A Relation-based Approach to Use Case Analysis

A.Fantechi*, S.Gnesi^, G.Lami^

*Dip. di Sistemi e Informatica - Università di Firenze - Italy

^ISTI - C.N.R. - Area della Ricerca C.N.R. di Pisa - Italy

Abstract

Use Cases are an effective tool for modeling functional requirements of software systems. A well written Use Case allows to depict a large amount of information regarding the behaviour of the system, as perceived by the actors. Use Cases have the advantage to be expressed using Natural Language expressions that have a fixed structure and this can mitigate some of the usual, NL-inherent, problems of interpretation. In this paper, we present an approach that, starting with the application of NL processing techniques to the Use Case scenarios, derives semantic information on the relations between the actors. This information, largely achievable in an automatic way, can be used to support the analysis of Use Case requirements document and it represents a starting point towards the formal verification of some relevant aspects.

1. Introduction

The problem of the analysis of software requirements with respect to some consistency and correctness parameters has been extensively exploited in several ways in recent years [14]. For example, formal methods and tools have been used for this purpose when a formal representation of software requirements has been adopted.

Currently, in the common practice formal notations are not always used in the first description of the system. More frequently Natural Language (NL) expressions are used to represent software requirements [12, 15]. It is hence quite important to provide methods and tools for the consistency and correctness analysis of them starting from their NL representation.

Use Cases are a powerful tool to capture functional requirements for software systems. They allow structuring requirements documents with user goals and provide a means to specify the interactions between a certain software system and its environment. In his book [3], Alistair Cockburn presents an effective technique for specifying the interactions between a software system and its environment. The technique is based on natural language specification for scenarios and extensions. Scenarios and extensions are specified by phrases in plain English language. This makes requirements documents easy to understand and communicate even to non-technical people.

The typical structure of Use Cases makes their analysis easier and more effective than classic Natural Language sentences. Quality is determined by the fulfilment of some predefined characteristics (target qualities) that in the case of Use Cases can be classified into three main groups: Expressiveness, Completeness and Consistency.

- *Expressiveness* category: it includes those characteristics dealing with the understanding of the meaning of Use Cases by humans, such as *Ambiguity* or *Understandability*
- *Consistency* category: it includes those characteristics dealing with the presence of semantics contradictions and structural incongruities in the NL requirement document.
- *Completeness* category: it includes those characteristics dealing with the lack of necessary parts within the requirement specifications.

The quality of NL components of Use Cases (typically sentences), may be analysed from a lexical, syntactical or semantical point of view [8]. For this reason it is proper to talk about, for example, lexical non-ambiguity or semantical non-ambiguity rather than non-ambiguity in general [10,11,13]. For instance, a NL sentence may be syntactically non-ambiguous (in the sense that only one derivation tree exists according to the syntactic rules applicable) but it may be lexically ambiguous because it contains wordings that have not a unique meaning.

In the practice expressiveness-related issues can be addressed by means of existing NL-based techniques and tools [2,6,9,17]. For example, ambiguity mitigation may be addressed in the following ways:

- By lexical evaluation: using lexical parsers to detect and possibly correct terms or wordings that are ambiguous.
- By syntactical evaluation: using syntactical analysers to detect sentences having different interpretations.

Understandability improvement may be addressed by lexical and syntactical evaluation as well: using linguistic parsers both to detect poorly understandable or complex parts of the document and to achieve readability indicators (metrics) based on the count of elements of the sentences (e.g. the number of characters or words of the sentences, the average length of the sentences, …).

It is more difficult to exploit NL-based techniques and tools able to provide some help in addressing Consistency and Completeness issues, because it is necessary to capture, at least at some level, the semantics of the Use Case under evaluation.

In this paper we present a methodology for the analysis of the Use Case based requirements documents. This methodology is aimed to extract semantic information from the Use Cases, based on NL processing techniques. This information regards the functional relations between the actors of the Use Case based requirements specification. From the elementary relation between two actors determined by the verb in a sentence we discuss how to derive more complex relations between the concepts present in the Use Cases description, which may help in assessing consistency and completeness issues. This derivation can be largely supported by automatic tools.

This paper is structured as follows: in section 2 we describe the kind of Use Cases we will consider and in section 3 we present our approach to support the analysis of consistency and completeness based on linguistic techniques. In section 4 we discuss how to derive the relations, presenting an integrated environment for the lexical, syntactical and semantic analysis of NL requirements that can be used to this aim. In section 6 we discuss the possible use of the relational approach and the related opportunities to improve the analysis of Use Case-based requirements documents.

2. Use Cases

A Use Case describes the interaction (triggered by an external actor in order to achieve a goal) between a system and its environment. A Use Case defines a goal-oriented set of interactions between external actors and the system under consideration. The term *actor* is used to describe the person or system that has a goal against the system under discussion. A primary actor triggers the system behaviour in order to achieve a certain goal. A secondary actor interacts with the system but does not trigger the Use Case.
A Use Case is completed successfully when that goal is satisfied. Use Case descriptions also include possible extensions to this sequence, e.g., alternative sequences that may also satisfy the goal, as well as sequences that may lead to failure in completing the service in case of exceptional behaviour, error handling, etc.. The system is treated as a "black box", thus, Use Cases capture *who* (actor) does *what* (interaction) with the system, for what *purpose* (goal), without dealing with system internals. A complete set of Use Cases specifies all the different ways to use the system, and therefore defines the whole required behaviour of the system. Generally, Use Case steps are written in an easy-to-understand, structured narrative using the vocabulary of the domain. The language used for the description is English. Any other natural language can be used as well, and although our analysis focuses on English, the same reasoning can be applied to other languages (considering the obvious differences in syntax and grammar rules). A scenario is an instance of a Use Case, and represents a single path through the Use Case. Thus, there exists a scenario for the main flow through the Use Case, and other scenarios for each possible variation of flow through the Use Case (e.g., triggered by options, error conditions, security breaches, etc.). Scenarios may also be depicted in a graphical form using UML sequence diagrams. Figure 1 shows the template of the Cockburn's Use Case taken from [4].
In this textual notation, the main flow is expressed, in the "Description" section, by an indexed sequence of NL sentences, describing a sequence of actions of the system. Variations are expressed (in the "Extensions" section) as alternatives to the main flow, linked by their index to the point of the main flow in which they branch as a variation.
Developers have always used scenarios in order to understand what the requirements of a system are and how a system should behave with respect to its environment. The work we present in this paper is an attempt to provide means to identify possible flaws in the textual scenario descriptions.

USE CASE #	< the name is the goal as a short active verb phrase>	
Goal in Context	<a longer statement of the goal in context if needed>	
Scope & Level	<what system is being considered black box under design> <one of: Summary, Primary Task, Subfunction>	
Preconditions	<what we expect is already the state of the world>	
Success End Condition	<the state of the world upon successful completion>	
Failed End Condition	<the state of the world if goal abandoned>	
Primary, Secondary Actors	<a role name or description for the primary actor>. <other systems relied upon to accomplish use case>	
Trigger	<the action upon the system that starts the use case>	
Description	**Step**	**Action**
	1	<put here the steps of the scenario from trigger to goal delivery, and any cleanup afte>
	2	<...>
	3	
Extensions	**Step**	**Branching Action**
	1a	<condition causing branching> : <action or name of sub.use case>
Sub-Variations		**Branching Action**
	1	<list of variation s>

Figure 1. Use Case template

3. Towards Linguistic Evaluation of Consistency and Completeness

To effectively address the Consistency and Completeness aspects of requirements specification, we should resort to their formalization, [9,18]. Indeed formal methods are a powerful mean to evaluate requirements because they provide a theoretical framework to verify their correctness. Formal methods require, however, a specific skill and this has increased their cost and prevented their wide application.

In this paper we investigate methods and tools that may provide an effective support to deal with Consistency and Completeness issues, but that are still based on NL analysis and then are more user-friendly.

Other works aiming at the improvement of the correctness of requirements relying on the Use Cases structure exist [1,5]. The methods and tools we are presenting in this paper also rely on the structure of the Use Cases and are based on the study of the relations between actors of Use Case-based description of a system.

The relations we are interested in are the "functional" relations, i.e. the relations or dependencies between two actors. These relations can be determined looking at the syntactical structure of each sentence of the Use Case scenarios defining a set of items (quadruples) where each primary actor (the subject of the sentence) has been put in relation with the secondary actor (the complement) according to the verb. The canonical form of these relations is:

(1.) *(Actor_1, verb_i, Actor_2, Use_Case_id).*

Each item compliant with (1.) describes an occurrence of a functional relation between two actors established by the verb and indicates the Use Case in which this relation occurs.

The functional relations between two actors, in the form (1.), can be extended, by transitivity, to other actors when two items with the following form exist: (A_i, v_1, A_j, UC_x) and (A_j, v_2, A_k, UC_y). In this way, hence, an indirect functional relation between the actor A_i and the actor A_k is also established by transitivity. Starting from this consideration, chains joining different actors can be built, where each item (A_i, v_x, A_j ,UC_x) of the chain is such that the previous item has the form (A_k, v_y, A_i, UC_y) and the following has the form (A_j, v_z, A_h, UC_z). The collection of all the items derivable from a Use Case-based requirements document is said *Relations core*. The Relations core embeds all the elementary functional relations between actors that can be extracted directly by the NL description.

We can derive specific, non-elementary, relations from the relations core. In the following we provide some definitions and define some properties based on the elementary relations (1.).

The *drives* relation, that we denote by A ➨ B, holds if at least one relation *(A, verb_i, B, Use_Case_id)* exists.

The *ignores* relation, denoted by A Ⓘ B, holds if no relation *(A, verb_i, B, Use_Case_id)* exists.

The drives relation between actors can be used to build the *Relation Graph*. The nodes of this graph represent the actors and an oriented arc connecting two nodes (A and B) indicates that A drives B.

Two nodes are adjacent if an arc from A to B exists.

A *path* from the node A to the node B on this graph is a sequence of adjacent nodes in a graph starting from the node A and arriving to the node B. On the basis of the Relations graph some further relations between actors can be defined:

The *is connected to* relation, denoted by A ⏵ B , holds if at least one path from A to B exists on the graph.

The *is chief* relation, denoted by A ⏩ B, holds if B is not connected to A and A is connected to B.

The *chief graph* (derived from the chief relation) is an acyclic graph composed of nodes (the actors) and oriented arcs connecting two actors, an arc originating from the node A and arriving in the node B means that A is chief of B.

Nodes of the chief graph having no incoming arcs are said *leader* nodes and nodes having no out arc are said e*xecutors* nodes.

An example of Relation and Chief graphs are provided in Figure 2.

The availability of the functional relations and of the graphs derived from them enable the capturing of some semantic information on the system we are describing. In particular, this information can be used to support the detection of critical points (in terms of consistency and completeness) in the interactions between different actors. These critical points can be revealed by analysing the set of derived direct and indirect relations.

4. Derivation of the relations between actors

The derivation of the relations core and the consequent construction of the relation chains, relations graph and chief graph, can be supported by automatic tools based on NL processing techniques.

In fact, the basic relations (1.) are detectable by using a syntactical parser able to identify the different components of a NL sentence. To our purpose, the key components to be identified are the subject(s), the verb and the complement(s) associated to the verb. Once this information is achieved, it is possible to define the relations and to build a data base containing the relations core derivable from the collection of Use Cases under analysis.

4.1 An example of Derivation of Relations

In this section we present an application of the relational approach to a sample Use Case document, with the aim to clarify the concepts discussed above.

The example we present in this section is derived, with few changes, from a sample System Requirements Document available on the web at the Cockburn's home page, which is reported in Appendix 1
[http://members.aol.com/acockburn/papers/prts_req.htm].
This document, describing a Purchase Request Tracking System, has the purpose to provide the functional requirements of a basic system for the official Buyers of the company, to track what they have ordered from Vendors against what they have been delivered. The documents is organised as a set of Use Cases.
The primary actors of this document are:

- Approver: typically the requestor's manager, who must approve the request.
- Authorizer: person who validates the signature from the Vendor.
- Buyer: person who manages the order, talking with the Vendor.
- Vendor: person or company who sells and delivers goods.
- Requestor: person putting in a request to buy something.
- Receiver: takes care of the arriving deliveries

The document contains fifteen Use Cases describing the behaviour of the system. We slightly modified it by adding two new Use Cases to make it more precise and suitable for the analysis. The Use Cases included into the document are compliant with the Cockburn's style, and they include several data as, for example, Preconditions, Post-conditions, Trigger, Extensions, etc. We simplified each Use Case by reducing the information associated to them. In particular, we took into account only the Primary Actor, the statement of the Goal and the description of the Scenario. These data represent the minimum set of information necessary to save the essential meaning of the Use Case. In Appendix 1, the set of the simplified Use Case we used for the experiment is shown.
The outcomes of the application of the relational approach to the simplified Use Cases of the case study are summarized in a collection of relations items between actors and a set of relations chains derived from the relations items.
For simplicity let us identify the actors of the case study by a letter:

A. Authorizer
B. Approver
C. Requestor
D. Buyer
E. Vendor
F. Receiver

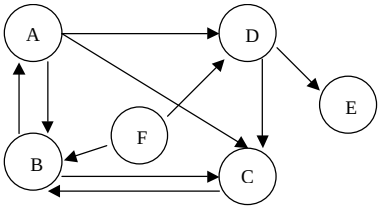
Figure 6. contains the relations derived from the case study where each actor is identified by the corresponding letter along with the corresponding relation graph and chief graph.
In the following a possible set of relations chains starting from the relation *(A, notify, B, UC3)*, is provided:

- `(A, notify, B, UC3), (B, send, C, UC6), (C, send, B, UC7).`
- `(A, notify, B, UC3), (B, send, C, UC6), (C, send, B, UC8).`
- `(A, notify, B, UC3), (B, send, C, UC9), (C, send, B, UC7).`
- `(A, notify, B, UC3), (B, send, C, UC9), (C, send, B, UC8).`
- `(A, notify, B, UC3), (B, send, A, UC12).`
- `(A, notify, B, UC3), (B, send, A, UC12), (A, change, B, UC3).`
- `(A, notify, B, UC3), (B, send, A, UC12), (A, send, C, UC9).`
- `(A, notify, B, UC3), (B, send, A, UC12), (A, send, D, UC16).`
- `(A, notify, B, UC3), (B, send, A, UC12), (A, send, C, UC9), (C, send, B, UC7).`
- `(A, notify, B, UC3), (B, send, A, UC12), (A, send, C, UC9), (C, send, B, UC8).`
- `(A, notify, B, UC3), (B, send, A, UC12), (A, send, D, UC16), (D, change, E, UC4).`
- `(A, notify, B, UC3), (B, send, A, UC12), (A, send, D, UC16), (D, send, C, UC9).`
- `(A, notify, B, UC3), (B, send, A, UC12), (A, send, D, UC16), (D, send, C, UC9), (C, send, B, UC7)`
- `(A, notify, B, UC3), (B, send, A, UC12), (A, send, D, UC16), (D, send, C, UC9), (C, send, B, UC8).`

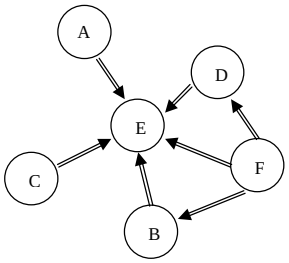
4.2 An integrated environment for Use Case Analysis

In this section we provide a description of MAIGRET, an integrated environment for natural language analysis. MAIGRET was built with the goal to provide an automatic support for the analysis framework of expressiveness, consistency and completeness aspects of natural language requirements documents. To reach this goal we have realized and integrated a set of tools, each one dedicated to a specific linguistic analysis purpose of NL sentences. In particular (figure 3), the involved tools are a lexical analyzer (QuARS) [6,7], a syntactical analyzer (SyTwo/Cmap) [16,19] able to extract items representing relations between the entities (actors) described in the requirements document and a third tool (Relation Manager) having the aim of storing and managing these relations. Figure 4 shows the high level architecture of MAIGRET.

Primay Actor	verb	Secondary Actor	UC
A	change	B	3
A	notify	B	3
A	notify	B	3
D	change	E	4
C	send	B	5
B	send	C	6
C	send	B	7
C	send	B	8
B	send	C	9
A	send	C	9
D	send	C	9
D	return	E	11
B	send	A	12
D	send	E	14
F	notify	D	15
A	send	D	16
F	send	B	17



Relation Graph



Chief Graph

Figure 2: Relations between actors and graphs

MAIGRET is composed of a processing part (the integrated tools) and of a static part (composed of the following modules: the WordNet English dictionary; a semantic network modelling the domain; and a requirements-specific dictionary) that represents the knowledge base for MAIGRET.

In the following we provide a more detailed functional description of the integrated tools.

- **QuARS** (Quality Analyzer for Requirements Specifications) is a sentence analyser aiming at reducing linguistic defects by pointing out those wordings that make the document ambiguous or not clear from a lexical point of view. The tool points out these defects without forcing any corrective actions, leaving the user free to decide whether modifying the document or not. Moreover the sentences are analyzed taking into account the particular application domain, and this is possible through the use of targeted dictionaries. In this sense the tool has been designed to be easily adaptable. The following are examples of expressiveness defects pointed out by QuARS; the underlined wordings are the indicators used by QuARS to point out the sentence as defective:
  - `the C code shall be clearly commented (vague sentence)`
  - `the system shall be as far as possible composed of efficient software components (subjective sentence)`
  - `the system shall be such that the mission can be pursued, possibly without performance degradation (optional sentence)`

  The first sentence contains the word "clearly" that makes the whole sentence vague. The second contains the wording "as far as possible" that makes it subjective. The third sentence is pointed out as defective because contains the word "possibly" that determines an option in it.

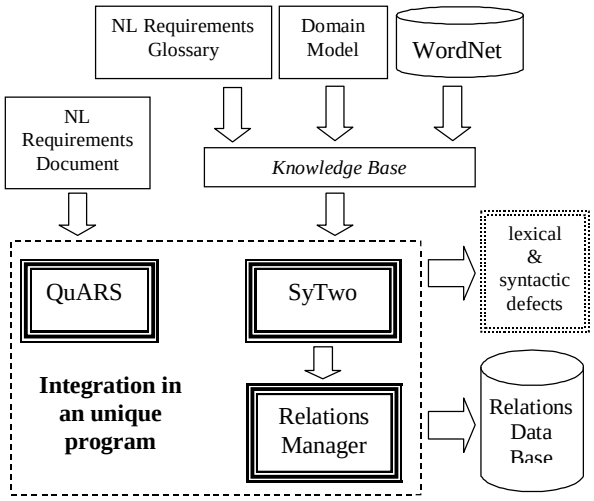


Figure 3: high level architecture of MAIGRET

SyTwo/Cmap is a syntactical analyser. Starting with a Use Case description (possibly already analyzed by QuARS), we can use SyTwo/Cmap to perform a syntactical analysis making the detection of syntactically ambiguous sentences possible. A syntactically ambiguous sentence can be pointed out if it has more than one derivation tree (i.e. the sequences of syntactical rules to be applied to build the sentence): this implies that the sentence may be understood in different ways. For example the sentence "*The system shall not remove faults and restore service*" may be syntactically understood at least in these two ways:

1. The negation *not* of the auxiliary verb *shall* is related to the first verb *remove* only, and not to the other verb *restore*. In this case, the meaning of the

sentence is that the system shall not remove the faults and it shall restore the service.

2. The negation *not* of the auxiliary verb *shall* is related to both the verbs *remove* and *restore*. In this case, the meaning of the sentence is that the system shall not remove the fault and shall not restore the service.

The functionalities described above for both tools allow therefore expressiveness characteristics to be analyzed. More interesting for the methodology presented in this paper is the additional functionality offfered by SyTwo/Cmap, which is able, on the basis of the knowledge of the syntactical structure, also to extract the relations between subjects, verbs and objects in a sentence, in a format compliant with (1.). This allows to capture all the relations between two actors in the problem domain.

- **Relations Manager**: it receives as input the set of relations derived by SyTwo/Cmap and it puts them in a database for analysis. Once such a database has been populated it is possible to:
  - extract, by means of queries, sub-sets of relations.
  - extract, by means of queries, sets of relation chains.

  The outcomes derived from the data-base, can be used in support of the analysis of the requirements document in order to extract the interactions between actors and build the relation and the chief graphs.

5. Applying the Relational Approach

In this section we discuss possible applications and developments of the relational approach to the Use Case-based requirements engineering.

Since relations indicate the presence of a verb in the use case relating two actors, they often indicate possible interactions between actors. Hence, relations chains can be interpreted as interaction schemata. *Walkthroughs* of these interaction schemata may be performed in search of undesired, inconsistent and incomplete dynamic behaviour of the system.

These schemata may also form the basis for a formal analysis of interactions, which however we do not address in this context.

Walkthroughs of interaction schemata may be aimed at detecting relation chains containing loops, because loops indicate a more complex kind of interaction, and may point to a possible synchronization problem (such as a deadlock). It is possible, in this way, to point out some potential synchronization problems in a sequence of actions.

Let us consider, for instance, the example of section 5.1. In this case we detect the relation chain: *(A, notify, B, UC3), (B, send, C, UC6), (C, send, B, UC7),* presenting a loop. If carefully walk through this chain, and we represent this interactions sequence on a time scale (see figure 6), it is possible to understand that some potential synchronisation problems may occur.

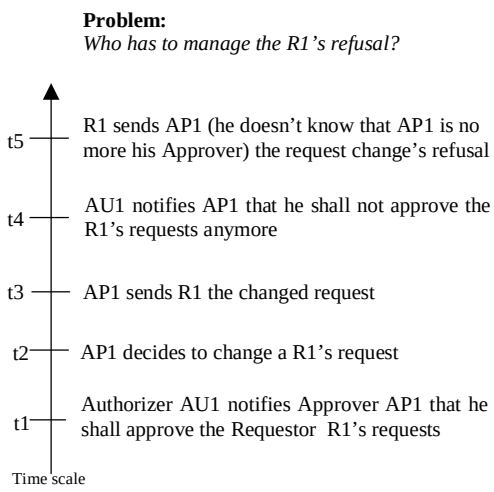


Figure 6: Interactions sequence

In fact, if the Approver AP1 sends the changed request to the Requestor R1 and, before that R1 tells A1 that this change is refused, Authorizer AU1 change the authorization to AP1 and makes A2 the Approver of R1, then who should manage R1's refusal?

In this case, it is possible to detect an inconsistency in the requirements due to an incomplete specification of the requirements because the notification of the changed Approver is not sent also to the associated Requestors. This kind of problems, that are hidden if we consider only the Use Cases-based requirements document, may be easily detected by using the relations chains.

Another possibility of exploiting this information is to point out those pairs of actors that have an higher number of interactions than the others. The pairs that, in the case study, have the highest number of different interactions are Approver-Requestor (5 interactions), Authorizer-Approver (4 interactions) and Buyer-Requestor (3 interactions). The indication that can be derived from these data is that the interactions between these actors are at the core of the functionality of the system, and therefore should be analysed in more detail in order to point at possible problems. Also, this information can give an indication to which parts of the system should be stressed at testing time.

The semantic information derivable from the derived relations and graphs can help the analysis of correctness and completeness of the requirements because some gaps in the specification of use cases can be easily found.

In fact, the graphs defined above (and in particular the chief graph) allow some interesting considerations to be made. The chief relation it is not to be intended as determining a hierarchy in terms of the importance of the role played by the actors. This relation and the information derivable from the graph is a semantic information that allows to enlighten the influence of an actor on the others. In particular, if a node A of the chief graph is connected with the node B by an arc (A, B), then it can be argued that the behaviour of B doesn't influence that of A. This kind of semantic information about the actors, that cannot be directly derived from the set of use cases, can play a relevant role for the analysis. In particular, it is possible to easily detect lacks in the relational structure of the requirements.
In the example shown in section 5.1, the relation F ⏩ D occurs. This occurrence enlighten a gap in the specifications because the buyer should have the capability of having a relation with the receiver (for instance, to ask the status of an on-going acquisition).

The relational approach can be oriented to achieve a *guidance for systematic construction of the Use Case requirements documents*. In fact, building the relations graphs in parallel with the definition of the use cases impels a continuing series of walkthroughs to check the part of the relations graph completed so far and examine how remaining relations should be added to the graphs themselves.

We wish in the end point at another application of the relational approach, which spans outside the context of Use Cases. A concept that has gained importance in the last years, especially in the telecommunication field, is the concept of feature. A feature is a capability of a system which provides value to the users, but is conceived as separate from the other features provided by a system to its users. However, at the system level, features can interact in a complex manner (a problem often referred as "Feature interaction"), so they cannot be treated as separate in the development of the system, and especially in the requirements document. A feature may even prevents other system activities: for instance, in a mobile handset user interface the "keyguard" feature prevents almost all other user-originated activities (but not incoming call handling).
The description of a feature by Use Cases can be trivial (in the keyguard example the scenario might be composed simply of the "set the keyguard on" activity) and the Use Cases may be not able to represent how the system behaviour is affected by a feature.
The knowledge of the influence of the features on the UCs can be important mainly for the testing of the system because the Use Cases are not enough for representing the consequences of the features on the functionalities they describe.
For this reason the relational approach to the Use Case analysis can be of interest to identify those Use Cases affected by a feature.
For example in order to identify the Use Cases affected by the keyguard feature, those UC can be detected having in their scenario a sentence like "User digit a key". These UC are influenced in case of the keyguard is set on.

6. Conclusions and Future works

In this paper we presented an approach to the analysis of Use Case requirements documents based on the relations between the actors. Starting from the simple relation between two actors derivable from a scenario sentence, by means of NL parsing tools, some more complex, derived relations have been defined. These relations are able to provide semantic information on the content of a requirements document, supporting the completeness and consistency analysis. The semantic information on the Use Case requirements documents that can be captured with this approach is only partial, w.r.t. the semantic of the whole requirements. Anyway, this information is able to provide a concrete support for the analysis.The use of the semantic information derivable with the relation-based approach has been discussed in this paper. In particular, the knowledge of the functional relations between actors expressed by the Use Cases allows to perform walkthroughs in the relation core to detect possible gaps in terms of consistency and completeness. Moreover, a guidance for a systematic construction of the Use Cases requirements document can be obtained by the parallel development of graphs and schemata representing the relations.
A related work to ours is that reported in [20], in which more sophisticated NL techniques are used to extract concept lattices out of Use Cases, which offer a richer information to the analysis. Our approach use simpler, low cost NL techniques to extract useful information: it would be interesting to see whether the benefits obtained by heavier NL techniques balance their higher costs.
The relation-based approach to the analysis of Use Cases is a promising research direction because it can be used as a mean to bridge the gap between the use of the informal NL descriptions typical of requirements documents, and the more formal artefacts typical of later stages of the development process. In particular, the study of the relations between actors, though starting from a light formalism as the Use Cases are, can provide enough information to move towards the application of formal methods with the support of automatic tools and in a user friendly way. We plan to investigate at this regard the annotation of the relation graph with pre-conditions and post-condition in order to perform simulations of the system and perform a more refined analysis.

Another subject that we are investigating is the extraction of test cases from Use Case scenarios. Also in this case, extracting information from the textual descriptions in the form of relations between actors helps in the definition of test cases covering the most intricate interaction schemes.

Acknowledgements

Part of the research work described in this paper was performed under the Eureka Σ2023 Programme, ITEA (ip00004, CAFÉ). We wish to acknowledge the work of Carlo Becheri on the Relations Manager component of the MAIGRET architecture.

7. References

[1] T. A. Alspaugh, A. I. Antòn, Scenario Networks: A Case Study of the Enhanced Messaging System, REFSQ'01, Interlaken, Switzerland, June 2001.

[2] V. Ambriola, V. Gervasi, Processing Natural Language Requirements, 12th IEEE Conf. On Automated Software Engineering (ASE'97), IEEE Computer Society Press, Nov. 1997.

[3] A. Cockburn, Writing Effective Use Cases, Addison-Wesley, 2000

[4] A. Cockburn, Structuring Use Cases with Goals, Journal of Object-Oriented Programming, Sep-Oct 1997 (part I) and Nov-Dec 1997 (part II)

[5] A. H. Dutoit, B. Peach, Developing Guidance and Tool Support for Rationale-based Use Case Specification, REFSQ'01, Interlaken, Switzerland, June 2001.

[6] F.Fabbrini, M.Fusani, S.Gnesi, G.Lami "The Linguistic Approach to the Natural Language Requirements Quality: Benefits of the use of an Automatic Tool", 26th Annual IEEE Computer Society - NASA Goddard Space Flight Center Software Engineering Workshop, Greenbelt, MA, USA, November 27-29 2001.

[7] F.Fabbrini, M.Fusani, S.Gnesi, G.Lami "Quality Evaluation of Software Requirement Specifications", Proc. of Software & Internet Quality Week 2000 Conference, San Francisco, CA May 31-June 2 2000, Session 8A2, pp.1-18.

[8] A.Fantechi, S.Gnesi, G.Lami, A.Maccari "Linguistic Techniques for Use Cases Analysis" Proceedings of the IEEE Joint International Requirements Engineering Conference - RE02. Essen, Germany, September 9 -13 2002.

[9] A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, P. Moreschini, "Assisting requirement formalization by means of natural language translation", Formal Methods in System Design, vol 4, n.3, pp. 243-263, Kluwer Academic Publishers, 1994.

[10] N.E.Fuchs, R.Schwitter, Specifying Logic Programs in Controlled Natural Language, Workshop on Computational Logic for Natural Language Processing, Edinburgh, April 3-5, 1995.

[11] E. Kamsties, B. Peach, Taming Ambiguity in Natural Language Requirements, ICSSEA 2000, Paris, December 2000.

[12] B.Macias, S.G. Pulman. Natural Language Processing for Requirement Specifications. In Redmill and Anderson, Safety Critical Systems, pages 57-89. Chapman and Hall, 1993.

[13] L. Mich, R. Garigliano, Ambiguity Measures in Requirement Engineering. Int. Conf. On Software Theory and Practice - ICS 2000, Beijing, China, Aug. 2000.

[14] B. A. Nuseibeh and S. M. Easterbrook, Requirements Engineering: A Roadmap, In A. C. W. Finkelstein (ed) "The Future of Software Engineering". (Companion volume to the proceedings of the 22nd International Conference on Software Engineering, ICSE'00). IEEE Computer Society Press.

[15]C. Rolland, C. Proix, A Natural Language Approach for Requirements Engineering. AISE'92, LNCS 593, Springer-Verlag, 1992.

[16] SyTwo/Cmap on-line. See: http://www.yana.net/SyTwo/Cmap/index.html

[17] W. M. Wilson, L. H. Rosenberg, L. E. Hyatt, Automated Analysis of Requirement Specifications, ICSE 1997, IEEE Computer Society Press.

[18] D. Zowghi, V. Gervasi, A. McRae, Using Default Reasoning to Discover Inconsistencies in Natural Language Requirements, Proc. of the 8th Asia-Pacific Software Engineering Conference, Dec. 2001.

[19] Cmap tool on-line: see http://www.yana.net/cmap/

[21] D. Richards, K. Boettger, and O. Aguilera, A Controlled Language to Assist Conversion of Use Case Descriptions into Concept Lattices, LNAI 2557

Appendix 1

UC1:
Goal: Requestor buys something through the system
Primary Actor: Requestor
Scenario: - Requestor: initiate a request
- Approver: check money in the budget, check price of goods, complete request for submission
- Buyer: check contents of storage, find best vendor for goods
- Authorizer: validate Approver's signature
- Buyer: complete request for ordering, initiate PO with Vendor
- Vendor: deliver goods to Receiving, get receipt for delivery (out of scope of system under design)
- Receiver: register delivery, send goods to Requestor
- Requestor: mark request delivered

UC2:
Goal: Requestor, manager or buyer wants to see the state of the system
Primary Actor: Requestor or manager or Buyer or Receiver
Scenario: - Reader asks to see any one or any multiple requests sorted by any imaginable criteria.
- System: show purchases
- Readers asks to get report printed

UC3:
Goal: Change who can approve purchases, what purchase

limits they have
Primary Actor: Authorizer
Scenario: - Authorizer notify the current Approver the changes decided
- Authorizer notify the new Approver the changes decided

UC4:
Goal: Add, delete, change name, address, phone number of vendors
Primary Actor: Buyer
Scenario:

UC5:
Goal: Create a request in the system
Primary Actor: Requestor
Scenario: - Requestor asks to initiate a request
- Requestor fills in the request form
- Requestor sends the request to the Approver

UC6:
Goal: Change any part of a request
Primary Actor: Approver
Scenario:
- Approver changes the request
- Approver sends the changed request to the requestor

UC7:
Goal: Accept changed request
Primary Actor: Requestor
Scenario: Requestor send to the Approver the OK on the changed request

UC8:
Goal: Refuse changed request
Primary Actor: Requestor
Scenario: Requestor send to the Approver the cancellation of the request

UC9:
Goal: Cause processing on a request to stop
Primary Actor: Approver, Authorizer, Buyer
Scenario: - Actor cancels the request
- Actor send to the Requestor the confirmation of cancelled request

UC10:
Goal: Finalize a request as delivered OK, no more work need be done on it
Primary Actor: Approver
Scenario:

UC11:
Goal: Initiate process to return goods, keep from paying for them
Primary Actor: Buyer
Scenario:

UC12:
Goal: Complete approval process for a request
Primary Act**or**: Approver
Scenario: - Complete the forms for request
- Send to the Authorizer the request approval

UC13:
Goal: Complete all parts of request and initiate POs
Primary Actor: Buyer
Scenario:

UC14:
Goal: From one or more Requests, generate PO to a single vendor
Primary Actor: Buyer
Scenario: - Buyer generates the Po
- Buyer send the PO to the Vendor

UC15:
Goal: Notify Buyer that a delivery did not arrive on time
Primary Actor: Receiver
Scenario: - Receiver verifies that the due date is past without receiving the delivery
- Receiver send the Buyer notification that a delivery did not arrive on time

UC16:
Goal: Establish that the request approver really has the needed signature authority
Primary Actor: Authorizer
Scenario: - Authorizer validate approver's signature
- Authorizer send to the Buyer authorization to buy

UC17:
Goal: Mark actual delivery against one or more POs
Primary Actor: Receiver
Scenario: - Receiver register delivery
- Receiver send notification of delivered request to the Approver