Automated Assistance for Use Cases Elicitation from User Requirements Text

Shadi Moradi Seresht and Olga Ormandjieva

Dept. of Computer Science and Software Engineering
Concordia University, Montreal, Canada
{sh_morad,ormandj}@cse.concordia.ca

Abstract

Software Requirements Engineering addresses specific challenges which exist in the effort to gain an understanding of the nature of the engineering problem arising from user's real-world needs and desires. This research is aimed at helping software analysts meet these challenges. The proposed methodology forms the basis of the automated process designed to capture the high-level system services and actors from the textual user requirements. This model 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 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 Expert Comparable Contextual (ECC) models that are extracted from reusable domain-specific data models. We illustrate the applicability of our methodology on an order invoicing case study and demonstrate it with a prototype tool. The results of the validation of our methodology prove that such a tool for assisting the elicitation of use-case models from textual requirements is feasible.

1. Introduction

The software requirements engineering (RE) process should begin with an analysis of the problem and agreement with the customer on what the software product must do. This agreement, in the form of textual user requirements, should be well written and completely understood by both customer and software analyst. In reality, customers 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 customer's problem and field of endeavor well enough to model requirements to satisfy their system needs.

The aim of this research is to provide a methodology and a supporting tool for the (semi-) automatic assistance of the user requirements analysis in the RE process. More specifically, this research addresses the challenges in the first steps toward elicitation of the Use-Case Model – a concise description of software system's actors and services recognized as one of the models most often used in coming to an agreement on the final set of requirements [8], as well as being well known as a conventional analysis method in RE [14].

Research Goal. The research goal of this paper is to generate a high-level contextual view on the software system's actors and services (Context Use-Case Model - CUCM) automatically and thus objectively from the textual user requirements. The importance and benefits of such a high-level view on the system to be developed are obvious. It would save development time, serve as a means to proofread the requirements, and facilitate communication between the users who provide the requirements and the software analysts who have to implement them. In other words, CUCM serves as a medium to communicate user requirements to the technical personnel responsible for developing the software.

Approach. Our approach combines two technologies: a formal graphical language called the Recursive Object Model (ROM) [24] and an Expert-Comparable Contextual (ECC) models extracted from the domain-specific 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. The paper targets the problem of automatic generation of CUCMs from text by: (i) applying the knowledge included in the ECC model to identify the actors; (ii) devising rules for extracting CUCM elements, such as actors and system services (or system use cases), and relating them, in addition, each sentence from the requirements text is assigned to exactly one use case with the help of a metrics-based text partitioning algorithm; and (iii) developing a prototype tool in support of the methodology to visualize graphically the CUCM.

The work presented in this paper forms part of a larger project, the Requirements Engineering Assistance Diagnostic (READ), which is aimed at applying Natural Language Processing (NLP) techniques to the RE process [13]. The authors would like to point out that, as Ryan concluded in [20], it is clear from a review of the history of NLP in RE that building a system which will automatically explain user needs is an unrealistic objective. However, considering that the elicitation of user requirements is a dynamic and social process error-prone due to the ambiguous nature of NL text, NLP techniques can assist the analyst in this process without replacing his/her role in RE.

The paper is organized as follows: section 2 reviews related work; the background required to understand the methodology is covered in section 3; the methodology is explained in section 4 and illustrated with a case study in section 5; the architecture of the tool implementing our approach is described in section 6; section 7 covers validation of the methodology; and, finally, in section 8, the conclusions are presented and directions for future work are outlined.

2. Related Work

Over the past few decades, extensive studies have been conducted in the area of applying linguistic techniques to the analysis of a requirements text. Although a wide variety of radically different case tools has been developed, common to all these approaches are a couple of basic concepts, such as relating nouns to classes, and relating adjectives to attributes [1, 4]. GOOAL (Graphical Object-Oriented Analysis Laboratory) [19], CM-Builder [11], and LIDA [18] are examples of such tools. GOOAL, presented by H. G. Perez-Gonzalez *et al.* [19], has only been tested with problems described in no more than eight sentences and an average of 100 words. CM-Builder [11] is a CASE tool which performs domain-independent object-oriented analysis. Unfortunately, it does not support any kind of dynamic diagram. Overmyer *et al.* introduced LIDA [18] with the main goal of helping the analyst in the transition from natural language text to object-oriented notation. It does not support the dynamic diagram either; moreover it requires considerable user intervention. Circle, the work of Ambriola *et al.* [3], attempts to validate NL requirements text with the help of the user after deriving a conceptual model automatically from the requirements specifications. Although Circle is in general use, it still does not consider the existence of ambiguities at the level of surface understanding. This could corrupt their model, making errors generated by it extremely difficult for a user to detect later on. In [23], Subramaniam *et al.* proposed a tool for automatic object/class identification. This case tool was developed as one of the add-ins of Rational Rose, and has two main functionalities: 1) use-case realization; and 2) class diagram generation. In [10], the method for processing textual use cases and extracting their behavioral aspects based on linguistic techniques is suggested. For his part, Some [22] developed a tool called UCEd with the aim of providing the framework for use case edition, clarification and finally developing the "executable specification integrating the partial behaviors of the use cases" in the form of the state machines. Therefore, the methodology reported in [22] 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 paper differs considerably from the related work in that our methodology is applicable in the early phases of RE and is meant to aid the analysts in requirements elicitation and analysis activities. Moreover, it is founded on a formal representation of the text and not on the parts-of-speech technique ([12, 13]), which allows for an automatic CUCM generation from text, which in turn would allow the number of costly human errors in the RE process to be reduced considerably. In addition, we bring an automatic expert assistance into the process by integrating into the process the ECC models containing domain-specific data. Details of the ROM and ECC models are provided next.

3. Background

The purpose of this section is to briefly introduce the formal representation of the syntactical structure of text with the ROM [5, 6, 25, 26] and to explain the ECC models extracted from domain-specific data models.

3.1 Recursive Object Model (ROM)

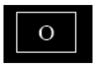
Our choice of formal representation is justified by the fact that the ROM has proven sufficient to represent the technical English text used in software engineering documents, where only statements are involved. The ROM was initially developed in [6] and further refined in [5, 25, and 26].

Mathematical Foundation. The linguistic structure's formal representation has been confronted with the fundamental mathematical and philosophical challenges of uniting two contrasting concepts. The axiomatic theory of design modeling [25] provides a solution to this problem based on the rigorous mathematical concepts on which the ROM has been developed for representing text [6, 25].

Graphical 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 [6, 25, and 26].

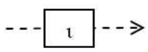
The ROM uses only five basic symbols to represent an object, a composite object, constraint, connection and predicate relations.

A word surrounded by a solid-line box represents a concrete entity corresponding to a noun in English:

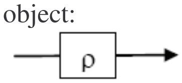


A constraint is a descriptive, limiting, or particularizing relation of one object to another:

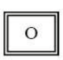
The connection relation (ι), represented by a dashed arrow, connects two objects which do not constrain one another:



The predicate relation (ρ), represented by a solid arrow, is the relation that describes an action of one object on another or that describes the state of an object:



A double solid line box represents a composite object, which consists of other kinds of objects:



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 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 a means to discover the high-level services of the system. In our methodology, the formal model is employed to bridge the textual user requirements and the high-level contextual view (CUCM) on the software system's actors and services.

3.2 Expert-Comparable Contextual (ECC) Model

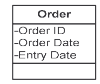
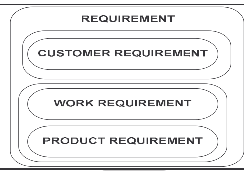
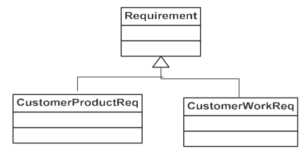
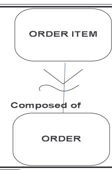
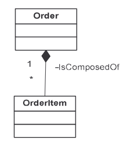
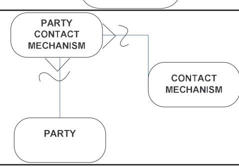
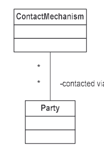
Generally, data models are developed to achieve a particular goal and to highlight the important features of something, considering that specific goal [30]. While reviewing the literature, we came across interesting evidence which supports the resemblance between the data models and the conceptual classes. For example, in [15], one of the strategies for identifying conceptual classes has been identified as the reuse of the universal data models. These models describe the structure of the data as well as their meaning, and data modeling is recognized as a standard for designing databases [21]. Our analyses reveal 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 intended to construct a comprehensive model for specific domains and further contemplate their potential for assistance in automatic generation of the CUCM of the system. 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. 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 [21], some of the constructs applicable to most organizations are: people and organizations, products, product ordering, shipping, work effort, invoicing, accounting and budgeting, and human resources. The process of deriving ECC from the data model is outlined below:

- Acquisition of the standards and conventions used in Data Modeling
- Developing the list of heuristics that can be used for mapping data models to the domain 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

Table 1 lists some of the transformation rules which map data models to ECC models, and Figure 1 shows an ECC model for the Invoicing System (attributes are omitted from the figure to increase readability). It is worth mentioning here that the ECC models are developed manually for each domain and can be later reused.

Building an ECC model obviously requires effort; however, once built, these models can be reused and either extended or modified, or both, revealing their potential for generating real, long-term benefits. The influence of these models on the improvement and completeness of the software domain model [15], such as useful information added about the relationships between concepts (e.g. generalization and composition), has been addressed in [17]. In this paper, we make use of the ECC in the Context Use-Case Modeling of the system. The details of our proposed methodology are described in the following section.

Table 1. Mapping Data Models to ECC Models

Data Model		Structural Model	
Naming Standard	**Diagramming Conventions**	**Naming Standard**	**Diagramming Conventions**
Entity & Attributes		**Concept &** Attributes	
Non-mutually exclusive Subtypes & Supertypes		Inheritance that includes all possible combinations	
One-to-Many Relationship (composed of)		**Aggregation**	
Many-to-Many Relationship		**Relation with many-to-many multiplicity**	

4. Methodology

This section describes an elaborate methodology which constitutes a proof of concept for the idea that a CUCM can be acquired through an (semi-) automated process, with a requirements text as input and a CUCM diagram representing the actors and services (use-cases), as well as summary-level use-case textual descriptions [7], as outputs.

Use case identification can be done at different levels, such as business/interaction [14], or with different scopes, such as functional/design [7]. In the requirements elicitation phase, business services are initially captured at a higher abstraction level as "summary-level use cases" [7]. These are further refined into functional or design user-goal use cases. 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 [7, 15].

Our goal is to identify the use cases within the scope of business/interaction, which is defined as the services provided by the system to the user. The steps of our methodology for 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;
Step 5: Draw a Context Use Case Diagram which depicts graphically the CUCM.
The above 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 on the user-goal use cases, 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. Hence, "If" clauses are frequent in user-goal use cases scenarios descriptions. In our methodology, the "if" clauses are visualized in the form of UML notes and may later be refined by the analysts to user-goal use cases.

4.1 Actors

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

Primary Actor. Each ECC model accommodates the possible roles played in that specific domain (e.g. customer and supplier in invoicing) 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 systems for an enterprise: building them individually or developing them from the perspective that an enterprise system is an enterprise-wide framework where the systems can collaborate (in other words, a holistic approach to system development) [21]. We are interested in the second approach.

Modeling the interrelationship between these systems makes it possible to automatically elicit the supporting systems associated with the system under development (SUD) as potential actors to support its services. In order to provide the flexibility to take account the customized needs of SUD stakeholders, which do not exactly match the standard domain models (ECC models), both these lists will be shown to the user for his/her approval and modification, if required.

4.2 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 (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 towards 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 or 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, relation directed from the SUD, entity towards which the relation is terminated> 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 (use case). This triggering event is normally stated in the requirements text using verbs such as request, ask, etc. If 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 omitted. In completing the use case identification process, we will study the relations that originate from the primary actors and are directed toward another entity. If an entity which is the target of a terminated relation is found on the predefined list, the relation and the entity will be ignored because this list contains some of the keywords, such as ID, username, password, etc., which are normally used in type (d) interactions. If the entity is not on the list, we scan the original text sentence by sentence, looking for tuples of the type <primary actor, relation directed from the actor, entity towards which the relation is terminated, to (for), system >. We are seeking the relations that are directed toward the system entity with the prepositional relations "of" 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.

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. High-level system services are usually described in narrative style and are referred to as "summary-level use cases".

4.3 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 the *SS*, where each partition groups the sentences related to one service (use case) in one equivalence class. The equivalence criterion is the rule for evaluating the closeness of a sentence to S_i. Such a grouping increases the visibility of a service in the text describing it, 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 were the space points are abstractions of sentences.

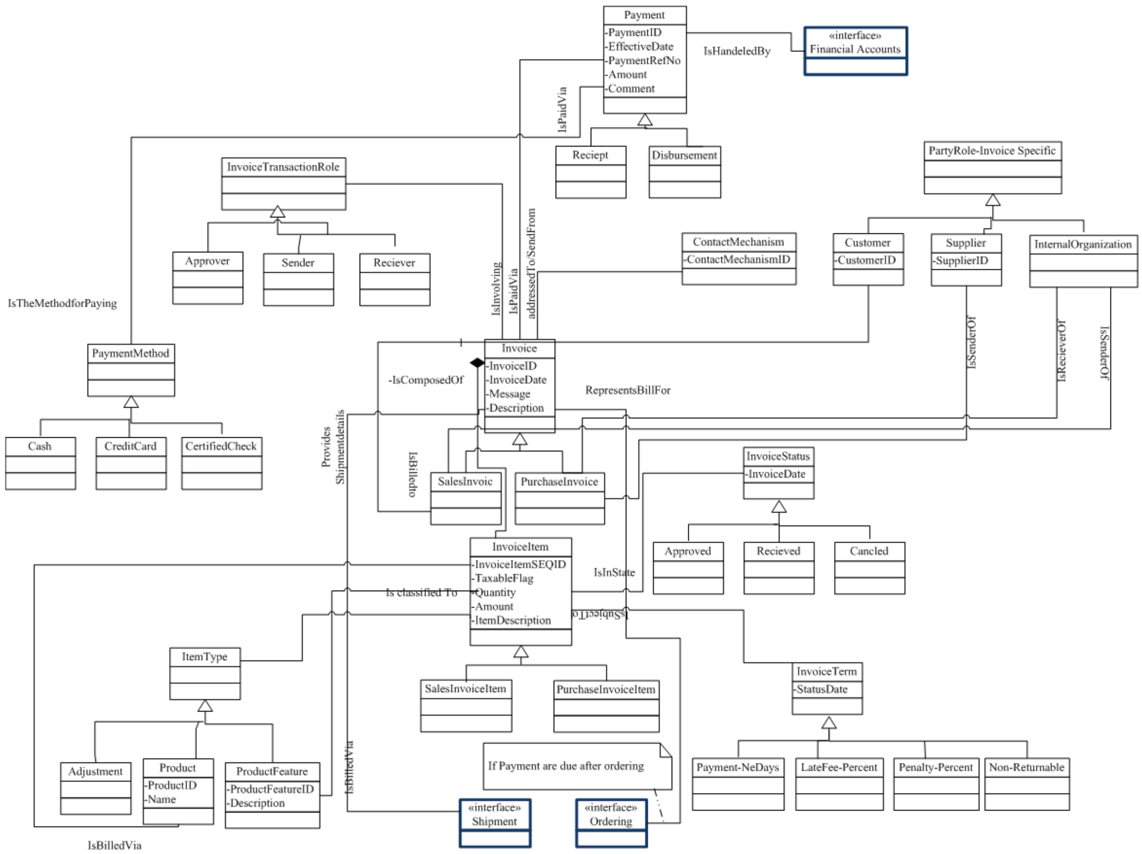


Figure1. ECC Model for an invoicing system

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 equivalence classes UCS_1, one class for each sentence S_i in the *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 an equivalence class UCS_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:

$$sd\ (S_1, S_2) = similarity\ (S_1, S_2) * dissimilarity\ (S_1, S_2)$$

It should be noted that a similar approach was originally proposed in [2] for a metric-based test case partitioning algorithm. The *similarity* (S_1, S_2) was redefined to adapt the formula to the 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, S_2, each of which contains the significant words in the corresponding sentences meaning modals, auxiliary verbs, determiners and etc. are ignored. 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 or "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, S_2 is defined. The first table (see, for example, Table 2(a)) contains actors and actions, and the second table (see Table 2(b)) 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:

$$Similarity\ (S_1, S_2) = 2^{-C\ (S1,\ S2)}$$

where C 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.

The dissimilarity measure between two binary strings representing S_1 and S_2 is calculated as the number of elementary transformations or in other words number of words that should be changed in order to transform string S_1 into string S_2 in Table 2. The more the set of objects 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. 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 2(a) and 2(b), is calculated as $sd\ (S_1, S_2) = 2^{-2} * 2 = 0.5$.

Table 2. Measuring Distance

(a) Calculating Similarity

	customer	request	place
S1	1	1	1
S2	1	0	1

(b) 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 formula. Each sentence is placed in one of the equivalence classes from which its distance is minimal. If the shortest distances are equal, the algorithm calculates the distance between that specific sentence and the rest of the sentences in each chosen equivalence class; the sentence is finally added to the class UCS_i from which its distance is minimal. The sentences corresponding to the set of binary strings in the UCS_i are then recovered, and the use case summary is generated and shown to the user.

4.4 Supporting Actors

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.

4.5 Use Case Context Diagram

The Context Use Case Diagram is generated from knowledge of the CUCM elements (actors, use cases, and their communications) extracted in the steps outlined above. A sample Context Use Case Diagram is shown in Figure 3.

4.6 Discussion

As mentioned earlier, user requirements text is normally written in NL. Writing style and the terminology used for describing the problem are highly dependent on the individuals who record them [14]. 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 [21]. They form the dictionary of terms which can be used in writing the user requirements, and authors 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 [7, 14] 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 section.

5. Illustration

In this section, we illustrate the proposed technique on the following minimal description, inspired by the Order Invoicing System problem statement for a fictitious company, CBM Corp [29]:
"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 purchase order to the publisher. 5. The CBMSys

receives the order invoice 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. 7. The CBMSys prepares the payment for the publisher 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. 11.The customer shall view the order status from the CBMSys.12. The customer enters the order ID into the CBMSys, and CBMSys shows the order status. 13. Customer and publisher shall be able to update their personal profile from the CBMSys."

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

Step 1: Actors. The primitive list of actors generated for the user contains: customer, supplier, and general organization as the primary actors, and shipment, ordering, and financial account as the supporting actors. With the help of the user, this list is refined to customer and publisher as primary actors, and accounts receivable, account payable, and customer persistent storage as supporting actors.

Step 2 System Services. The overall formal presentation of the text describing the system is shown in Figure 2.

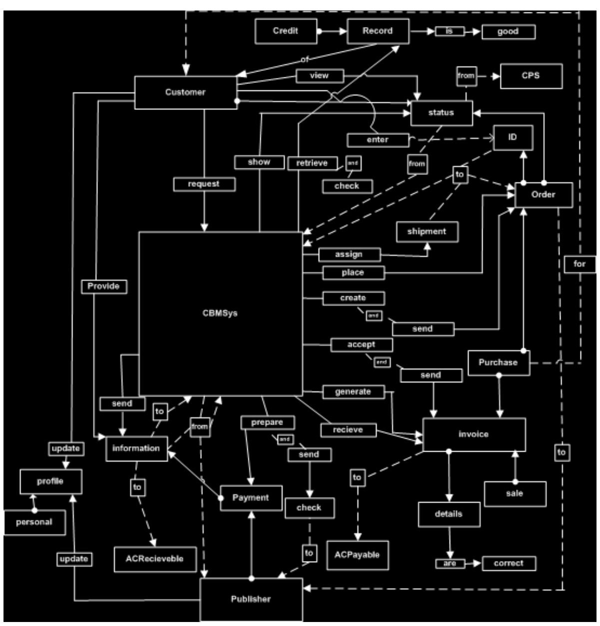


Figure 2. Formal Presentation of the text

From this formal presentation, all the relations that are rooted in primary actors and the entity toward which the relation is directed, and also the relations that stem from the "system" entity plus the entity toward which they are directed, are extracted as potential use cases of the system. Tables 3 and 4 give a few samples of potential use cases extracted from the ROM presentation. For the former group of potential use cases, each <primary actor, trigger, relation, entity> 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, the communication is ignored.

For example, <customer, request, place, order> is valid because the sentence that carries all these keywords exists in the original problem statement as "*The customer requests the CBMSys to place an order.*" In contrast, <customer, request, prepare, invoice> is not valid because none of the sentences in the original problem statement contains all these keywords. As for the latter group of potential use cases, every tuple consisting of <primary actor, relation, entity, to (from), system> is aligned with the problem statement, as a result of which "update profile" is identified as a valid use case associated with both the customer and the publisher, and "check status" which is related to the customer, whereas "enter ID" is ignored because ID is one of the keywords on the predefined list. The sentences chosen for the *SS* are: {1, 11, 13}.

Table 3. Finding Use Cases from a Formal Presentation (System perspective)

Actor	Trigger	Sample Potential Use case
customer	Request	show status
		prepare payment
		place order

Table 4. Finding Use Cases from a Formal Presentation (Actor perspective)

Actor	Sample Potential Use case
customer publisher	update profile
	enter ID
	check status

Step 3: Summary-level use case briefs. The first step of the metric-based text-partitioning algorithm results in three UCS_j sets, namely {1, 2, 3, 4, 5, 6, 8, 9, 10}, {11, 2,12}, {13,7}, corresponding to the sentences 1, 11, and 13 from the *SS* and the use cases "Place order", "Check Status", and "Update Profile". As can be seen, sentence 2 belongs to both UCS_1 and UCS_2 because its

distance from sentences 1 and 11 is the same. In the next step, the distances between sentence 2 and the rest of the sentences in both UCS_1 and UCS_2 are calculated, and it is concluded that the distance between sentence 2 and UCS_1 is the shorter than the distance between sentence 2 and UCS_2; therefore, sentence 2 will be omitted from UCS_2. The final equivalence classes are UCS_1 = {1, 2, 3, 4, 5, 6, 8, 9, 10}, UCS_2= {11, 12}, UCS_3= {13, 7}. All the sentences are correctly assigned to their corresponding equivalence class, except sentence 7 which was wrongly assigned to UCS_3; in reality, it belongs to UCS_1. The reason might be originated in the definition of similarity criteria. The refinement of the distance metric definition will be tackled in our future work.

Step 4: Supporting Actors. The supporting actors identified from the UCS_1 are CPS and the Accounts Receivable System. This shows that there exists a communication between these two supporting actors and the corresponding use case "place order". No supporting actors are identified in UCS_2 or UCS_3.

Step 5: Context Use Case Diagram. The graphical model consolidating the above information is shown in Figure 3.

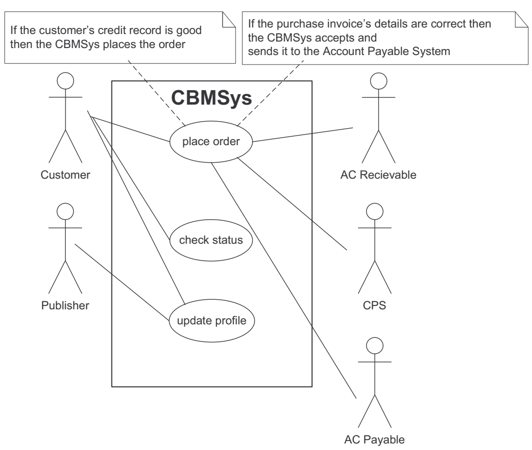


Figure 3. Context Use Case Diagram

Having presented the methodology to prove the concept, we now introduce the prototype tool.

6. READ Tool

The automated visualization of the contextual model from the requirements text is achieved through the Eclipse Visualization Plug-in (EVP). The EVP leverages the open-source eclipse framework, specifically the UML2 component and tools project, which represents an implementation of the UML 2.x OMG metamodel using the Eclipse Modeling Framework (EMF) and a set of UML diagram editors developed using the Graphical Modeling Framework (GMF) for viewing and editing UML models [27, 28].

The inputs to the EVP consist of one XML file containing the ROM presentation of the user requirements, another XML file embodying the ECC model, the other XML file containing the dictionary of keywords for type (d) relations, and finally the XML file that contains the textual description of the requirement text. These XML files are then processed by the UC Modeler module, which is responsible for generating the final XML file that represents the context use case diagram elements(e.g. primary actors, use cases, etc.). The processing steps of this module are described in section 4 and will not be explained again due to lack of space. The model described in this XML file is then transformed into the XMI file that is in accordance with the UML2 metamodel provided by the UML2 component using the Transformer module. The UML2 Tools use case diagram editor is then used to view and edit the use case context view. These diagrams are also saved in XMI files. Figure 4 shows the architecture of the system, and Figures 5 and 6 present sample snapshots of the tool.

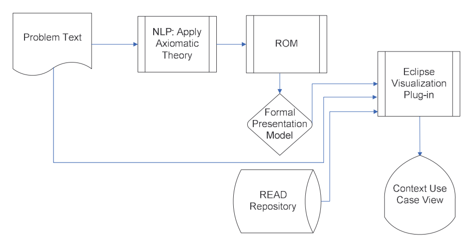


Figure 4. Tool Architecture

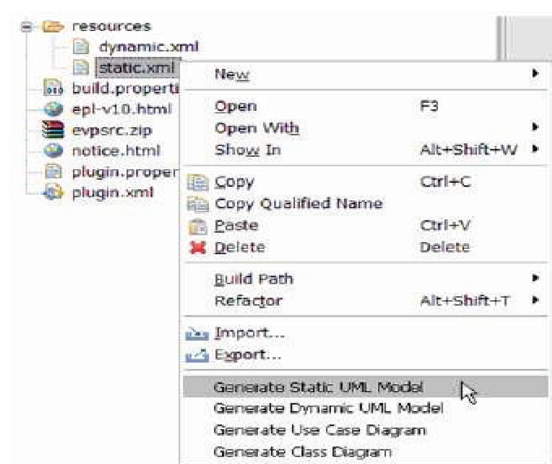


Figure 5. Snapshot of the EVP

A controlled experiment was designed to validate the methodology introduced in this paper, the details of which are explained next.

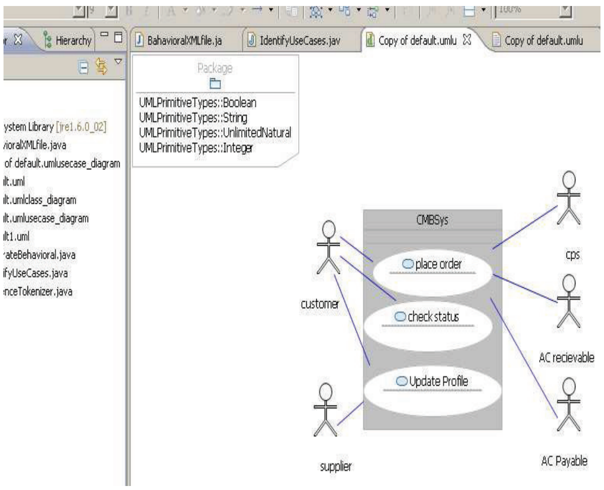


Figure 6. Snapshot of the EVP

7. Validation

In order to evaluate our methodology, we designed an experiment similar to the approach in [9]. In our experiment, the same invoicing system description (see section 5) was given to two experts who created the corresponding use case diagrams. The intersection of these models served as a benchmark for validation purposes. The case study was also given to five graduate students in software engineering with a good knowledge of use-case modeling; as a result, five use-case diagrams were developed. The students' use case diagrams were analyzed carefully and summarized based on the average number of correct and incorrect choices of actors, use cases, and their communications. Next, the students' use case diagrams and the use case diagram developed using our methodology were compared with the expert 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. 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) plays the role of the supplier (proposed by the ECC model) in this system, however they do not have the same names). Finally, the actors, the use cases, and their communications were considered different if they were either incorrect or added extra (but valid) information. The validation results are summarized in Table 5. For instance, 16.66% of the actors automatically identified by the READ tool were equal, 33.33% were equivalent, and the other 50% were 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% were incorrect.

Table 5. Validation Result

	Actor			
	Equal	Equivalent	Different	
			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

We concluded from this experiment that interaction with the user 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. READ is better at identifying the high-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 is also important to know the percentage of information that is missing from the READ result as compared to the expert models. The results of this comparison are shown in Table 6. For example, 6.66% of the use cases identified by the experts are missing from the average student' models, and 20% of the actors extracted by the experts from the text are missing from the READ tool's model. We can conclude 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 user will eliminate this deficiency. The results of the above experiment prove that such a tool for assisting the elicitation of the use cases from textual requirements is feasible.

It should be noted that only the user requirements text, without the ECC model, was given to the students and experts. The purpose of the experiment was to compare the automatically generated results of our tool with those derived by human analysts without influencing their requirements analysis process, whereas in our methodology we use the ECC as a substitution for the knowledge and experience of the human analysts.

Table 6. Validation Result

	Actor
	Missing
READ	20%
AVG_Students	8%
	Use Case
READ	0
AVG_Students	6.66%
	Communication
READ	0
AVG_Students	28.57%

8. Conclusions and Future Work

According to the statistics [16], a large number of software projects either fail or deliver products that do not provide the required functionality that customers expect. The functionality of the software system to be developed is usually captured as a Use-Case Model in the requirements analysis phase. This paper proposes a methodology for automatically assisting the software analysts in the early steps of the use case model elicitation process. It also briefly describes a tool which implements the proposed methodology, illustrates the approach on a case study, and discusses the results of the evaluation. The remaining challenge here relates to the inconsistencies that may arise in a large requirements text and that we are unable to identify automatically, as well as to the scalability of the approach, which is yet to be determined through larger, real-world case studies.

In the context of the READ project, detecting ambiguities at the level of surface understanding has already been tackled successfully in [13], and we are currently investigating means for visualizing the non-functional requirements (NFRs) that are automatically extracted and classified from text [12]. Specifically, NFRs will be highlighted to increase their visibility, and explicitly linked to the corresponding functional requirements, and they will be integrated with the domain model and CUCM. We believe that such automated assistance would be very beneficial to Requirements Engineering.

Acknowledgments

The authors would like to thank Dr. Yong Zeng (Concordia Institute for Information Systems Engineering, Concordia University) for providing information and references on the Recursive Object Model (ROM) introduced in section 3 of this paper.

References

[1]Abbott, R. J. Program Design by Informal English Descriptions, CACM, vol. 26, iss. 11, pp. 882-894 (1983)

[2]Abu Talib, M., Ormandjieva O., Abran,A., Khelifi A.,and Buglione.L. Scenario-based Black-Box Testing in COSMIC-FFP: a Case Study. ASQ Software Quality Professional Journal 8 (3), pp.23-33, June 2006.

[3]Ambriola, V .and Gervasi, V. Processing natural language requirements, In: Proceedings of Automated Software Engineering (ASE'97), 12th IEEE International Conference, November 1-5, pp. 36-45 (1997)

[4]Booch, G. Object-Oriented Development, TSE, vol.12, n.2, pp. 211-221 (1986)

[5]Chen, Z. Y. Formalization and Classification of Product Requirements Using Axiomatic Theory of Design Modeling, Master's Thesis, Department of Electrical and Computer Engineering, Concordia University (2006)

[6]Chen, Z. Y., Yao, S., Lin, J. Q., Zeng, Y., and Eberlein, A. Formalization of product requirements: From Natural language description to formal description, Int. J. of Manufacturing Research, vol. 2, no. 3, pp. 362-387 (2007)

[7]Cockburn, A. Writing Effective Use Cases, Addison-Wesley (2001)

[8]Diaz I., Losavio, F., Matteo, Al., and Pastor, O. A specification pattern for use cases, Information & Management, vol. 41, iss. 8, pp. 961-975 (2004)

[9]Díaz I., Moreno L., Fuentes, I., and Pastor O. Integrating Natural Language Techniques in OO Method. Proceedings of the 6th Int'l. Conference on Intelligent Text Processing and Computational Linguistics (CICLing'05), LNCS Springer Verlag, February 2005.

[10]Drazan, J. and Mencl, V. Improved processing of textual use cases: Deriving behavior specifications. In: Proceedings of SOFSEM 2007, LNCS, vol. 4362, pp. 856-868 (2007)

[11]Harmain, H. M. and Gaizauskas, R. CM-Builder: An Automated NL-based CASE Tool, In: Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE'2000), pp. 45-53 (2000)

[12]Hussain I., Kosseim, L., and Olga Ormandjieva. Using Linguistic Knowledge to Improve the Detection of Non-Functional Requirements Specifications. Proceedings of the 13th International Conference on Applications of Natural Language to Information Systems (Accepted at NLDB 2008).

[13]Hussain, I., Ormandjieva, O., and Kosseim, L. Automatic Quality Assessment of SRS Text by Means of a Decision Tree-Based Text Classifier. In: Proceedings of the Seventh International Conference on Quality Software (QSIC 2007), pp. 209-218, Portland, USA (2007)

[14]Kim, J., Park, S., and Sugumaran V. Improving use case driven analysis using goal and scenario authoring: A linguistics-based approach, Data and Knowledge Engineering, 58 (1 Spec. Iss.), pp. 21-46 (2006)

[15]Larman, C. Applying UML and Patterns. 3rd ed., Prentice Hall (2004)

[16]Leffingwell, D. and Widrig, D. Managing SoftwareRequirements. Addison-Wesley (2003)

[17]Moradi Seresht, Sh., and Ormandjieva, O. Towards Automatic Diagnostic of Conceptual Problems in Requirements Engineering. Accepted at the 2008 International Conference on Software Engineering Theory and Practice SETP-08 (2008)

[18]Overmyer, S. L. V., Rambow, O. Conceptual Modeling through Linguistics Analysis Using LIDA, In: Proceedings of the 23rd International Conference on Software Engineering (ICSE2001), pp. 401-410 (2001)

[19]Perez-Gonzalez, H. G., and Kalita, J. K. GOOAL: A Graphic Object-Oriented Analysis Laboratory, In: Companion of the 17th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA '02), pp. 38-39 (2002)

[20]Ryan, K. "The role of natural language in requirements engineering," Requirements Engineering, 1993., Proceedings of IEEE International Symposium on, pp. 240–242, 4-6 Jan (1993)

[21]Silverston, L. The Data Model Resource Book, vol. 1, Wiley Computer Publishing (2001)

[22]Some S., Supporting use case based requirements engineering. Journal of Information and Software Technology, Volume 48, Issue 1, January 2006, Pages 43-58

[23]Subramaniam, K., Liu, D., Far, B. H., and Eberlein, A. Automatic Transition from Use-Cases to Class Model, In: Canadian Conference on Electrical and Computer Engineering, IEEE CCECE 2003, vol. 2, pp. 831-834 (2003)

[24]Zeng, Y. Axiomatic theory of design modeling, Transaction of SDPS: Journal of Integrated Design and Process Science 6, pp.1-28 (2002)

[25]Zeng, Y. Formalization of Design Requirements, in Integrated Design and Process Technologies, IDPT-2003, Austin, Texas, December 3-6 (2003).

[26]Zeng, Y. "Recursive object model (ROM): a graphic language for representing linguistic information in design," Computers in Industry (accepted) (2008)

[27]EclipseModelingFramework,http://wiki.eclipse.org/MDT-UML2-FAQ

[28]GraphicalModelingFramework,http://wiki.eclipse.org/MDT-UML2Tools

[29]Mylopoulos, J. Structured Analysis and Design Technique (SADT), http://www.cs.toronto.edu/~jm/2507S/Notes04/SADT.pdf

[30]West, M. "Developing High Quality Data Models," Shell International Limited, http://www.matthewwest.org.uk/Documents/princ03.pdf.