

Automatic Comprehension of Textual User Requirements and their Static and Dynamic Modeling

Las Vegas Nevada, USA, June 26-29, 2006

O. Ormandjieva, M. G. Ilieva
Department of Computer Science and Software Engineering
Concordia University
Montreal, Quebec, Canada

Abstract - Requirements engineering is the most important activity in software engineering, and is concerned with the gathering and understanding of user requirements written in natural language (NL). There is a gap between the textual description of the software to be developed, and the software UML models abstracting the static or dynamic views of the software. Our research is aimed at filling this gap by automatically extracting from text a more universal (graphical) hybrid model to present the variety of knowledge that can be found in textual requirements written in unlimited NL. The advantages of a hybrid diagram include its ability to illustrate, in a way which is graphical and simple to perceive, both the static and dynamic view of the system to be developed. The hybrid diagram serves as a source for an automatic inspection of the requirements text, and will provide timely feedback to the user on the actual content of the requirements he/she has written. Such feedback will enable correction and clarification of the text before it is delivered to the software developers, thereby helping them to avoid incorrect interpretation of the requirements. The hybrid diagram also allows for automatic generation of scenario-based black-box test cases for ensuring the final quality of the software.

Keywords: Natural Language Processing, Requirements Engineering, Knowledge Mining, UML diagrams

1.0 Introduction

The importance of requirements engineering (RE) is crucial for the software engineering discipline, as it deals with the gathering and structuring of the initial knowledge on the software to be developed. The main objectives of RE are: i) precise documentation of the requirements to serve as an input to the design and implementation; ii) verification of the quality of requirements; for instance, their completeness or consistency.

Often the customer's expectations regarding the functionality of the system are unrealistic, incomplete or, at the very least, unclear. It is the responsibility of the requirements engineer to elicit from the stakeholders their expectations on the characteristics of software to be developed, and document them for further reference. Some of the current techniques for requirements gathering and analysis, such as interviews, brainstorming, etc., have proved useful, but they are informal and rely on the engineer's ability and expertise in communicating with the stakeholders.

Our research targets the assessment of the process of eliciting requirements from an uncontrolled natural language (NL) description of the software to be developed. In our experience, unrestricted NL is the most common way to present these types of documents. To achieve an objective assessment of the preciseness of requirements and to guarantee their ultimate quality, the requirements engineer has to be equipped with the right requirements analysis tool. In general, the expectations for such a requirements modeling tool are the following:

- 1) It must be easy for all the stakeholders in the RE process to use. The

stakeholders are users, developers, managers, etc., who in general use different languages to present their knowledge and their objectives. The tool must use a notation that is easily understood by all the participants in the RE process.

2) It must be multifunctional, so that it can describe different aspects of the requirements.

3) It must describe, with sufficient precision, the requirements, the processes and their changes.

The automated and interactive processing of the textual requirements, resulting in both static and dynamic views of the software to be developed, with a graphical interface for visualizing the static and dynamic models, will assess the developers' and stakeholders' ability to understand the requirements and speed up the RE process. The process is iterative, starting with an imprecise initial model and then upgrading it in an interactive manner to a UML model, which will serve as an input to the design phase. Publications on the automatic analysis of NL requirements offer different solutions in terms of their methods of analysis of NL, as well as in terms of the models to use to present the knowledge extracted. Based on our experience and research on the categories of textual requirements and their corresponding models, we grouped together the various theories in the following ways:

1) Theories which process requirements text in order to make it more understandable, precise and suitable for further automatic processing, with the result that a conceptual model of the requirements is obtained [3, 7, 13].

2) Theories which analyze restricted NL containing simple sentences describing the consequences of actions, such as the steps in a scenario specification, for which the most appropriate formal (graphic) representation is a UML [14] sequence (or similar) diagram [2, 12].

3) Theories which analyze restricted NL containing uncomplicated, lean sentences, describing the structure of the problem domain, the relationships in this structure, as well as the actions performed, for which the appropriate formal presentation is the UML class diagram [8, 10, 11, 15].

4) Theories which analyze more complicated, but lean and unambiguous text containing knowledge about actors, action, exchange of messages, business rules and their verification, for which the UML activity (or similar) diagrams are appropriate [1, 6, 9].

The following diagram schematically represents the above statements:



Our methodology processes uncontrolled NL and extracts more knowledge from the text, in terms of quantity and variation, than related approaches. The models that we offer as a result of our analysis are therefore more complex and combine both static and dynamic views of the system under development. They resemble the Use Case Maps (UCM) developed and applied in the telecommunications industry [4, 5]. While the transfer from text to maps (models) is considered to be a difficult and challenging task for the UCM, our approach naturally maps the text to the graphical presentation, and subsequently transfers one type of graphic to another, simpler and more natural one.

Our paper is organized as follows: Section 2 explains the linguistic aspects of our approach, while section 3 introduces its engineering aspects, which are illustrated on a simple case study. The conclusions and directions for future work are outlined in section 4.

2.0 Linguistics aspects of our approach

The research on automatic analysis of NL is concerned primarily with an

initial clarification of the text and with the creation of theories for subsets of syntax and semantic forms. In our approach, we explore unlimited text because that is the type of text most commonly found in the everyday practice of software engineers. The advice to write clear and precise descriptions is not always followed by users, who tend to write them in terms that intuitively remind them of their problem. For this reason, we have proceeded with the analysis of unlimited NL descriptions with the expectation that every text has its own specifics derived from the NL style of its author. The main steps in the text analysis process for extracting the knowledge required to build UML models are outlined in the stages below. For more details, see [11].

1st stage: POS tagging and table presentation of the text. At this stage, we define the grammatical categories of the words in the sentences of the text. For this, we use the MBT tagger [16]. We process the tagged text (using grammar rules and heuristics) to divide the complex sentences into their building subphrases. These consist of the members of the Su, Pr, Ob triad (Subject, Predicate, Object) that also serve as the main columns in our internal table described in [11]. This tabular presentation is somewhat informal, and a convenient and effective way to display knowledge for sequential automated processing. It is flexible as well, in that it can include different numbers of columns in which different knowledge extracted from a text can be stored depending on the characteristics of the text and the processing objectives (for example, adverbs).

2nd stage: Semantic network. We build the semantic network from the tabular presentation of the text, where the Su, Pr, Ob triad is mapped as two knots labeled Su and Ob, connected by a directed arc from Su to Ob and labeled Pr. Different graphical elements present different semantics. For instance, simple relationships - verbal (*reject, release*), prepositional (*on, to*) - are those in which there exists a close relationship between words situated close to one another. A conditional relationship connects two phrases of the *if-then* language construct type (see 5th sentence, Table 2). Compositional relationships are structural, and are presented with key words, such as *consist of, type of, set of, kind of, etc.* Attributive relationships (adjective) are attached to the knot that they explain, and are presented with a dashed knot ("*pending order*", see 7th sentence, Table 2). The graphical set of elements is open to additions and improvements. The semantic network cannot include presentation of the entire text, however. For example, in the case study shown in section 3.4, we have not presented the business rules, i.e. phrases like "*if Su is attribute/condition then action*". We can extract this knowledge from the tabular presentation. The rules are arranged in order of their appearance, which keeps the knowledge for the order of the activities (follows the natural order of the sentence).

3rd stage: use of knowledge for the semantic network for building UML models. The advantage of the semantic network is that it presents the objects of the text in a synthesized, short and efficient manner. Each object is presented through its connections (relationships) to the other elements of the text.

From the semantic network and the glossary containing key words, we extract knowledge such as: 1) actors and real-world concepts; 2) the messages between them; 3) the actions of each actor; 4) the business rules involved; 5) special control actions like "go to", "repeat", "retry"; 6) presentation of the time sequence. We summarize this knowledge in what we call a hybrid diagram between the UML sequence and activity diagrams introduced in the next section.

3.0 Engineering aspects of our approach

The hybrid diagram is extracted automatically from the textual requirements, and represents simple and clear knowledge concerning the software to be developed, as described in the text. The diagram is easy to perceive by the user who wrote the requirements, thus allowing for fast

"prototyping" of the text's content; a modification required by the graphical presentation hints at potential problems in the description of the software. Once the user agrees on the content, the diagram can be easily mapped to UML sequence diagrams, activity diagrams and class diagrams (representing the domain model), where the mapping preserves the consistency of these diagrams that represent different views on the software to be developed.

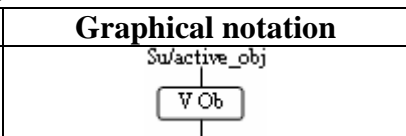
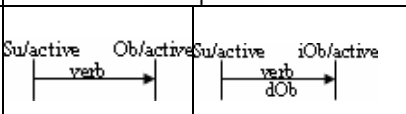
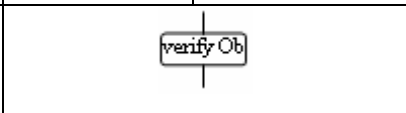
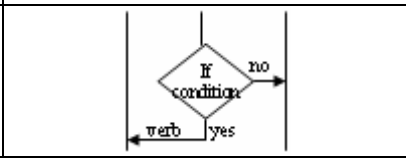
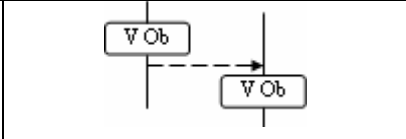
3.1 Methodology for building a hybrid diagram

The steps for analyzing the text and its mapping to a hybrid diagram are outlined below:

Step 1. We extract the participating real-world concepts (including actors) from the semantic network (see section 2) in which these concepts are presented by the nodes. We search the text for sentences of the order: Subject → Verb → direct_Object → preposition (direction) → indirect_Object. Usually, this word sequence signifies the following: the action Ve is performed by Su on dOb and directed towards iOb. We consider Su and iOb as active concepts and present them as a lane, similar to the way lanes are used in a UML activity diagram. Each active concept can initiate an action; dOb is a passive concept, and, as such, forms part of the action's description; in a hybrid diagram, the actions are presented as rectangles attached to the lane of the corresponding active concept. For example, in the phrase order department matches the article to the order, the candidates for lane presentations are order department and order (Su and iOb), while matches the article is treated as an action (see Table 1). "System" is, by default, a concept.

Step 2. We extract the actions performed by the real-world concepts identified in Step 1. There are five types of actions: internal, communication, check point, decision and implicit. These types, their meaning and their graphical notations are presented in Table 1.

Table 1: Hybrid Diagram Notation

Type of action	Sentence syntactical structure	Graphical notation
1. internal	Su/active_obj – Verb – Ob/passive_obj	
2. communication	Su/active_obj – Verb – Ob/active_obj Su/active_obj – Verb – dOb – prep – iOb/active_obj	
3. check point	Su/active_obj – key word – Ob/passive_obj key word : verify, check, test...	
4. decision	If condition then type 1. action If condition then type 2. action	
5. implicit	Passes the result of an action. Designated to maintain the sequence between the actions in different lanes without a connection.	

Step 3. In this step, we identify the order of actions in time. We have two options: i) sequence of actions, where each action can be performed only after the previous one has been completed; ii) concurrency of actions, where actions can be performed in parallel in time. We assume that, in general, the actions are sequential and take place according to the order in which they appear in the text. Adverbs like simultaneously, in parallel, at the same time, etc. are indicators of concurrency.

The advantages of a hybrid diagram include its ability to illustrate both the static and dynamic view of the system to be developed in a form which is graphical and simple to perceive. It allows the many details presented in the NL textual description to be hidden, and thus make it possible to concentrate on the correctness of the very high-level abstract description of the system. The hybrid diagram serves as an automatic inspection of the requirements text, and provides timely feedback to the user on the actual content of the requirements he/she has written. This feedback will lead to correction and clarification of the text before it is delivered to the software developers, thereby helping them avoid incorrectly interpreting the requirements.

3.2 Building Static and Dynamic Views

The UML diagrams applicable to the RE are specially designed to abstract one of the views, but never both at the same time. For instance, at the requirements specification stage, the static view is modeled in UML as a class diagram used to abstract the domain model, in which the classes are the real-world concepts (not software classes); the sequence diagrams, activity diagrams, state diagrams and use-case diagrams model different aspects of the dynamic view of the system. For instance, sequence diagrams abstract instances of interactions between the user and the system called scenarios. In contrast, the hybrid diagram is a composite of both the static and the dynamic view of the software boundary. The static view of the system is generated from the semantic network.

3.2.1 Static View: Domain Model. The knowledge from the semantic network can easily be transferred to the domain model. What is interesting in the semantic network are the internal knots in the net, which apply to concepts, and the connections within them that apply to their relationships. In our work, we adapt the various heuristics for finding the candidate concepts and their relationships, the details of which are explained in [11].

3.2.2 Dynamic View: Sequences of Interactions. Write down all the possible paths of interaction between the concepts in the hybrid model, from the "trigger" (first action) to the final action by following the sequence of the actions identified in 3.1 - Step 2. Each path represents a sequence of interactions between the user and the system; the corresponding sequence diagram is generated automatically from the hybrid model to facilitate the writing of the corresponding scenario in which the participating concepts and the actions are described textually. We are currently working on identifying and assigning business rules to each of these paths, and relating them to the corresponding scenarios.

3.3 Automatic black-box scenario-based test case generation

Each path of interaction identified in 3.2.2 can serve as a basis for the creation of a scenario-based black-box test case. Scenario-based testing is a typical black-box testing methodology at the system level, where scenarios depict the sequence of executions of the system. Testing implies a trade-off between limited resources and schedules, and inherently unlimited test requirements. As a result, we need a finite test set and enough testing to obtain reasonable assurance of quality. The generated sequences of the interactions in 3.2.2 represent all the possible scenarios outlined in the user requirements, and can be used to automatically generate black-box scenario-based test cases for the system to be applied later in the conformance testing activity.

3.4 Illustration

We illustrate our approach on a simple example extracted from [9]. Although this example falls into the "controlled natural language" category, we have

chosen it because of its simplicity, which allows us illustrate our methodology in a quick and simple way. Figure 1 presents the semantic network. Table 2 presents the original text and the hybrid diagram generated following our methodology. The domain model generated from the semantic network is depicted in Figure 2. A sample sequence of interactions is shown in Figure 3.

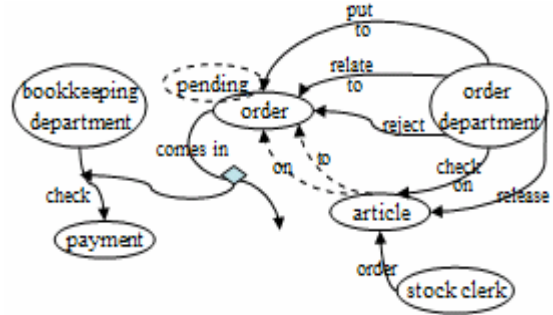


Figure 1: Semantic Network

Table 2: Illustration

Text Requirement Specification	Hybrid diagram
<ol style="list-style-type: none"> 1. The order comes in. 2. The order department checks each article in the order. 3. If the articles are available in stock, then the order department matches the articles to the order. 4. If the stock quantity of an article is less than the minimal stock quantity, then the stock clerk must reorder this article. 5. When the order comes in, the bookkeeping department checks the payment. 6. If the payment is authorized and all the articles are in stock, then the order department releases the order. 7. If payment is authorized, but not all the articles are in stock, then the order department must put the order into the pending orders file. 8. If the payment is not authorized, then the order department must reject the order. 	

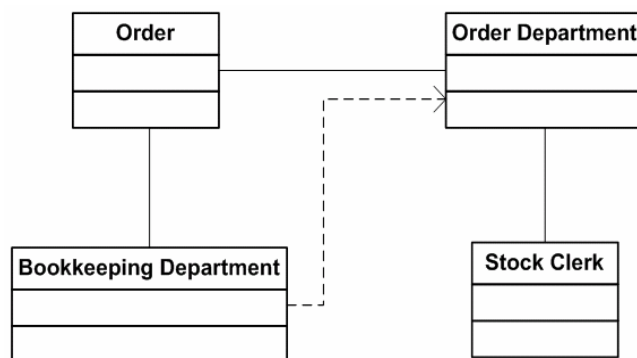


Figure 2: Static View (Domain Model)

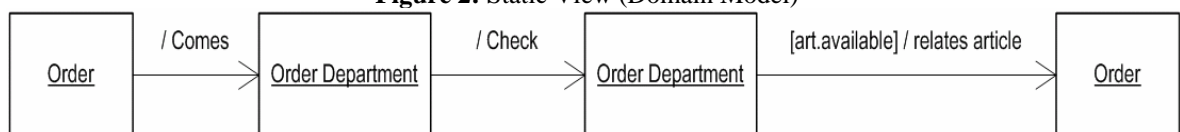


Figure 3: Sequence of Interactions

3.5 Simplified hybrid diagram

Our hybrid diagram clearly illustrates the views of the system, however it takes up a great deal of space, which may result in some confusion. If we have many diagrams of this type which must be processed and stored, we need

a shorter description. A simplified hybrid diagram would contain a brief description of the scenario with the actions distributed sequentially or in parallel over time, from the first to the final action. Time is plotted on the vertical axis and starts with the first action, as shown in Figure 4. This simplified diagram clearly illustrates the possible interaction paths, which are exactly the same as the paths extracted from the hybrid diagram (see, for instance, Figure 3); thus, the two diagrams are congruent in terms of the knowledge presented, and either can be used.

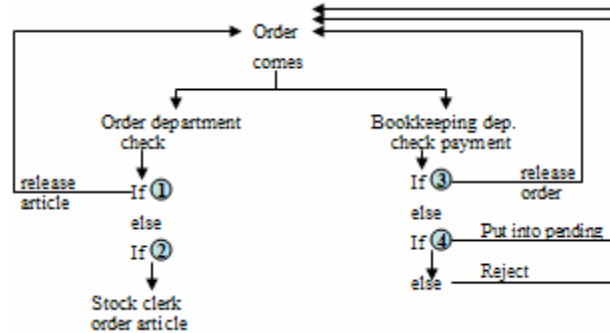


Figure 4: Simplified Hybrid Diagram

4.0 Conclusions and Directions for Future Work

Requirements engineering is the most important activity of software engineering, and is concerned with the gathering and understanding of the user requirements written in NL. Our research targets the extraction and abstraction of knowledge on the characteristics of software to be developed from the user requirement written in NL. An experienced requirements engineer extracts fragmented pieces of knowledge from the text and analyzes them following patterns formed from his/her experience. However, the inexperienced requirements engineer looks for a match between the knowledge in the text and the respective models for their presentation, and is often discouraged by the lack of such models. In exchange, he/she receives diagrams that present just part of the knowledge in the text. Cognizant of this gap between knowledge in the text and the models that present it, we try to fill it by creating more universal (graphical) hybrid model for presenting the more varied knowledge that can be found in textual requirements written in unlimited NL.

We propose an automated building of a hybrid (both static and dynamic) model from text to graphically represent the knowledge mined from that text. In the hybrid diagram, we abstract real-world concepts, actions, decisions and interchanges between the concept messages. The lane for every concept is visible, as are all the activities that it performs. The activities are distributed on a time line. The simplified hybrid diagram maps the way to complete a functionality distributed in time and space (between concepts). We have shown the way in which one hybrid graphical presentation is transformed into a UML class model (static view), and how the dynamic view can be mapped to a scenario and its corresponding sequence diagrams.

Our future work includes an analysis of different case studies and the development of a tool to interactively verify requirements through their animation at the requirements specification stage.

5.0 References

- [1] A. Frankl and T. King. "How to Detect Requirements Errors - a Guide to Slashing Software Cost and Shortening Development Time". In <http://www.n8systems.com/whitepaper/paper.php>, December 2005.
- [2] C. Rolland, Ben Achour. "Guiding the construction of textual use case

specification". In Proceedings of Data & Knowledge Engineering 25: 125--260, 1998.

[3] C. Rolland, N. Prakash. "From Conceptual Modeling to Requirements Engineering". Annals of Software Engineering, Special Volume on Comparative Studies of Engineering Approaches for Software Engineering, 1999.

[4] D. Amyot, X. He, Y. He, D. Y. Cho. "Generating Scenarios from Use Case Map Specifications". In Proceedings of the Third International Conference On Quality Software (QSIC'03): 108--115, 2003.

[5] D. Amyot, L. Logrippo, R. J. A. Buhr, T. Gray. "Use Case Maps for Capture and Validation of Distributed Systems Requirements". In Proceedings of the Fourth IEEE International Symposium on Requirements Engineering RE'99: 44--53, June 1999.

[6] G. Fliedl, C. Kop, H. C. Mayr. "From Scenarios to KCPM Dynamic Schemas: Aspects of Automatic Mapping". In Proceedings of the 8th International Conference on Applications of Natural Language to Information Systems (NLDB'03): 91--205, 2003.

[7] J. F. M. Burg and R. P. van de Riet. "The Impact of Linguistics on Conceptual Models: Consistency and Understandability". In Proceedings of the 1st International Workshop on Applications of Natural Language to Databases (NLDB'95): 131--146, 1995.

[8] K. Subramaniam, D. Liu, B. H. Far and A. Eberlein. "UCDA: Use case driven Development Assistant Tool for Class Model Generation". In Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2004): 324--329, 2004.

[9] G. Fliedl, Ch. Kop and H.C. Mayr. "From textual scenarios to a conceptual schema". Data & Knowledge Engineering 55(1): 20--37, October 2005.

[10] B.S. Lee, B. R. Bryant. "Automated conversion from requirements documentation to an object-oriented formal specification language". In Proceedings of the 2002 ACM Symposium on Applied Computing (SAC): 932--936, March 2002.

[11] M. Ilieva, O. Ormandjieva. "Automatic Transition of Natural Language Software Requirements Specification into Formal Presentation". Lecture Notes in Computer Science LNCS 3513: 392--397, 2005.

[12] V. Mencl. "Deriving Behavior Specifications from Textual Use Cases". In Proceedings of the Workshop on Intelligent Technologies for Software Engineering (WITSE04): 331--341, 2004.

[13] N. Boyd. "Using Natural Language in Software development". Journal of Object-Oriented Programming 11(9): 45--55, 1999.

[14] Unified Modeling Language (UML) 2.0 -<http://www.uml.org/>

[15] V. S. Alagar, M. Chen, O. Ormandjieva, M. Zheng. "Automated Test Generation from Object-Oriented Specifications of Real-Time Reactive Systems". In Proceedings of the 10th Asia-Pacific Software Engineering Conference (APSEC2003): 406--414, 2003.

[16] W. Daelemans, J. Zavrel, P. Berck and S. Gillis. "MBT: A Memory-Based Part of Speech Tagger-Generator". In Proceedings of the 4th Workshop on Very Large Corpora, Copenhagen, Denmark: 14--27, 1996.