Application of Linguistic Techniques for Use Case Analysis

A.Fantechi*, S.Gnesi^, G.Lami^, A. Maccari+
*Dip. di Sistemi e Informatica - Università di Firenze - Italy
fantechi@dsi.unifi.it
^Istituto di Elaborazione della Informazione - C.N.R. - Area della Ricerca C.N.R. - Italy
{gnesi, glami}@iei.pi.cnr.it
+ Nokia Research Center, Software Architecture Group - Finland
alessandro.maccari@nokia.com

Abstract

The Use Case formalism is an effective way of capturing both Business Process and Functional System Requirements in a very simple and easy-to-learn way. Use Cases are mainly composed of Natural Language (NL) sentences. The use of NL as a way to specify the behavior of a system is however a critical point, due to the inherent ambiguity originating from different possible. We discuss the use of methods based on a linguistic approach, that analyze functional requirements expressed by means of textual (NL) Use Cases. The aim is to collect quality metrics and detect defects related to such inherent ambiguity. In a series of preliminary experiments, we applied a number of tools for quality evaluation of NL text to an industrial Use Cases document. We also discuss the application of linguistic analysis techniques to support semantic analysis of NL expressed Use Case.

ACM Codes: D.2.1 Requirements/Spicification; D.2.8 Metrics.

1. Introduction

Use Cases [5] are a powerful tool to capture functional requirements for software systems. They allow structuring requirements according to user goals [6] and provide a means to specify the interaction between a certain software system and its environment.
Graphical object modeling languages have become very popular in recent years. Among those, UML [17] introduces a set of graphical notation elements for Use Case modeling. UML Use Case diagrams are easy to understand and constitute a good vehicle of communication. However, they mainly serve as a sort of table of content for Use Cases, presenting the connections between actors and Use Cases, and the dependencies between Use Cases.
System behavior cannot be specified in detail with Use Case diagrams. In his book [5], Alistair Cockburn presented an effective technique for specifying the interaction 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.
Natural language is powerful (the expression power of the English language is said to be higher than any other language in the world), well-known and generally easy to understand. However, it is also prone to ambiguities, redundancies, omissions and other defects that can lead to problems when precision and clarity are essential (it is the case of software requirements specifications particularly for embedded, mission-critical and performance-sensitive systems). Formal requirements specification languages (such as Z [22], B [1], LOTOS [4], etc.) were invented

secifically to tackle this problem. They add formality and remove ambiguity, but are hard to understand by non-experts, which limits their practical application to some restricted domains.
Nokia has started using the natural-language-based Use Case modeling technique to specify functional requirements for the mobile phone software user interface. As part of the project CAFÉ [9], we have initiated a joint research project on the use of methods based on a linguistic approach with the aim to collect metrics and perform a qualitative analysis of the requirements. We investigate the application of linguistic techniques to Use Cases to expand information to support semantic analysis and consistency checks. This paper is structured as follows: in section 2 we describe what Use Cases are and which kind of Use Cases we will consider; in section 3 we give an overview of some related works, placing our contribution therein; in section 4 we discuss the NL-based approach to analyze Use Cases, in section 5 we present three available tools for quality evaluation of NL components of Use Cases, along with the metrics that can be calculated by using them; in Section 6 we show the results of the application of the tools to an industrial case study, finally, in section 7 we discuss open research issues on the linguistic approach, mainly for the semantic analysis of Use Cases.

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. Every Use Case constitutes a goal-oriented set of interactions between external actors and the system under consideration. The term *actor* is used to describe any person or system that has a goal against the system under discussion or interacts with the system to achieve some other actor's goal. 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 the goal that is associated to it is reached. 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": 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 actors can 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 usually 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 execution path of a Use Case, and represents a single path through the Use Case that leads to success in achieving the goal (the Main Success Scenario). 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 [6].

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, Sub-function>	
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 the 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 after>
	2	<...>
	3	
Extensions	**Step**	**Branching Action**
	1a	<condition causing branching> : <action or name of sub-use case>
Sub-Variations		**Branching Action**
	1	<list of variations>

Figure 1. Use Case template

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. For instance, in the telecommunications domain, the use of UML sequence diagrams (formerly known as message sequence charts) is very popular. Unfortunately, this understanding process has rarely been documented in an effective manner. The research we perform is an attempt to improve the understanding process by identifying possible flaws in the textual scenario descriptions.

3. Related works on Natural Language Processing (NLP) applied to requirements

Several studies dealing with the evaluation and the achievement of quality in natural language requirement documents can be found in the literature. We will briefly discuss some that we consider to be of particular interest to our research.

Macias and Pulman [19] apply domain-independent Natural Language Processing (NLP) techniques to control the production of natural language requirements.
They propose the application of NLP techniques to requirements documents in order to control:

- the vocabulary used, which must be fixed and agreed upon, and
- the style of writing, i.e., a set of pre-determined rules that should be satisfied in order to make documents clear and simple to understand; they associate an ambiguity rate to sentences, depending on the degree of syntactic and semantic uncertainty of the sentence. The information is conveyed by discovering under-specifications, missing information, unconnected statements.

Finally, they discuss how NLP techniques can help the design of subsets of the English-grammar to limit the generation of ambiguous statements

Goldin and Berry [14] implemented a tool for the extraction of abstractions from natural language texts, i.e. of repeated segments identifying significant concepts on the application field of the problem at hand. The technique proposed is restricted to a strict lexical analysis of the text.

Hooks [15] discusses a set of quality characteristics necessary to produce well-defined natural language requirements. This paper discusses some common problems which arise when requirements are produced, and looks at how to avoid them. It provides an in depth survey of the principal sources of defects in natural language requirements and the related risks.

Wilson and others [24] examine the quality evaluation of natural language software requirements. Their approach defines a quality model composed of quality attributes and quality indicators, and develops an automatic tool to perform the analysis against the quality model aiming to detect defects and collect metrics.

Other works investigate how to handle ambiguity in requirements. In particular, Fuchs [13] proposes to solve the problems related to the use of NL in requirements documents by defining a limited natural language, called Attempt Controlled English (ACE), able to be easily understood by stakeholders and by any person involved into the software development process. This subset of English is simple enough to avoid ambiguities, so that domain specialists are allowed to express requirements using natural language expressions and to combine these with the rigor of formal specification languages.

Kamsties and Paech [18] focus especially on the ambiguity evaluation of natural language requirements. They start from the consideration that ambiguity in requirements is not just a linguistic-specific problem and put propose the idea of a checklist addressing not only linguistic ambiguity but also the ambiguity related to a particular domain.

Mich and Garigliano [20] propose a set of metrics for semantic and syntactic ambiguity in requirements. Their approach is based on the use of information on the possible meanings and roles of the words within a sentence and on the possible interpretation of a sentence. This is done using the functionalities of a tool called LOLITA.

Natt och Dag et al. [21] recently presented an approach based on statistical techniques for the similarity analysis of NL requirements aimed at identifying duplicate requirement pairs. This technique may be successfully used for revealing interdependencies and then may be used as a support for the consistency analysis of NL requirements. In fact, the automatic determination of

clusters of requirements dealing with the same arguments may support the human analysis, aimed at detecting inconsistencies and discrepancies, by focusing on smaller sets of requirements.

The approach proposed by Ambriola and Gervasi [3] to address the problem of achieving high quality NL requirements is to define a system that can build (semi)formal models in an almost automatic fashion, extracting information from the NL text of the requirements; the system can then measure and check the consistency of these models. This system can be profitably adopted as a means to induce the use of a suitable style in writing the requirements.
To our knowledge, NL processing techniques have not yet been applied to the analysis of Use Cases.
Therefore, the work we present has some novelty in that it examines a particular, yet crucially important and widely used, type of NL requirements document.
In the next sections, we will show the added value that can be obtained by focusing NL processing techniques on this particular approach for requirements definition.

4. Analysis of Use Cases by means of Natural Language-based techniques

Natural Language (NL) plays a relevant role in the specification of requirements by Use Cases because actors, actions, scenarios, responsibilities, goal etc. are specified in NL and the use of NL as a way to specify the behavior of a system is always a critical point, due to the inherent ambiguity originating from different interpretations of natural language descriptions.
The use of techniques for the linguistic analysis of natural language texts can be envisaged also to remove interpretation problems in requirements documents that are based on Use Case descriptions. The analysis made by means of NL-based techniques is useful to address several interpretation problems related to linguistic aspects of Use Cases. These problems may be grouped into three main categories:

- *Expressiveness* category: it includes those characteristics dealing with the understanding of the meaning of Use Cases by humans. In particular, we consider:
  - *Ambiguity mitigation*: detection and correction of linguistic ambiguities in the sentences;
  - *Understandability improvement*: evaluation of the understandability level of a requirements document and indication of those parts of it needing to be improved.
- *Consistency* category: it includes those characteristics dealing with the presence of semantics contradictions and structural incongruities in the NL requirements document.
- *Completeness* category: it includes those characteristics dealing with the lack of necessary parts within the requirements document.

The NL components of Use Cases (typically sentences), may be analysed from a lexical, syntactical or semantic point of view. For this reason it is proper to talk about, for example, lexical non-ambiguity or semantic non-ambiguity rather than non-ambiguity in general. 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.
Figure 2 shows schematically that the quality of NL requirements, and in particular of Use Cases, can be represented as a two-dimensional space, where the horizontal dimension is composed of the main target qualities to be achieved (Expressiveness, Consistency and Completeness) and the vertical dimension is composed of the different points of view from which the target qualities can be considered.

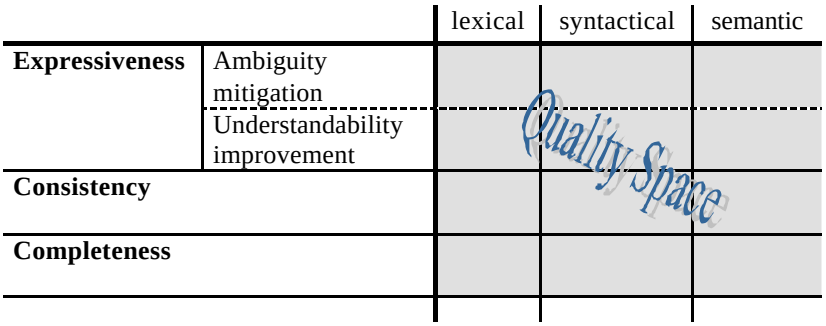

Figure 2. Two-dimensional representation of the NL requirements quality

NL based techniques are not sufficient to cover completely all the above issues, in particular consistency and completeness, since these also address questions about the semantics of the NL sentences. However, NL based techniques may provide a significant help in analysing expressiveness problems of Use Cases from a linguistic point of view. In particular, it is possible to provide measures for the evaluation of the quality of Use Cases defining some linguistic-based metrics derived from the application of a set of tools for the analysis of NL requirements documents.

We concentrate in this paper on expressiveness-related issues, leaving consistency and completeness problems to further studies, of which some preliminary results are anticipated in Section 7.

The expressiveness-related issues for *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 (i.e. that may have multiple meanings according to the context).
- by syntactical evaluation: using syntactical analysers to detect sentences having different interpretations on the basis of different derivation trees.

Understandability improvement may instead be improved in the following ways:

- by lexical evaluation: using lexical parsers both to detect poorly understandable wordings that decrease the readability of a document and to achieve readability indicators 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 etc.).
- by syntactical evaluation: using syntactical analysers to detect sentences having a too complex syntactical structure and hence hard to be understood (e.g. sentences with conjunctions, disjunctions, multiple subjects, objects, verbs).

5. Quality evaluation of Use Cases

The typical structure of Use Cases makes the NL analysis easier and more effective. Our objective is the application of methods and tools for the analysis of NL requirements documents in order to easily detect linguistic inaccuracies in Use Cases, in such a way that defects can be removed as early as possible.

To this aim, we define a set of metrics that can be used to evaluate the quality of requirements documents based on Use Cases, according to the categories listed in the previous section. We have addressed this problem starting from the definition of a set of metrics related to quality characteristics that fall in the Expressiveness category, that can be derived from the application of three different tools developed to perform linguistic analyses of NL requirements documents i.e.:

QuARS [12], ARM [24] , SyTwo [23]. This set of metrics are based on quality properties and quality indicators used by the considered tools to evaluate NL requirements.

5.1 QuARS

The tool QuARS (Quality Analyzer for Requirements Specifications) is based on the quality model, shown in Table 1 [11]. This quality model is composed of a set of high-level quality properties for NL requirements to be evaluated by means of syntactic and structural indicators. These are directly detectable looking at the sentences of a requirements document. The quality model has been defined by considering existing related literature and by taking advantage from matured experience in the field of requirement engineering and software process assessment according to the SPICE (ISO/IEC 15504) model [16].

The QuARS quality model, though not exhaustive, is sufficiently specific to include a significant part of lexical and syntax-related issues of requirements documents.

QuARS 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 such 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.

Property	*Indicator*	*Description*
Testability	Vagueness	It is pointed out when parts of the sentence hold inherent vagueness, i.e. words having a non uniquely quantifiable meaning
	Subjectivity	It is pointed out if the sentence refers to personal opinions or feeling
	Optionality	It reveals a requirement sentence containing an optional part
	Weakness	It is pointed out in a sentence when it contains a weak main verb
	Underspecification	It is pointed out in a sentence when the subject of the sentence contains a word identifying a class of objects, without a modifier specifying an instance of this class
Consistency	Under-reference	It is pointed out in a when a sentence contains explicit references to: - not numbered sentences, - documents not referenced in the document under analysis - entities not defined nor described in the document under analysis
Understandability	Multiplicity	It is pointed out if the sentence has more than one main verb or more than one direct or indirect complement that specifies its subject
	Implicity	It is pointed out in a sentence when the subject is generic rather than specific.
	Comment Frequency	CFI (Comment Frequency Index) = NC / NR where NC is the number of requirements having one or more comments, NR is the total number of requirements. A comment is intended to be a sentence, clearly identified by a keyword (e.g. "comment:", or "comment="), which aims to improve the understanding of the requirement that includes it.
	Unexplanation	It is pointed out when a sentence contains acronyms not explicitly and completely explained within the document under analysis

Table 1. The QuARS quality model

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.

5.2 ARM

The objective of the Automated Requirement Measurement Tool (ARM) is to provide measures that can be used to assess the quality of a requirements specification document [24]. ARM is not intended to be used for the evaluation of the correctness of a specified requirements document. This tool can be seen, similarly to QuARS, as an aid for “writing the requirements right,“ not “writing the right requirements”.

In ARM, a quality model similar to that defined for QUARS is employed; in this model, some quality indicators for requirements documents and specification statements have been defined, basing on a representative set of NASA requirements documents; the indicators have been identified on the basis of words, phrases, and linguistic structures that were considered related to quality attributes. These individual indicators have been grouped according to their indicative characteristics. For example, the categories related to single sentences are those presented in Table 2 together with their quality indicators. ARM scans a text file in search of such default indicators. The user can supply new domain-dependent quality indicators.

	IMPERATIVE	CONTINUANCE	DIRECTIVE	OPTION	WEAK PHRASES	INCOMPLETES
I	Shall	below:	e.g.	Can	adequate	TBD
N	Must	as follows:	i.e.	May	as appropriate	TBS
D	is required to	following:	For example	Optionally	be able to	TBE
I	are applicable	listed:	Figure		be capable of	TBC
C	are to	in particular:	Table		capability of/to	not defined
A	responsible for	support:	Note:		easy to	not determined
T	Will	and			effective	but not limited to
O	Should	:			as required	as a minimum
R					normal	
S					provide for	
					timely	

Table 2. Standard ARM Indicators

5.3 SyTwo

SyTwo is a tool developed as a Web application performing the linguistic analysis of an English text. This tool performs a lexical and syntactical analysis of a text, aimed to verify if it is suitable for being used in a requirement document. This tool can analyze the English text both to check its conformance to the rules of the Simplified English, and to detect some defect specifically conceived for evaluating the quality of requirements. To this aim, SyTwo partially adopts the QuARS quality model.

SyTwo builds, using a natural language grammar, the derivation trees of each sentence. During the analysis process, each syntactic node is associated with a feature structure which specifies morpho-syntactic data of the node and application-specific data, such as errors with respect to the quality

model. The output is composed of an error code, corresponding to a predefined type of defect, and of the indication of the part of the text the defects originate from.
Furthermore, SyTwo provide the value of the Coleman-Liau metrics for readability evaluation.
A syntactically ambiguous sentence can be pointed out, by SyTwo, when the sentence has more than one derivation tree: 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 (see figure 3):

1. The negation *not* of the auxiliary verb *shall* is related to the first verb only (*remove*), 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

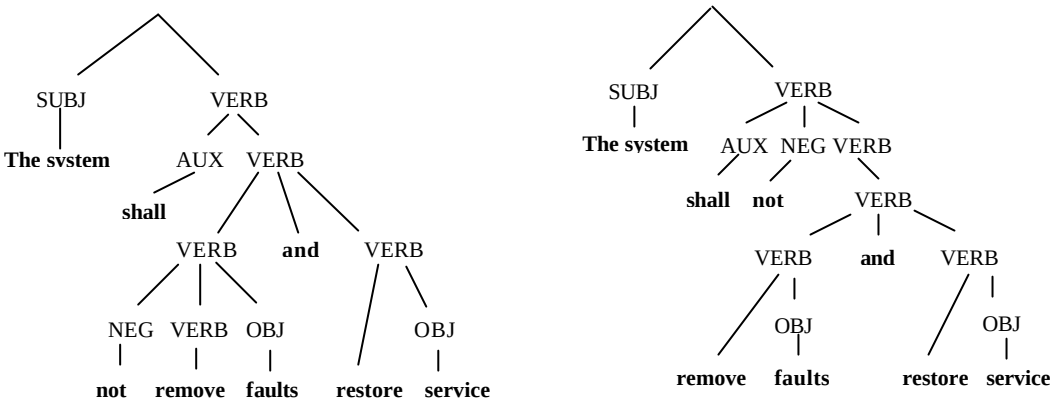

Figure 3: Two possible derivation trees

SyTwo is able to capture the syntactical structure of a sentence identifying its components and their syntactic role. From this information a component of SyTwo, called Cmap, is able to extract the relations among subjects, verbs and objects in a sentence, building the so called "conceptual maps", which we will show in section 7 to be useful to perform further analysis of requirements documents.

5.4 Achievable Metrics

As any other evaluation process, the quality evaluation of NL software requirements has to be conducted against a model. In this case the model is directly derivable from the Quality Models of the tools we are addressing to. The use of these tools allows some metrics, especially related to the Expressiveness category to be gathered in order to perform a quantitative evaluation of a requirements document.
These metrics are described in Table 3. The acronyms used in the Type column of Table 3 mean:
UN = Understandability,
RE = Readability,
TR = Traceability,
MA = Maintainability,
AM = Ambiguity,
SC = Specification Completion,
CS = Consistency.

Metrics	Type	Formula	*Rationale*
Coleman-Liau Formula	RE	5.89 * (Nl / Nw) - 0.3 * (Ns / (Nw / 100)) - 15.8. Where: Nl = n. of letters in the requirements document Nw = n. of words in the requirements document Ns = n. of requirement sentences in the requirements document	It measures the difficulty in reading the document
Average number of words per sentence	RE, UN	Nw / Ns	Short sentences make the requirements document more readable/ understandable
Continuance Index	TR, MA	N_{con}/Ns. Where: N_{con}= n. of continuances in sentences. Continuances are phrases as "the following:" that follow an imperative verb and precede the definition of lower level requirement specification (see Table 2)	The use of continuances indicates a well structured document, but too many continuances indicate multiple, complex requirements
Comment Frequency	UN	Nc / Ns. Where: Nc = n. of comment sentences. (see Table 1)	The comments within the requirements document reduce the risk of misinterpretations
Directives Frequency	UN	Nd / Ns. Where: Nd = n. of directives. Directives are words or phrases that indicate examples or other illustrative information	Directives make the document more understandable.
Multiplicity	UN	N_{mul} / Ns. Where: N_{mul} = n. of sentences having more than one main verb or more than one direct or indirect complement that specifies its subject.	The presence of multiple sentences makes the requirements document more difficult to be read and understood
Vagueness	AM	N_{Vag} / Ns. Where: N_{Vag} = n. of sentences including words holding inherent vagueness, i.e. words having a non uniquely quantifiable meaning.	The presence of vague sentences increases the level of ambiguity of the requirements document
Subjectivity	AM	N_{sub} / Ns. Where: N_{sub} = n. of sentences referrings to personal opinions or feelings.	The presence of subjective sentences increases the level of ambiguity of the document
Optionality	AM	N_{opt} / Ns. Where: N_{opt} = n. of sentences containing an optional part	The presence of optional sentences increases the level of ambiguity of the document
Weakness	AM	N_{wea} / Ns. Where: N_{wea} = n. of sentences containing a weak main verb.	The presence of weak sentences increases the level of ambiguity of the requirements document
Underspecification	SC	N_{usp} / Ns. Where: N_{usp} = n. of sentences having the subject containing a word identifying a class of objects without a specifier of this class.	The presence of underspecification makes the requirements document not fully specified
Implicity	UN	N_{imp} / Ns. Where: N_{imp} = n. of sentences having the subject generic rather than specific.	The presence of implicit sentences makes the requirements document prone to be misunderstood
Under-reference	CO	N_{ure} / Ns. Where: N_{ure} = n. of sentences containing explicit references to: -unidentified sentences of the requirements document itself; - documents not referenced into the requirements document itself - entities not defined nor described into the requirements document itself.	The presence of these references introduces inconsistencies in the requirements document
Unexplaination	UN	N_{une} / Ns. Where: N_{une} = n. of sentences containing acronyms not explicitly and completely explained within the requirements document itself.	The presence of acronyms not explicitly and completely explained makes the document prone to be misunderstood

Table 3. Metrics

6. A Case Study

As a case study, we have considered a requirements document, taken from an industrial project. We have analysed the document with QuARS, ARM and SyTwo. This document, provided by Nokia, describes the functional requirements for the user interface of a new feature (FM radio player) to be included in a line of mobile telephones. This feature should provide the possibility to use a phone as a built-in mono frequency modulation (FM) radio. The first product to include this feature has been the 8310 [26].The document analysed is composed of about one hundred Use Cases. The outcomes in terms of the proposed metrics are reported in Table 4. The information about the quality of the analyzed document provided by these metrics may be summarized as follows.
Concerning the values 1, 5 and 7, here are three samples of defective sentences related to these metrics taken from the analyzed Use Cases:
The user enters the frequency as described in the above procedure (Implicit sentence: indicator *above*).
In addition, the user is naturally able to adjust the volume (Vague sentence: indicator *naturally*)
The user can switch the radio on by selecting `Radio` *from the menu* (Under-specified sentence: indicator *menu*).
The word "menu" has been set as under-specified by the tool users. However, while generally the sentence must be recognized as under-specified, and it is good to have its under-specification pointed out by the tool, in this particular case the detection of the defect may not trigger any improvement actions on the document. This is because the user interface configuration and styling is done independently of (and after) component development and integration. Therefore, it may be a methodological choice to leave this defect unsolved until the very end of the software integration phase.

	Metrics name	Reference values	Actual Value	Used tool
1	Vagueness	The closer it is to 0 the more unambiguous the requirements document is	4	QuARS/ SyTwo / ARM2.1
2	Subjectivity	The closer it is to 0 the more unambiguous the requirements document is	0	QuARS/ SyTwo
3	Optionality	The closer it is to 0 the more unambiguous the requirements document is	0	QuARS
4	Weakness	The closer it is to 0 the more unambiguous the requirements document is	0	QuARS/ SyTwo/ ARM2.1
5	Under-specification	The closer it is to 0 the better specified the requirements document is	19	QuARS
6	Under-reference	The closer it is to 0 the more consistent the requirements document is	0	QuARS
7	Implicity	The closer it is to 0 the more understandable the requirements document is	12	QuARS
8	Unexplaination	The closer it is to 0 the more understandable the requirements document is	0	QuARS
9	Coleman-Liau Formula	Typically ranged from 0,4 (easy) to 16,3 (difficult)	17.6	SyTwo
10	Average number of words per sentence	Simple sentences have a number of words less than 10 - 12	14,82	QuARS
11	Continuance Index	Optimal range: 0.1 – 0.2	0	ARM 2.1
12	Comment Frequency	Optimal range: 0.1 – 0.3	0,04	QuARS
13	Directives Frequency	Optimal range: 0.1 – 0.3	0,08	ARM 2.1
14	Multiplicity	The closer it is to 0 the more understandable the requirements document is	12	QuARS

Table 4. Metrics values

In general, a high value of these indicators points towards an inaccurate selection of the terms used in the Use Case.
The values of metrics 9, 10 and 14 indicate that the sentences need to be simplified in order to decrease the risk to be misinterpreted. Below a sample of a multiple sentence taken again from the analysed Use Cases:
The phone displays the confirmation note Frequency set and goes to the FM Radio state displaying the selected frequency with the channel number and name if a channel in that frequency has already been saved earlier.
The values of metrics 12 and 13 seem to indicate that the document is poor of extra information that might make it more understandable. However, the reference values for these two metrics are derived from the good practices of NL requirements, and they could be not fully significant for Use Case requirements, because these kinds of requirements specifications are inherently more descriptive.

7. Open research issues

To effectively address the Consistency and Completeness aspects of requirements specifications, we should resort to their formalization [10, 25]. 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 section we discuss the application of NL based techniques that can provide an effective support to deal with Consistency and Completeness issues. The way we intend to follow is based on the study of the functional relations, i.e. the relations or dependencies between two significant entities (typically actors) of Use Cases.
We can observe that a system specification written as Use Cases is structured in three semantic layers:

1) the specification is, at its higher level, composed by a set of Use Cases plus other artefacts and models; each Use Case defines a goal for a primary actor and some secondary actors, establishing relations among actors.
2) in each Use Case the scenario and its extensions play a major role in specifying the system operation; that is they define the sequential control flow, with exceptions defined by the extensions.
3) each scenario or extension sentence has its internal, linguistic structure, which defines a relation among (primary and secondary) actors and the operations they perform or take part into.

It is on the third layer that the linguistic analysis has an immediate application, but the structure of the previous layers give important information as well. Indeed, works aiming at the improvement of the correctness of requirements relying on the Use Cases structure exist already [2, 8].
Our aim is the definition of a relational structure combining both the results of the linguistic analysis on such sentences and the structure implied by the other layers.
The availability of the functional relations 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.
Interaction schemata can be derived by the relations; these interaction schemata may be analysed in search of undesired, inconsistent and incomplete dynamic behaviour of the system, for example, an

interaction schema containing a loop may point to a possible synchronization problem (such as a deadlock).
These schemata may also form the basis for a formal analysis of interactions, which we do not address in this context, leaving it to future works.
While on the first two layers relations are explicit in the structure of the Use Case description, on the third layer relations can be determined by looking at the syntactical structure of each sentence of a Use Case description, defining a set of items where each primary actor has been put in relation with the secondary actors according to the used verb.
The techniques for extracting such information from Use Cases documents can be based on available NLP techniques as the Functional Dependency Grammar (FDG) parsers. These are tools able to enrich text (plain ASCII or in advanced formats such as XML, SGML, HTML) with functional dependencies that tell about sentence-level relations and functions between words and linguistic structures. Figure 4 shows the outcomes of the functional analysis of the sentence "Insurance company verifies all details are within policy guidelines", performed using the FDG Conexor tool [7]. The outcome says that the item number 1 `insurance` is an attribute of the item number 2 `company` that is on its side, the subject of the item number 3 (the verb `verify`) and so on. The last column of Figure 4 contains functional, surface syntactic and morphological tags useful for further linguistic analysis.

```
 1 insurance      insurance     attr:>2      @A> %>N N NOM SG
 2 company        company       subj:>3      @SUBJ %NH N NOM
 3 verifies       verify                     @+FMAINV %VA V PRES SG3
 4 all            all           det:>5       @DN> %>N DET
 5 details        detail        obj:>3       @OBJ %NH N NOM PL
 6 are            be                         @+FMAINV %VA V PRES
 7 within         within        ha:>6        @ADVL %EH PREP
 8 policy         policy        attr:>9      @A> %>N N NOM SG
 9 guidelines     guideline     pcomp:>7     @<P %NH N NOM PL
10 <s>            <s>
```

Figure 4. Functional analysis of a sentence

Similar information can be obtained by the conceptual maps produced by the Cmap component of the SyTwo tool described in section 5.
The organization and the analysis of the relations that can be extracted from a Use Case description using these techniques will be the subject of our next research activity.

8. Conclusions

Requirements specification by means of Use Cases allows functional requirements to be captured in an effective way by means of scenarios. Developers have always used typical scenarios (often in graphical form) in order to understand what the requirements of a system are and how a system works; Use Cases provide a means to rigorously express requirements along these lines.
When expressing goals, scenarios and conditions with NL, it is necessary to identify the defects due to the inherent ambiguity of NL (for instance: vagueness, poor specification and poor understandability). For this reason, tools and techniques that were conceived for "traditional" textual requirements can be effectively applied to Use Cases to detect defects and collect metrics.

We have proposed the use of available linguistic techniques to support the semantic analysis of Use Cases. Linguistic techniques may provide an effective support towards the achievement of quality requirements, but are not sufficient to completely address the aspects of correctness and consistency check of requirements. The rich structural information of Use Cases adds new possibilities in this direction, when combined with linguistic analysis of their textual information.

Acknowledgements
Part of the research work that is described in this paper was performed under the Eureka ?2023 Programme, ITEA (ip00004, CAFÉ [9]).

9. References

[1] Abrial JR. The B Book - Assigning Programs to Meanings. Cambridge University Press, August 1996.
[2] Alspaugh TA, Antòn AI. Scenario Networks: A Case Study of the Enhanced Messaging System, REFSQ'01, Interlaken, Switzerland, June 2001.
[3] Ambriola V, Gervasi V. Processing Natural Language Requirements, 12th IEEE Conf. On Automated Software Engineering (ASE'97), IEEE Computer Society Press, Nov. 1997.
[4] Bolognesi T, Brinksma E. Introduction to the ISO Specification Language LOTOS. Computer Networks, 14 (1), 25-59, 1987.
[5] Cockburn A. Writing Effective Use Cases, Addison-Wesley, 2000.
[6] Cockburn A. Structuring Use Cases with goals, Journal of Object-Oriented Programming, Sep-Oct 1997 (part I) and Nov-Dec 1997 (part II).
[7] Conexor tool. See http://www.conexoroy.com/
[8] Dutoit AH, Peach B. Developing Guidance and Tool Support for Rationale-based Use Case Specification, REFSQ'01, Interlaken, Switzerland, June 2001.
[9] Van der Linden F. Software Product Families in Europe : The ESAPS & Café Projects IEEE Software july/August 2002.
[10] Fantechi A, Gnesi S, Ristori G, Carenini M, Vanocchi M, Moreschini P. 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.
[11] Fabbrini F, Fusani M, Gnesi S, Lami G. An Automatic Quality Evaluation for Natural Language Requirements, 7th International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ'01, Interlaken, Switzerland, 2001.
[12] Fabbrini F, Fusani M, Gnesi S, Lami G. 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.
[13] Fuchs NE, Schwitter R. Specifying Logic Programs in Controlled Natural Language, Workshop on Computational Logic for Natural Language Processing, Edinburgh, April 3-5, 1995.
[14] Goldin L, Berry DM. Abstfinder, a prototype Abstraction Finder for Natural Language Text for Use in Requirements Elicitation: Design, Methodology, and Evaluation. First International Conference on Requirements Engineering, 1994.
[15] Hooks I. Writing Good Requirements, Proc. Of the Fourth International Symposium of the NCOSE , 1994, Vol. 2., pp. 197-203.
[16] ISO/IEC TR 15504 (Parts 1-9), 1998
[17] Jacobson I, Booch G, Rumbaugh J. The Unified Modelling Language Reference Manual. Addison-Wesley, 1999.
[18] Kamsties E, Peach B. Taming Ambiguity in Natural Language Requirements, ICSSEA 2000, Paris, December 2000.
[19] Macias B, Pulman SG. Natural Language processing for requirement specifications. In Redmill and Anderson, Safety Critical Systems, pages 57-89. Chapman and Hall, 1993.

[20] Mich L, Garigliano R. Ambiguity Measures in Requirement Engineering. Int. Conf. On Software Theory and Practice - ICS 2000, Beijing, China, Aug. 2000.
[21] Natt och Dag J, Regnell B, Carlshamre P, Andersson M, Karlsson J. Evaluating Automated Support for Requirements Similarity Analysis in Market-Driven development Seventh International Workshop on Requirements Engineering: Foundation for Software Quality, Interlaken, Switzerland, June 4-5 2001.
[22] Spivey JM. The Z Notation: A Reference Manual, 2nd edn., London Prentice-Hall, 1992.
[23] SyTwo on-line. See: http://www.yana.net/sytwo/index.html
[24] Wilson WM, Rosenberg LH. Hyatt LE. Automated Analysis of Requirement Specifications. Proceedings of the Nineteenth International Conference on Software Engineering (ICSE-97), Boston, MA, May 17-23, Pages: 161 -171.
[25] Zowghi D, Gervasi V, McRae A. Using Default Reasoning to Discover Inconsistencies in Natural Language Requirements, Proc. of the 8th Asia-Pacific Software Engineering Conference, Dec. 2001.
[26] http://www.nokia.com/phones/8310/index.html.