wesley, 2003.
- [24] D. Liu, K. Subramaniam, B. Far, and A. Eberlein, "Automating transition from use-cases to class model," in *Proc. IEEE Canadian Conference on Electrical and Computer Engineering (CCECE'03)*, Montreal, Canada, May 2003, pp. 831–834.
- [25] C. D. Manning and H. Schtze, *Foundations of Statistical Natural Language Processing*. Cambridge, Mass.: The MIT Press, June 1999.
- [26] J. Martin and J. Odell, *Object Oriented Methods: A Foundation*. Englewood clifs, New Jersey: Prentice Hall, 1995.
- [27] B. Meyer, *Object Oriented Software Construction*, 2nd ed. Upper Saddle River, NJ: ISE Inc., 1997.
- [28] F. Meziane and S. Vadera, "Towards automatic modeling of requirements," *Malaysian Journal of Computer Science*, vol. 9, no. 2, pp. 1–13, 1996.

- [29] L. Mich, "NL-OOPS: From natural language to object oriented requirements using the natural language processing system LOLITA," *Natural Language Engineering*, vol. 2, no. 2, pp. 161–187, 1996.
- [30] L. Mich, M. Franch, and N. I. Pierluigi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering*, 2004, vol. 9, no. 1, pp. 40–56, February.
- [31] A. M. Moreno, "Object-Oriented analysis from textual specifications," in *Proc. 9th IEEE International conference on Software Engineering and Knowledge Engineering (SEKE'97)*, Madrid, Spain, January 1997, pp. 290–294.
- [32] J. Mylopoulos. Structured analysis and design technique (SADT). [Online]. Available: <http://www.cs.toronto.edu/~jm/2507S/Notes04/SADT.pdf>
- [33] P. M. Nugues, *An Introduction to Language Processing with Perl and Prolog: An Outline of Theories, Implementation, and Application with Special Consideration of English, French, and German (Cognitive Technologies)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
- [34] J. N. och Dag, "Managing natural language requirements in large scale software development," Ph.D. dissertation, Lund University, Lund, Sweden, 2005.
- [35] S. P. Overmyer, B. Lavoie, and O. Rambow, "Conceptual modeling through linguistic analysis using LIDA," in *Proc. 23rd International Conference on Software Engineering (ICSE'01)*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 401–410.

- [36] H. G. Perez-Gonzalez and J. K. Kalita, "GOOAL: A graphic object oriented analysis laboratory," in *Proc. 17th ACM/SIGPLAN Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'02)*. New York, NY, USA: ACM, 2002, pp. 38–39.
- [37] K. Ryan, "The role of natural language in requirements engineering," in *Proc. IEEE International Symposium on Requirements Engineering (RE'93)*, San Diego, CA, USA, January 1993, pp. 240–242.
- [38] S. Moradi Seresht and O. Ormandjieva, "Automatic assistance for use case elicitation from user requirements text," in *Proc. 11th Workshop on Requirements Engineering (WER'08)*, Barcelona, Spain, September 2008.
- [39] S. Moradi Seresht and O. Ormandjieva, "Towards automatic diagnostic of conceptual problems in requirements text," in *Proc. 2008 International Conference on Software Engineering Theory and Practice (SETP'08)*, Orlando, Florida, USA, July 2008, pp. 105–114.
- [40] S. Moradi Seresht, O. Ormandjieva and S. Sabra, "Automatic conceptual analysis of user requirements with the requirements engineering assistance diagnostic READ tool," in *Proc. 6th International Conference on Software Engineering Research, Management and Applications (SERA'08)*, Prague, Czech Republic, August 2008, pp. 133–142.
- [41] M. Saeki, H. Horai, and H. Enomoto, "Software development process from natural language specification," in *Proc. 11th International Conference on Software*

*Engineering (ICSE'89)*, May 1989, pp. 67–73.

- [42] M. Sighireanu and K. J. Turner. Requirement capture, formal description and verification of an invoicing system. [Online]. Available: <http://www.inrialpes.fr/vasy/Publications/Sighireanu-turner-98.html>
- [43] L. Silverston, *The Data Model Resource Book (Volume 1)*. New York, NY, USA: Wiley Computer Publishing, 2001.
- [44] S. Some, “Supporting use case based requirements engineering,” *Information and Software Technology*, vol. 48, pp. 43–58, 2006.
- [45] I. Sommerville, *Software Engineering*, eight ed. Boston, MA, USA: Pearson Addison Wesley, 2007.
- [46] M. A. Talib, O. Ormandjieva, A. Abran, A. Khelifi, and L. Buglione, “Scenario-based black-box testing in cosmic-ffp: a case study,” *Software Quality Professional*, vol. 8, no. 3, pp. 23–33, 2006.
- [47] Linguistic Engineering. [Online]. Available: <http://portal.unesco.org>
- [48] M. West. Developing high quality data models. [Online]. Available: <http://www.matthewwest.org.uk/Documents/princ03.pdf>.
- [49] K. E. Wiegers, *Software Requirements* :. Redmond, WA: Microsoft -Press, 2003.
- [50] Y. Zeng, “Formalization of design requirements,” in *Proc. 7th World Conference on Integrated Design and Process Technology (IDPT'03)*, Austin, Texas, USA, December 2003, pp. 209–218.



- [51] Y. Zeng, "Recursive object model (ROM) modeling of linguistic information in engineering design," *Computers in Industry*, vol. 59, pp. 612–625, August 2008.
- [52] Y. Zeng, "Axiomatic theory of design modeling," *Integrated Design and Process Science*, vol. 6, no. 3, pp. 1–28, 2002.