A Content Analysis Technique for Inconsistency Detection in Software Requirements Documents

Alessandro Fantechi, Emilio Spinicci

Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze
Via di S. Marta, 3 - 50139 Firenze (Italy)
Phone: +39 055 4796265
Fax: +39 055 4796363
{fantechi, spinicci}@dsi.unifi.it

Abstract. This paper presents J-RAn (Java Requirement Analyzer), a tool that implements a novel Content Analysis technique to support the verification of consistency and completeness of a Software Requirement Specification. This technique exploits the extraction, from a requirement document, of the interactions between the entities described in the document as Subject-Action-Object (SAO) triples (obtainable using a suitable syntactic parser). SAO triples represent a concept in its most synthesizing form. Analyzing the distribution of such concepts in the requirement document helps to locate possible sources of inconsistency and incompleteness.

1 Introduction

The use of Natural Language is in practice the most common way for specifying software requirements, because of the easy comprehension and sharing among all people involved in the software development process, from the stakeholders to the designers. However, the inherent ambiguity of Natural Language may lead to incorrect or inconsistent definition of requirement specifications, with the introduction of semantic contradictions or the lack of necessary information.

Several Natural Language Processing techniques and tools have been developed and applied to software requirements documents for consistency and completeness verification, by revealing sentences affected by ambiguity: among them, tools for lexical evaluation, such as QuARS [6] and ARM [17] provide structural and quality indicators on the basis of a suitable model, defined considering the existing literature and experiences in the field of requirement engineering and software process assessment, to detect and possibly correct terms or wordings that are ambiguous: for example, QuARS provides defect indicators for the testability of a requirement description in order to highlight defective sentences in terms of vagueness (that is, the sentence contains words having a non uniquely quantifiable meaning), subjectivity (if a sentence refers to personal opinions) optionality (a sentence containing one or more optional terms), and so on.

Tools for syntactical evaluation exploit instead syntactical analyzers to detect sentences having different interpretations, as in the case of the tools LOLITA [8, 9] and Circe-Cico [1]. LOLITA is a general purpose NLP tool that is able to identify the morphological, grammatical, semantic and pragmatic relations of a sentence, by representing them with a semantic net, which is used as a knowledge base of the system. The tool allows for the extraction of parsing trees that represent possible interpretations of a sentence; the most likely one is identified by computing ambiguity measures on the parsing trees, using statistical techniques. The Circe-Cico environment provides a support for identifying, selecting and validating NL requirements, with the integration of information from the application domain, by producing parsing trees of the input NL documents. In its latest version [2, 3], the Circe-Cico tool allows the formalization of NL requirements in Abstract State Machines (ASM) or UML diagrams.

In this work, we introduce an alternative way for the detection of inconsistent descriptions of software requirements, by exploiting Linguistic techniques successfully applied to the analysis of patent documents, for the extraction of information related to the novelty of a given invention [4]. Such techniques are based on the identification of Subject-Action-Object (SAO) triads, which represent the concepts expressed in a sentence in the most synthesizing form. The Java Requirement Analyzer (J-RAn) tool [4] is capable to automatically extract from the requirement document each paragraph in Natural Language related to a requirement description; sentences of these paragraphs are then processed by a syntactic parser to extract SAO triads, that is, information related to all the interactions carried out or received by entities detailed in the specification; the tool allows these interactions to be modelled in the form of a UML Class Diagram in XMI format. However, the element of novelty in J-RAn is constituted by the implementation of a SAO-based Content Analysis technique, which returns a measure of how much a paragraph deals with a topic of interest: for example, a software functionality specified in a group of requirements. These measurements are obtained by assigning a score to each SAO, according to suitable weights for Subject, Action, Object and to a set of dictionaries related to the functionalities to be investigated. In the case of software requirements descriptions, these scores can help to detect in the document possible redundant or missing information, which can constitute a source of potential ambiguity or inconsistency. The latest development of the tool allow for the generation of Content Analysis dictionaries by exploiting the automatic selection of SAOs belonging to the description of the software functionalities under investigation. In the next sections, the structure of the J-RAn tool will be presented, together with a detailed description of SAO-based Content Analysis and an example of application.

2 The Java Requirement Analyzer

Information Extraction from Software Requirement Specifications is made difficult by the fact that, in general, the use of standardized document models and writing rules cannot be assumed for this kind of technical documentation; actually, companies and organizations use their own templates to write their SRSs. What can at least be assumed is that each requirement should be identified by a reference or paragraph tag, which can vary depending on the document model; J-RAn is able to extract the descriptive paragraphs related to each software requirement, by searching each requirement tag matching the tag format, defined by the user by means of regular expressions. After loading a document, a suitable extractor module selects requirement paragraphs and sentences detailing software requirements, using the tag format previously defined. The extraction of paragraphs and of the corresponding sentences is implemented using Phrasys NLP components [10]. Sentences related to each paragraph are then processed by Link Grammar [12], a syntactic parser whose output is used to generate the set of SAO triads. The availability of this information allows SAO-based Content Analysis and UML Class Diagram generation to be performed. Figure 1 shows the sequence of operations J-RAn

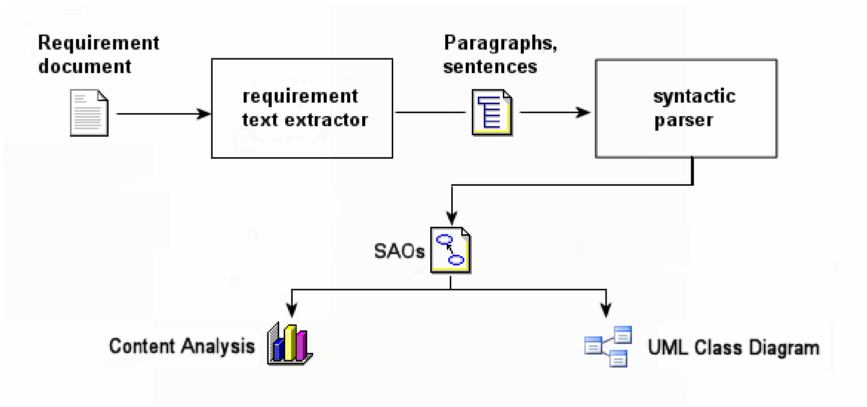


Figure 1. Processing chain of Software Requirement documents in J-RAn.

performs to process a requirement document.

Next subsections discuss the generation of paragraphs and sentences related to each requirement description, and the extraction process of SAO triples. Next releases of the tool will allow also structured requirement definitions to be imported from requirement management tools, such as Telelogic Doors.

2.1 Extracting Requirements using Phrasys NLP components

Phrasys is a collection of NLP modules, available at no cost for evaluation purposes, implemented in Java following the Java Beans conventions. The modules of the collection are chained each other by means of suitable *listeners,* which handle document processing through *events.* Events carry information that is first processed and then passed from one component to the next.

J-RAn incorporates the Phrasys components needed for the extraction of the paragraphs and sentences related to each requirement description: the *LineReader*, a basic component that generates text lines from an input file, and the *Sentencer*, which splits input text lines into paragraphs and sentences.

The *Sentencer* receives *LineEvents* from the *LineReader* component, converts these *LineEvents* into *SentenceEvents* and *ParagraphEvents,* and passes them on to registered listeners. *ParagraphEvents* of requirement descriptions, selected accordingly with the defined tag format, and the associated *SentenceEvents* are assembled in a tree structure visualized in the Analysis Result window (Figure 3). Each extracted sentence embodied in the *SentenceEvents* is then passed to Link Grammar for syntactic parsing and SAO extraction.

The *Lemmatiser* component of Phrasys is actually incorporated in J-RAn as well, and can be optionally selected, after the processing by Link Grammar, to reduce elements of SAOs in their base form: this will allow the use of a synonym dictionary for Content Analysis in future releases of the tool.

2.2 Extracting SAO triples using Link Grammar parser

Link Grammar is a syntactic parser for English language, released under GNU General Public License and based on the link grammar theory [13]: the parser assigns to a given sentence a syntactic structure, which consists of a set of labelled links connecting pairs of words.

To find a syntactic structure, Link Grammar uses a dictionary that incorporates Wordnet and a set of rules for linking words, that is, each word is assigned a set of connectors that allow a linkage with another word in the sentence. A sentence defined by the link grammar is a sequence whose words are connected so to verify the following conditions:

- *Satisfaction*: the links between each pair of words meet the linking rules, that is, each word is linked in a pair according with the assigned connectors;
- *Planarity*: the links do not cross;
- *Connectivity*: all words and related links form a fully connected graph.

A parsed sentence may exhibit multiple linkages, which can be shown either in a textual or graphical output, both detailing the connectors linking the words in the structure. While determining a syntactic structure, Link Grammar assigns a cost vector to each complete linkage found, taking into account the number of unused words, the relative size of components combined with conjunctions, and the total length of the linkage.

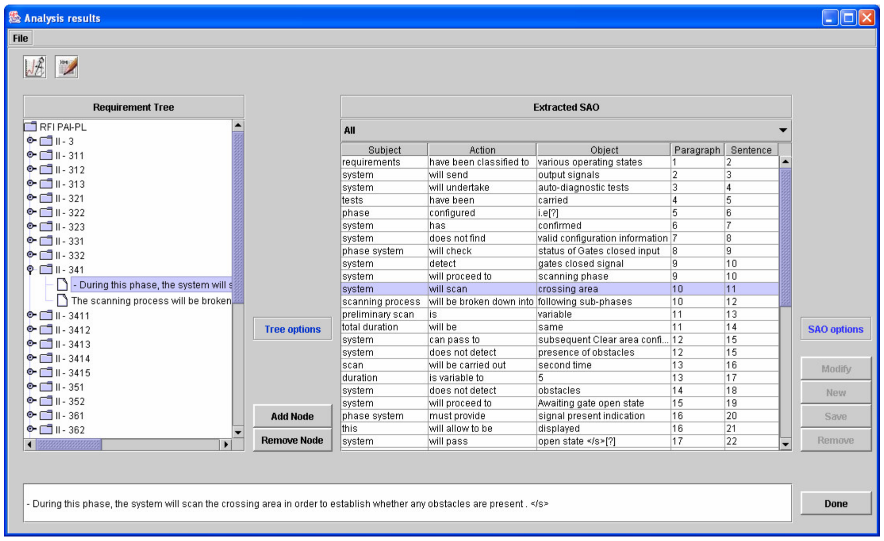
J-RAn selects the best cost vector linkage, which corresponds to the most likely interpretation of the sentence, among all the linkages returned by Link Grammar exploiting the output parser in the textual form. During the parsing, Link Grammar splits the sentence into individual words, or *tokens*, and annotate extracted tokens with essential classification labels (*Part-Of-Speech* tags) such as ".n" (noun) or ".v" (verb), needed by the parser to identify the subject, the verb and the object in the sentence.

The SAO extraction is performed by J-RAn locating any verb and its connected auxiliaries in the sentence (searching for ".v" POS-tags); once the verb has been found, the subject and the object can be located if any "S" or "O" connector is found linking the verb with other tokens of the sentence.

The following is an example of SAO extraction from the sentence "*During this phase, the system will scan the crossing area …*": Figure 2 shows the part of interest of the graphic output after Link Grammar parsing; the subject and the object linked to the verb "will scan" are the tokens "system" and "area". To complete the extraction, further tokens may be added following any attribute ("AN") or determinative ("D") connectors. The resulting SAO is "system – will scan – crossing area".

```
 +------------------------------------------Xp-------------------------- ------+
 +---------------Wd-----------------+                                          |
 |     +-------------CO-------------+              +------------MVp------------+
 |     +--------Xc--------+         |              +----------Os---------+     |
 |     +------Js-----+    |         |              |    +-------D*u------+     |
 |     |      +--Dsu-+    |  +--Ds--+---Ss--+---I--+    |       +---AN---+     |
 |     |      |      |    |  |      |      |       |    |       |        |     |
//  during this.d phase.n , the system.n will.v scan.v the crossing.n area.n  //
```

Figure 2. Partial parsing provided by Link Grammar.



Subject	Action	Object	Paragraph	Sentence
requirements	have been classified to	various operating states	1	2
system	will send	output signals	2	3
system	will undertake	auto-diagnostic tests	3	4
tests	have been	carried	4	5
phase	configured	i.e[?]	5	6
system	has	confirmed	6	7
system	does not find	valid configuration information	7	8
phase system	will check	status of Gates closed input	8	9
system	detect	gates closed signal	9	10
system	will proceed to	scanning phase	9	10
system	will scan	crossing area	10	11
scanning process	will be broken down into	following sub-phases	10	12
preliminary scan	is	variable	11	13
total duration	will be	same	11	14
system	can pass to	subsequent Clear area confi...	12	15
system	does not detect	presence of obstacles	12	15
scan	will be carried out	second time	13	16
duration	is variable to	5	13	17
system	does not detect	obstacles	14	18
system	will proceed to	Awaiting gate open state	15	19
phase system	must provide	signal present indication	16	20
this	will allow to be	displayed	16	21
system	will pass	open state </s>[?]	17	22

- During this phase, the system will scan the crossing area in order to establish whether any obstacles are present . </s>

Figure 3. J-RAn Analysis Result window.

SAO extraction may be performed also from sentences in passive form: passive sentences exhibit a subject receiving some action provided by an agent, which appears in the sentence in place of the object. Passive sentences may be processed referring to proper connectors, and paying attention to reverse subject and agent of the action. In this case, the extraction process also requires the verb to be lemmatized.

The full list of extracted SAOs, together with the structure of requirement paragraphs, is resumed in the Analysis Result window of Figure 3. The SAO extraction, as well as the other steps of the processing chain, is conducted interactively, so enabling the user to edit the results by adding missing SAOs and by modifying or deleting wrong extractions.

3 SAO-based Content Analysis

Content Analysis is an empirical research method to analyze the content of a communication, mostly text but not limited to it, renowned in the field of social science studies [7, 15]. The purpose is to draw inferences from the text using a category system: a content analysis of a text addresses specific aspects of interest and does not refer to the whole meaning of the text.

Text mining is a particular method within content analysis that is limited to identify and represent units of text automatically. Text mining techniques are often based on statistical procedures and use the co-occurences of words, or more precisely, character strings.

In the work of Siemens [11], basic text mining techniques for content analysis are explained: the use of dictionaries and glossaries of keywords allows for producing a statistics of their occurrences in a text, and for detecting if the content of a document deals with a topic of interest; the located document sections can be tagged to improve searching operations.

Content analysis has been applied to a wide range of research studies, as well as integrated into several kinds of information retrieval systems: its basic principles can also be found in the IBM's WebFountain [16], to cite the most recent text mining application to the analysis of web documents.

SAO-based Content Analysis consists in using suitable dictionaries of verbs and terms related to a topic of interest, and in assigning a score to each SAO if its subject, verb and object belong to the adopted dictionaries. By giving different weights to subject, verb and object, it is possible to put in evidence functional relationships or terminology accordingly to the purpose of the analysis. Reduced weights are assigned in the score if a partial matching between SAOs and dictionary elements occurs.

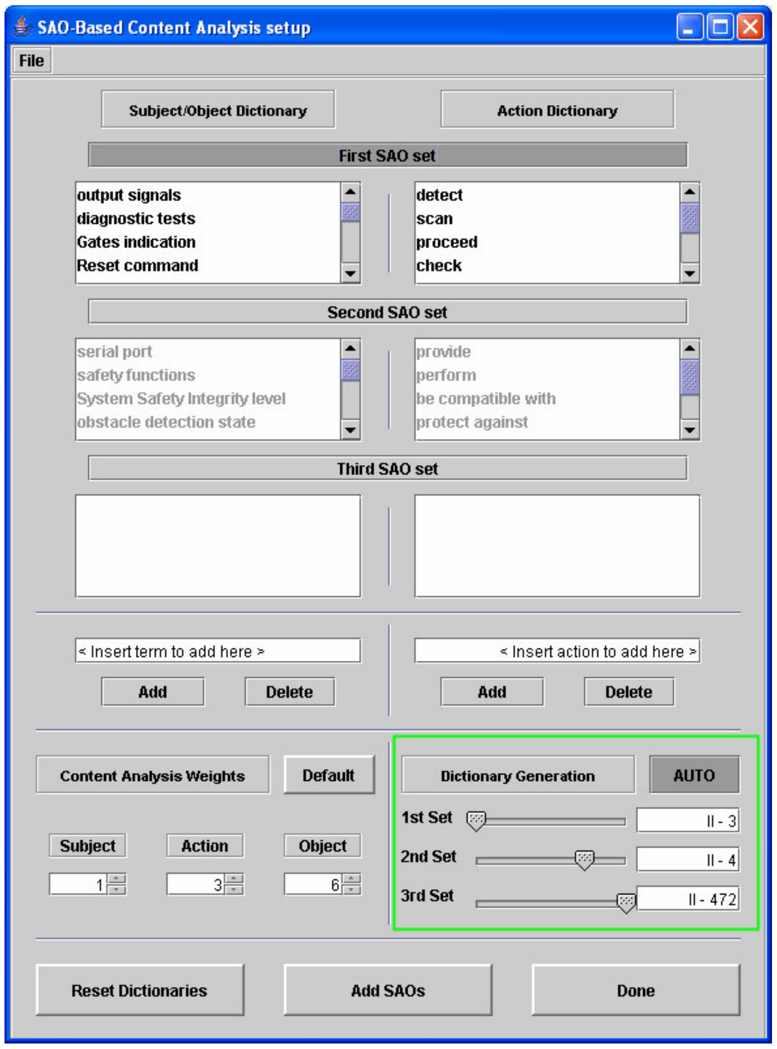
The scores of SAOs belonging to a given sentence say how much this sentence deals with the topic of interest. A suitable chart showing the score vs. sentence/paragraph number immediately shows whether specific parts of a document deal with the topic of interest, and in which measure.

In the context of requirements documents, SAO-based Content Analysis can be used as a mean to detect possible inconsistencies, by verifying for example whether requirements corresponding to a given functionality are expressed only in the desired sections of the document: if other paragraphs are dealing with the same functionality, the document probably contains a redundancy, that is a possible source of inconsistencies.

On the other hand, SAO-based Content Analysis can also help to verify the completeness in the definition of a software functionality, i.e. the presence of all the necessary parts in a requirement specification using properly tailored dictionaries.

The experiments we have conducted on a software requirements document use a score assignment in which the subject is unfavoured (weights: Subject:1, Verb:3, Object:6). This is due to a common situation in software requirements, where only few actors (the system, the user, the interface, and so on) are used as subjects of the prescribed actions; hence, most of the extracted SAOs present the same subject. A lower weight for the subject will avoid a noisy Content Analysis chart.

J-RAn currently offers three different types of Content Analysis charts: an Average, Max or Cumulative chart can be visualized, all of them versus requirement paragraphs. The first chart exploits the average SAO score per paragraph, providing the average relevance of a paragraph according to the used software functionality dictionary. The second chart uses the maximum score obtained by SAOs belonging to each paragraph, and highlights the document locations where the maximum relevance with the topic of interest has been reached. The third chart uses the sum of all the SAOs scores in each



First SAO set

output signals	detect
diagnostic tests	scan
Gates indication	proceed
Reset command	check

Second SAO set

serial port	provide
safety functions	perform
System Safety Integrity level	be compatible with
obstacle detection state	protect against

Subject 1 Action 3 Object 6

1st Set II - 3
2nd Set II - 4
3rd Set II - 472

Figure 4. The Dictionary Generation feature in Content Analysis Setup .

paragraph, quantifying in this way the relevance of a paragraph with respect to the software functionality under investigation.

Finally, multiple dictionaries can be used simultaneously, one for each of the analyzed functionalities, in order to generate a chart where the relevance of a paragraph with respect to each functionality can be compared. Current implementation of J-RAn allows to investigate up to three different functionalities (i.e. groups of software requirements located in three different regions of the document); next releases will allow the number of dictionaries to be defined by the user.

3.1 Automatic Dictionary Generation

In the experiments conducted on requirement documents, compiling correct dictionaries for SAO-based Content Analysis has resulted quite expensive in terms of time spent in collecting keywords for the software functionalities under investigation: these keywords should be adherent enough to the extracted SAOs in order to be able to detect the content of a paragraph for each functionality, but not too specific, in order to prevent possible inconsistencies (i.e. description of a functionality detected in paragraphs not belonging to it) to remain undetected. Therefore, the current release of J-RAn implements an automatic generation feature to help the user in compiling Content Analysis dictionaries: Figure 4 points out a panel in the Content Analysis Setup window where suitable sliders are used to mark the start and the end of the three regions of the requirement document containing the functionalities to be investigated: if the automatic generation button is switched to “AUTO”, dictionaries for each document region are automatically compiled after SAO extraction, by including Subject, Action and Object from the requirement descriptions previously marked. Disabling automatic generation allows the manual compilation of the dictionary; the user is able to manually modify each keyword anyway. Possible manual corrections or modifications made by the user to the set of extracted SAO in the Analysis Result window will automatically update the generated dictionaries in the Content Analysis Setup window.

3.2 Application example

The example of application of the SAO-based Content Analysis technique presented in the following refers to a requirement document of a railway grade crossing protection control system. The analyzed document is composed of two sections, one related to the functional requirements of the sys-

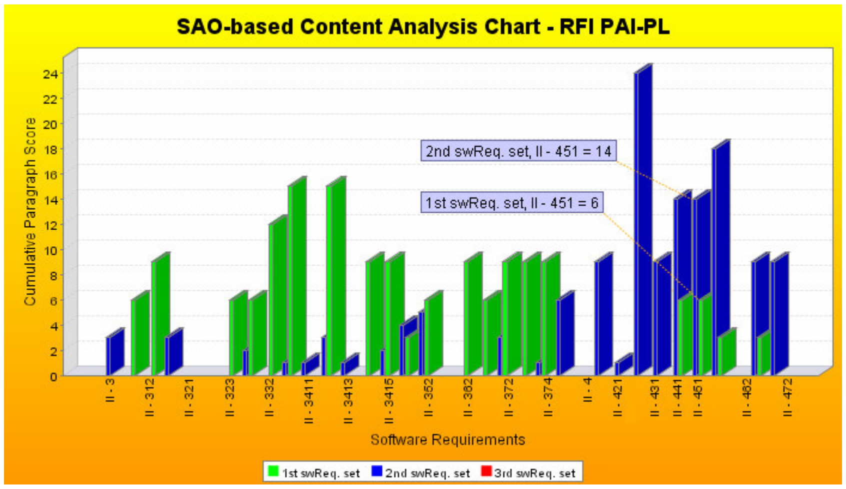


Figure 5. Content Analysis Chart generated by J-RAn.

tems, i.e. those including all the operations the system should perform on the crossing area, and the other with the environmental requirements, describing the required characteristics of the areas where the system will operate, as well as any external factors which could affect system operation. Figure 5 shows a Cumulative Chart generated after applying SAO-based Content Analysis to the two previous categories of requirements selected for investigation: functional requirements (light grey bars) and environmental requirements (dark grey bars). The dictionaries for content analysis have been generated according with the corresponding SAOs: for example, the functional requirements dictionary comprises terms like “obstacle”, “crossing area”, “diagnostic test”, and actions like “detect”, “scan”, “perform”; the environmental requirements dictionary contains terms like “safety functions”, “environmental standards”, “interferences”, and actions like “provide”, “be compatible” and so on.

The obtained chart shows the distribution into two groups of light and dark grey bars, according with the two main sections of the document. Higher scores belong to those paragraphs where more than one SAO has been extracted (e.g. where the paragraph contains more sentences), and therefore, where the description of the requirement has resulted to be more detailed. For both sections of the document, lower bars of the opposite category may have been originated because of a partial (and minimum) matching between SAO elements and the terms contained in the opposite dictionary: for example, “state” can be found in both dictionaries, as well as “obstacle”.

Relevant values of a requirement category in a paragraph not belonging to the corresponding document section require the related requirement description to be checked: in the example, an interesting consideration can be made in examining the environmental requirement labelled *II-451*: as emphasized in Figure 5, the score of the dark grey bar of the paragraph, equals to 14, derives from the SAOs extracted from the following (shortened) description: "*Any human errors arising during the installation and maintenance […] shall not result in the generation of a clear area confirmation signal. The system must pass to a vital error state or to a false obstacle detection state.* ". The last sentence originates two SAOs, "system – must pass – vital error state" and "system – must pass – false obstacle detection state", which are assigned a value of 6 caused by a partial matching with the terms "error state", "obstacle detection state" in the dictionary used for functional requirements, thus generating a not negligible light grey bar. Indeed, the last sentence constitutes a redundant definition of a functional requirement: it should be moved in the corresponding section of the document, or integrated in the functional requirement corresponding to the definition of system operating states, and replaced in requirement *II–451* with a reference to the interested functional requirement. The current definition may originate inconsistencies in the case of modification to functional requirements dealing with operating states of the system, thus not affecting the environmental requirements section. Similar considerations can be made for requirement *II–441*.

4 Conclusions

We have presented the Java Requirement Analyzer (J-RAn), a tool capable to perform a functional analysis of software requirements documents by generating the Subject-Action-Object (SAO) triples, via a syntactic parsing of the sentences in Natural Language extracted from the requirements descriptions. The set of generated SAOs is used to perform a SAO-based Content Analysis as a mean to verify the completeness of requirement definitions and to reveal possible sources of inconsistencies, such as redundant descriptions.

J-RAn is still at a prototypal stage, and is used as a workbench for experimenting the adoption of Content Analysis techniques as well as other Natural Language Processing techniques for requirement documents evaluation and analysis. The experimentation of the tool is however still limited, hence it is too early to give general results on the effectiveness of the approach supported by J-RAn. For the application example presented in this paper, we have experienced a 63% of correctly extracted SAOs, while wrong SAOs and missing extractions have been resulted respectively in a 21% and in a 16%.

Wrong and missing extractions are related partially to inaccurate parsing and partially to a defective processing of Link Grammar output; a refinement of the extraction rules by exploiting more of the connectors provided by Link Grammar is thought to allow J-RAn to improve the above figures. Unsatisfactory experiments have been carried out with other parsers, achieving lower percentages of correct extractions.

Other improvements and extensions that are planned for J-RAn include:

- integrating the tool within academic tools for requirements documents management, such as REM [5], as well as commercial ones, such as Requisite Pro and Doors;
- replacing Phrasys with customized libraries for the extraction of requirement descriptions, and integrating in J-RAn free academic POS-tagging and lemmatizing tools, such as TreeTagger [14], to improve extraction of SAOs and dictionary generation.

Experimentation with several requirement documents coming from diverse industrial sectors is also a priority, as well as an assessment of the impact and cost/benefit relationships in using the tool on a large amount of documents.

References

1. V. Ambriola, V. Gervasi, *Experiences with Domain-Based Parsing of Natural Language Requirements*, Proc. 4 th International Conference NLDB '99, Klagenfurt, Austria, 1999.
2. V. Ambriola, V. Gervasi, *On the parallel refinement of NL requirements and UML diagrams.* In Proc. of Workshop on Trasformations in UML, Genova, Italy, Apr. 2001.
3. V. Ambriola, V. Gervasi, *Synthesizing ASMs from natural language requirements*. In Proc. of the 8th EUROCAST Workshop on Abstract State Machines, Feb. 2001.
4. G. Cascini, A. Fantechi, E. Spinicci, *Natural Language Processing of Patents and Technical Documentation.* Proc. of DAS 2004 6th IAPR International Workshop on Document Analysis Systems, Firenze, Italy, September 2004, LNCS vol. 3163 (2004).
5. A. Durán, A. Ruiz Cortés, R. Corchuelo, M. Toro, *Supporting Requirements Verification Using XSLT*. Proc. of RE 2002, Essen, Germany, Sept. 2002.
6. S. Gnesi, G. Lami, G. Trentanni, *An automatic tool for the analysis of natural language requirements.* CSSE Journal vol. 20 (1), CRL Publishing: Leicester, Jan. 2005; 53-62.
7. K. Krippendorff, *Content Analysis : An Introduction to Its Methodology*, 2nd Edition. Sage Publications, Dec. 2003.
8. L. Mich, Ambiguity *identification and resolution in software development: a linguistic approach to improve the quality of systems*. In Proc. WESS'01, Firenze, Nov. 2001.
9. L. Mich, R. Garigliano, *Ambiguity Measures in Requirement Engineering*. Int. Conf. On Software Theory and Practice - ICS 2000, Beijing, China, Aug. 2000.
10. *Phrasys Natural Language Processing java library*, http://www.phrasys.com.
11. R. Siemens, *Practical Content Analysis Techniques for Text-Retrieval in Large, Untagged Text-bases*. ACM SIGDOC' 93 Conference, University of Waterloo, Oct. 1993.
12. *Link Grammar parser*, http://www.link.cs.cmu.edu.
13. D. D. K. Sleator, D. Temperley *Parsing English with a Link Grammar*. Third International Workshop on Parsing Technologies, Aug. 1993.
14. *TreeTagger*, http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger.
15. R. P. Weber, *Basic Content Analysis*. Sage Publications, Aug. 1990.
16. *WebFountain*, www.almaden.ibm.com/webfountain.
17. W. M. Wilson, L. H. Rosenberg, L. E. Hyatt, *Automated Analysis of Requirement Specifications*. ICSE'97, Boston, MA, May 1997.