Text analytics for discovering concerns in requirements documents

Alejandro Rago[1,3], Claudia Marcos[2,4], Andrés Diaz-Pace[1,3]

[1] Instituto Superior de Ingeniería de Software Tandil (ISISTAN), CONICET
Campus Universitario, Paraje Arroyo Seco, B7001BBO, Tandil, Bs. As., Argentina
Te: +54 (2293) 439682 Ext. 42 - Fax: +54 (2293) 439681

[2] Instituto de Sistemas Tandil (ISISTAN), UNICEN University
Campus Universitario, Paraje Arroyo Seco, Tandil, Bs. As., Argentina

[3] CONICET, National Council for Scientific and Technical Research
C1033AAJ, Bs. As., Argentina

[4] CIC, Committee for Scientific Research
B1900AYB, La Plata, Argentina

{arago,cmarcos,adiaz}@exa.unicen.edu.ar

Abstract. Recent trends in the software engineering community advocate for the improvement of textual requirements using (semi-)automated tools. In particular, the detection of incomplete or understated concerns at early development stages hold potential, due to the negative effects of untreated concerns on the development. Assistive tools can be of great help for analysts to get a quick picture of the requirements and narrow down the search for latent concerns. In this article, we present a tool called *REAssistant* that supports the process of discovering concerns in textual specifications. To do so, the tool relies on the UIMA framework and EMF-based technologies to provide an extensible architecture for concern-related analyses. Currently, the tool is configured to process textual use cases by using a number of textual analytics modules that identify lexical, syntactical and semantic entities in the specifications. We have conducted a preliminary evaluation of our tool in two case studies, obtaining promising results when comparing to manual inspections and to another tool.

Keywords: software requirement, tool-support, natural language processing, requirements analytics, crosscutting concern, use case specification

1 Introduction

Accomplishing a good understanding of stakeholders' requirements is a necessary condition for building systems correctly, and more importantly, for developing the correct system. In general, most software projects use textual specifications for documenting the concerns of a system [15]. The underlying reason for using

natural-language writing relies on its simplicity, which promotes a fluid communication between parties. While some of the concerns only address functionality, there are other concerns whose analysis is vital for reaching a successful project outcome (e.g., quality attributes, business goals, architectural-significant concerns). Unfortunately, requirements engineering (RE) activities often drive analysts to focus on functional concerns in order to figure out what the system should do, leaving little room for attending relevant concerns. Examples of such concerns are synchronization, performance, distribution, among others. Consequently, it is not uncommon to find several key concerns scattered among multiple documents or lightly treated on supplementary requirements documents.

To overcome this situation, developers need to go through several requirements documents skimming concerns and analyzing their effects over the system. Yet, this inspection is a difficult, time-consuming and error-prone activity, conditioned by the size of requirements specifications. In this context, tools able to support the detection of concerns in a (semi-)automated fashion are of great help for the analysts. In principle, assistive tools should interpret textual requirements and reason about their contents for extracting concern information.

In the last few years, several approaches and tools for discovering concerns have been developed [3,19,1]. The majority of them are ad-hoc solutions that combine Information Retrieval (IR) and Natural Language Processing (NLP) techniques. Typical requirements applications of IR and NLP techniques include the identification of word properties, recurrent concepts, entities, behaviors, among others. However, the poor semantic support of existing approaches prevents the uncovering of crosscutting effects, which are really important in the understanding of a concern. Moreover, existing tools do not always support the addition of processing modules (such as semantic-aware requirements tagging).

In this article, we use several technologies for discovering concerns and their effects in textual requirements documents. We have developed a tool called *RE-Assistant* (REquirements Analysis Assistant) that is composed of two main components. The first component implements an extensible infrastructure for processing requirements on top of the Unstructured Information Management Architecture[5] (UIMA) [9]. *REAssistant* uses UIMA to flexibly assemble and execute pipelines of NLP modules. The second component deal with the extraction of concerns from the annotations produced with UIMA. To do so, we opted for a querying language similar to SQL, built on top of the EMF-Query2[6] technology. This choice allows us to express simple but yet powerful concern queries.

The contributions of our work are twofold. First, our infrastructure builds on UIMA to provide a powerful NLP pipeline for processing textual requirements and therefore allows its extension with little effort. As a proof of concept, we wrapped several third-party NLP libraries into the UIMA pipeline and also developed some RE-specific modules for "annotating" use cases. Second, we have integrated a technology for harnessing the outputs of textual requirements processing modules and provide ways for querying them effectively. Using this

[5] UIMA project: http://uima.apache.org/

[6] EMF-Query2 project: http://www.eclipse.org/modeling/emf/?project=query2#query2

technology, we provide support for analysts to rapidly inspect the requirements documents using the knowledge acquired with the analytics pipeline.

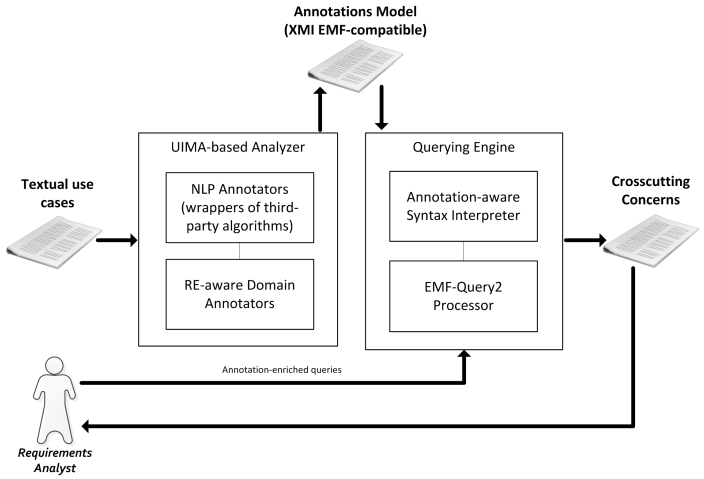
We have performed a preliminary evaluation of our infrastructure over two publicly-available case studies. These experiments consisted on expressing simple queries for uncovering crosscutting concerns. The results obtained so far are encouraging, when compared to the manual identification of concerns by humans or the assistance provided by another concern mining tool.

The rest of the paper is structured into 5 sections. Section 2 introduces the key ideas of our approach and describes the architecture behind our tool. Section 3 presents a comparative evaluation of our tool. Section 4 is devoted to related work. Finally, section 5 gives the conclusions and analyzes future lines of work.

2 *REAssistant*: a tool to analyze requirements

We have developed an infrastructure for processing textual requirements, which tries to overcome the limitations of existing textual analytics and to incorporate semantic knowledge of the requirements domain. Instead of building this infrastructure from scratch, we investigated and selected state-of-the-art technologies for tackling these problems. The *REAssistant* approach works as follows (see Figure 1). Initially, the analyst provides as input a set of textual use cases. These textual requirements are processed by the *UIMA-based Analyzer* component. In simple terms, this component executes a series of text analytics on the use cases to produce useful information from the sentences. This component relies on the UIMA framework to deploy and parametrize different analyses. UIMA serves

Fig. 1. *REAssistant* components



as the basis foundation for building analytic applications that process unstructured information such as requirement documents. Each text analytic in UIMA is wrapped by a so-called annotator. The output of an annotator are annotations, which identify and label specific regions of the text. As end result, the *UIMA-based Analyzer* generates an annotation-based representation of the use cases (see Figure 3). Since *REAssistant* works with individual sentences, each use case is decomposed into its behavior steps for processing.

Then, the requirements analyst is able to input and execute queries with the help of the annotations previously generated. Unfortunately, UIMA does not process the outcomes of its annotators. To cover for this limitation, *REAssistant* uses a querying language of EMF model. The *Querying Engine* component seamlessly connects the UIMA annotations and the EMF-Query2 technology. The queries are simple and expressive for searching concerns of interest for the analysts, and can be further refined as new annotators are incorporated to the *UIMA-based Analyzer*. A nice property of the *Querying Engine* is that it allows to search concerns in very different ways. For instance, an analyst can write down a query to look for verbs that occur very often in the requirement documents, or to look for relevant keywords that relate to particular concerns. In our approach, we take these ideas one step further, using domain knowledge about use cases to find crosscutting concerns.

Finally, the execution of a query in *REAssistant* will produce as output a subset of sentences (usually, use case steps) that are related somehow to the concern being searched. Furthermore, the parts of the sentences can later be highlighted to show that they are potentially crosscut by the selected concerns. It is worth mentioning that each query codifies knowledge about concerns, and how they relate semantically to natural language expressions. It is up to the analyst to write and refine queries for satisfactory results.

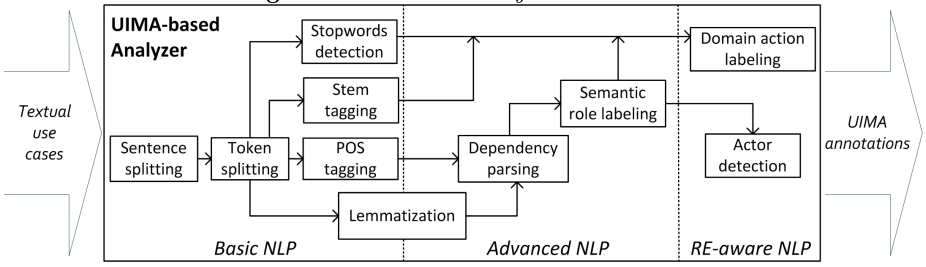
2.1 The requirements processing pipeline

A key aspect of our approach is its reliance on UIMA [9]. The *UIMA-based Analyzer* component makes an extensive usage of the annotation mechanisms provided by UIMA. An annotation identifies and labels (i.e., annotates) a specific region of a text document. For instance, an annotation can label a noun as "object" or a verb as an "action" in a sentence. A nice feature of annotations is that they can be arranged in layers on the same text. As explained above, the building blocks of an UIMA application are the so-called *annotators*. An annotator is a module that iterates over an artifact (e.g., a textual document) in order to discover new annotation types based on existing ones, and updates a shared representation structure. Some annotators can work as standalone modules while others may need the output (annotations) produced by other annotators. An end-to-end analysis requires the configuration of several annotators.

The *UIMA-based Analyzer* implementation comprises several annotators that are classified in two groups: *NLP annotators* that process language-independent text (e.g., the part-of-speech of a word), and *RE-aware annotators* that take into account the context of a requirements specification for the analyses (e.g.,

an actor of a use case step). Figure 2 depicts the arrangement of the annotators in our pipeline. The first group of annotators (leftmost and middle columns) is in charge of recognizing lexical, syntactical and domain-independent semantic properties of the text. Particularly, these annotators perform basic tasks such as detecting sentences and word boundaries, identifying word features (e.g., part-of-speech, stems), and also advanced tasks such as recognizing relationships (e.g., dependencies between words and predicates). The second group of annotators (rightmost column) is in charge of detecting concepts relevant to the requirements engineering domain. For example, *REAssistant* has annotators that identify the actors of a use case step, and the roles played by these actors in the intended behavior (i.e., a conceptual abstraction). Within UIMA, we wrapped all the NLP algorithms above as annotators. Furthermore, in case of alternative NLP implementations, the annotators can be easily exchanged (e.g., substitution of the OpenNLP POS tagger for its counterpart in Stanford CoreNLP).

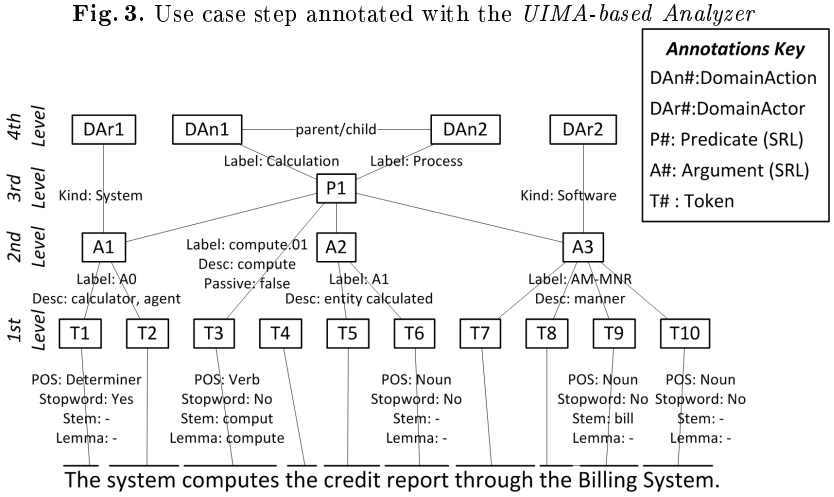
Fig. 2. *UIMA-based Analyzer* annotators



As a motivating example, let's consider the use case step "The system computes the credit report through the Billing System". Figure 3 shows the linguistic analysis carried out by the *UIMA-based Analyzer* on this particular sentence. Each rectangle denotes an annotation that points either to the base text or to other annotations. The annotations of the first level correspond to tokens (Basic NLP). The annotations of the second and third levels correspond to the Semantic Role Labeling (SRL) module, which reveals the predicate structure and the meanings of predicates (Advanced NLP). The annotations of the fourth level have a higher abstraction and encompass RE-specific concepts (RE-aware NLP).

NLP annotators The first group of annotators executes basic and advanced NLP tasks. These tasks are responsible for recognizing grammatical structures in individual sentences (actually, use case steps). This information is progressively extracted by means of several text analytics annotators. The end goal is the identification of *sentence predicates* and the semantic arguments filling the roles of those predicates [11]. We argue that predicates contribute to clarify the

Fig. 3. Use case step annotated with the *UIMA-based Analyzer*



meaning of sentences, hence, they enable the discovery of latent crosscutting relations in the textual use cases.

The Basic NLP tasks (see Figure 2) include: i) **sentence splitting**, for identifying sentence boundaries; ii) **token splitting**, for extracting token from sentences; iii) **stopwords detection**, for discarding irrelevant tokens; and iv) **stemming**, for reducing each token to its lexical root. In addition, **Part-of-Speech** (POS) **tagging** is used for identifying the linguistic category of each token (e.g., noun, verb, adjective, participle, pronoun, preposition, etc.). Implementations of these five techniques are already available. Well-known packages include: OpenNLP[7], Stanford CoreNLP[8], and Mate-Tools[9]. For example, OpenNLP provides algorithms for sentence splitting, token splitting and POS tagging. Stanford CoreNLP provides similar algorithms but also supports dependency parsing. In the example of Figure 3, annotations from T1 to T10 correspond to tokens. In more detail, T3 is a verb and its stem is "comput".

We believe that techniques such as POS tagging might not always provide enough information for identifying crosscutting concerns [16]. For instance, verbs such as "have" or "do" can be hints of a crosscutting behavior, but the contextual information of the verb is needed for a more precise analysis. For this reason, we enhanced the standard NLP pipeline with three tasks (see Figure 2), namely: **lemmatization**, **dependency parsing** and **semantic role labeling** (SRL) [5]. For the purposes of *REAssistant*, lemmatization and dependency parsing can be seen as prerequisites for applying SRL. One of the benefits of SRL is the recognition of the semantic arguments associated with a predicate (or verb) and the classification of these arguments into specific roles (e.g., the

[7] http://opennlp.sourceforge.net/projects.html
[8] http://nlp.stanford.edu/software/corenlp.shtml
[9] http://code.google.com/p/mate-tools/

agent, the patient, the manner, the time, the location, etc.). Existing packages, such as Mate-Tools, provide algorithms for lemmatization, dependency parsing, and SRL. The annotations of the second and third level (P1, A1, A2 and A3) of Figure 3 correspond to the SRL module and provide richer information, such as the predicate structure or and their meanings. In detail, A1 is an argument and represents the "calculator" in the phrase.

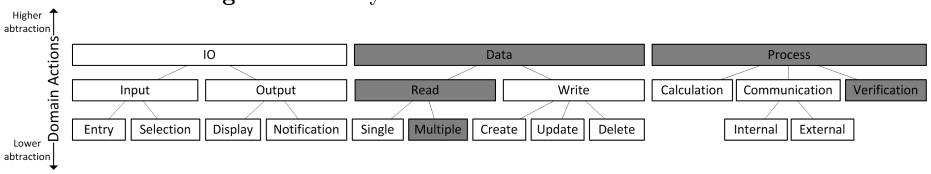
Requirements-aware annotators Although the SRL annotator does a good job exposing the predicate structure of the sentences, the interpretation of such predicates might not take into account intrinsic characteristics of requirements engineering activities. For instance, none of the *NLP annotators* can distinguish subtle differences between the participants of use case steps. Although predicate structures may uncover the participants, they are insensitive for determining whether a participant is a human actor or an external system. On the contrary, humans analysts show great interpretation capabilities when analyzing use cases. Also, human analysts understand phrases/words based on common expressions and lexicon related to the "story telling" of use case scenarios [7]. For instance, an analyst can interpret that phrases like "create a new record" and "store the new information" are semantically-equivalent as requirements for a given system. The *NLP annotators* offer yet little help for detecting these kind of similarities.

The issues above are actually a domain-specific characteristic of requirements that needs to be addressed after the NLP analyses. This is where the *RE-aware domain annotators* come into play, providing mechanisms to spot and label domain information. The first annotator takes as input the SRL annotations and detects actors and their properties, using arguments as the starting point. The second annotator uses the notion of domain actions as a mechanism to adapt (and somehow simplify) the "general" predicates from SRL to the use cases terminology. Specifically, we use a dictionary of actions for use cases, which is refinement of the work of Sinha et al. [20]. This dictionary contains a hierarchy of domain actions, in which the upper levels are action categories and the lower levels cover abstract actions. Coming back to the example of Figure 3, the annotations of the fourth level encompass requirements-specific concepts. In more detail, DAn1 and DAn2 are domain actions indicating that P1 is actually a Process/Calculation behavior in the context of use cases, and DAr2 indicates that A3 holds a software actor.

Figure 4 shows the hierarchy of domain actions considered in our approach. The annotator computes domain-action annotations out of predicate annotations. The association of a given predicate to a set of domain actions is performed using a multi-class/label classifier (MLC) [21]. Basically, MLC is a machine learning technique that takes an instance and associates it with one or more disjoint labels representing conceptual classes. In our case, the classifier annotates each predicate with a set of likely domain actions. The predicates are previously filtered by the stemming and stopwords tasks to reduce "noise" in the text. The algorithms of the Mulan toolkit [10] were used for learning the classifier.

[10] http://mulan.sourceforge.net/

Fig. 4. Hierarchy of Domain Actions for use cases



Algorithm 1 Simple NLP-based query

```
select S from
    [#Sentence#] as S,
    [#Token#] as T
where for T(
    stem = 'commun' or lemma = 'interaction' or lemma = 'internet' or
    lemma = 'external' or lemma = 'separate' or lemma = 'online' or
    lemma = 'server' or lemma = 'offline' or stem = 'connect'
)
where T.begin > S.begin
where T.end < S.end
```

2.2 The querying engine

After processing the textual requirements and generating the corresponding annotations, we still have to analyze and extract crosscutting concerns from the data. We advocate for a solution in which the analysts can make queries in the annotation space, in a similar way SQL is used for retrieving data from databases. The *Querying Engine* component provides searching facilities through a querying language for UIMA annotations. This component relies on the EMF-Query2 technology for several reasons. Since UIMA annotations models are stored in a EMF-compatible format, the EMF-Query2 development fits well in our work. EMF-Query2 can help analysts to customize and refine queries on the fly without much effort. An interesting feature provided by the *Querying Engine* is that requires no implementation changes as new textual annotators are added to the *UIMA-based Analyzer*. Furthermore, queries are dynamically linked to the UIMA annotation typesystems and can reference new types on demand.

Analysts using *REAssistant* can express search patterns that, when applied to the UIMA annotations, can reveal the presence of a particular concern in the requirements documents. A simple query might look for occurrences of concern-specific keywords. In the case of a Distribution concern, for instance, a query can look for tokens such as 'commun', 'server', 'external', or 'internet', among others. Algorithm 1 shows this query. However, an analyst can improve the detection of the Distribution concern by leveraging on the RE-aware annotations. Let's consider the example use case step of Figure 3. In that sentence, there is not only a computation, but also an interaction with a secondary actor. So, with the objective of uncovering a hidden Distribution concern, a second query could be written to retrieve sentences with domain actions "Communication",

Algorithm 2 Advanced RE-aware query

```
select S from
    [#Sentence#] as S,
    [#DomainActor#] as DAr
    [#DomainAction#] as DAn
where for DAr(label = 'Process')
where for DAn(kind = 'Software')
where DAr.begin > S.begin where DAr.end < S.end
where DAn.begin > S.begin where DAn.end < S.end
```

"Verification" and "Calculation", grouped by the class "Process". These actions should be linked to the Distribution concern if a "Software" actor is present. Note that this linkage comes from a semantic interpretation of the use cases, rather than from a syntactic text analysis. That is, by making the "Process" class and the "Software" actor explicit, Distribution arises as a candidate concern.

3 Preliminary evaluation

In order to assess the performance of our approach, we conducted a series of experiments with two well-known case studies. The first case study is the Course Registration System (CRS) [4], a distributed system to be used for managing an university courses and subscriptions. In this system, students can enroll in existing courses, professors can create new courses and report student's grades. The CRS documentation consists of 8 use cases (about 20 textual pages). The second case study is the Health Watcher System (HWS) [12], a web-based system that serves as a mediator between citizens and the municipal government. Citizens can register complaints about the healthcare service, read health-related news and make queries regarding health issues. The HWS requirements specification consists of 9 use cases (about 19 textual pages) and other researchers have used as testbed for analyzing its crosscutting concerns and quality-attribute properties [18,22].

The experiments involved both manual and tool-supported processing of the case studies. For comparison purposes, we initially established a "golden" standard (i.e., the ideal solution) for the CRS and HWS. Each golden standard contained the concerns and crosscutting concerns that are believed to be correct, according to the information available from the case studies. Three kind of experiments were conducted. The first type of experiments comprised the inspection of the requirements documentation by real people. In CRS, we had with two groups of analysts (referred to as Groups A and B) that separately inspected and manually labeled crosscutting concerns and analyzed their effects. In HWS, we also had with two groups (referred to as Groups C and D) that conducted the same analysis. The four groups were allotted 2 hours to complete the analysis. Group A and B were composed by two senior undergraduate students, while C and D were composed of two PhD. students each. The second type of experiment was performed only on the HWS case study, and consisted of analyzing

the textual specification using EAMiner [19]. EAMiner is a third-party tool for mining concerns. At last, we wrote a small number of queries (similarly to the examples in Algorithms 1 and 2) for the annotations. Then, we ran those queries on the two case studies with help of the *Querying Engine*.

As evaluation criteria, we used standard Information Retrieval (IR) [2] measures such as Precision and Recall. Unlike similar evaluations [18,1], we used a fine-grained approximation to compute the Precision and Recall measurements. The evaluation represents the occurrence of a crosscutting concern in the text at the level of sentences. In addition, to solve some discrepancies found in the names of semantically-equivalent concerns produced by different analysts/tools, we defined two matching tables for CRS and HWS. Thus, the computation compares the sentences tagged in the golden standard against the sentences tagged in each experiment. A use case step is considered as a true positive if the golden standard contains a crosscutting effect for the step, and the group/tool actually detected that effect. A similar reasoning is used for true negatives, false positives and false negatives. The interpretations of the measures in this context are the following: precision measures how many of the crosscutting effects detected by the group/tool are correct; while recall measures how complete the concern detection is, in the sense that some crosscutting effects might be missed. For an assistive tool, an acceptable precision and a high recall are desirable. That is, we would like *REAssistant* to find as many crosscutting effects as possible, even at the cost of detecting a few incorrect ones, because the analyst will ultimately assess the outputs produced by the tool.

The golden standards comprise 8 crosscutting concerns with 143 crosscutting effects (use case steps) and 6 crosscutting concerns with 185 crosscutting effects for the CRS and HWS, respectively. With *REAssistant*, we used queries for finding the following concerns: *Integrity Management*, *Error Management*, *User Access Control*, *Presentation*, *Distribution*, *Persistence* and *Concurrence*. Table 1 summarizes the measurements obtained during the experiments. Overall, the

Table 1. Experimentation Results

		Precision	*Recall*	*F-Measure*
CRS	Human Group A	0,702	0,230	0,346
	Human Group B	**0,772**	0,391	0,519
	REAssistant	0,579	**0,643**	**0,609**
HWS	Human Group C	0,761	0,584	0,661
	Human Group D	**0,962**	0,411	0,576
	EAMiner	0,654	0,302	0,413
	REAssistant	0,789	**0,827**	**0,807**

four human groups (A to D) achieved a very good precision (~75% and ~85% in CRS and HWS, respectively) but a rather low recall (~30% and ~50, respectively). The recall might depend on the expertise of the analysts and also on

the time spent on the concern analysis. Moreover, there is a clear difference in recall between the groups in CRS (A and B) and those in HWS. We believe that more experienced analysts are able to discover more concerns than novice analysts, as it was the case of groups A and B versus groups C and D. Regarding tools, they achieved a comparable precision (between 60% for CRS and 80% for HWS) although lower than the precision of the human groups due to the influence of false positives. Nonetheless, an important advantage of the tools was the speed-up to discover candidate concerns. The four groups needed between 90-120 minutes to perform the analysis, while on the contrary, both EAMiner and *REAssistant* took only 10-30 seconds. Regarding the tools' recall, *REAssistant* outperformed human analysts in both case studies (~65% vs. ~40% in CRS and ~85% vs. ~60% in HWS). We believe that humans have trouble to find all the relevant concerns, and in this scenario tools such as EAMiner and *REAssistant* can become really helpful. Moreover, *REAssistant* also beat EAMiner's recall by a considerable margin (~85% vs. ~30%). We argue that this difference is due to the better support for advanced NLP and RE-aware analysis provided by our approach. The F-measure values obtained from the evaluation consistently showed the improvements of our approach.

4 Related work

In the last few years, several approaches have been developed to enhance the analysis of requirements documents, and particularly, those written in natural language. A study carried out by Mich et al. [15] reported that approximately 80% of the requirements are specified in natural language, with little or no structure. Many researchers have advocated for the use of Information Retrieval (IR) [10] and Natural Language Processing (NLP) [14] techniques to automate the detection of patterns in the text and extract useful information for human analysts. In the context of this work, we are interested on those approaches that mine relevant concerns from requirements in a semi-automated fashion.

A large number of existing approaches belong to the early aspects research community. Rosenhainer [17] applied Information Retrieval techniques to automate the search of early aspects in software specifications. He performs individual queries on requirements documents, based on regular expressions and string matching techniques. Baniassad and Clarke developed the Theme approach [3] as a tool support for aspect orientation at the requirements and design levels. Specifically, Theme/Doc builds views of the requirements specifications, in order to expose the relationships between system behaviors and reveal crosscutting functionality. The tool requires two inputs: key actions and key entities, which are usually provided by an analyst. Later on, Busyairah and Zarinah [1] developed the 3CI approach based on Theme/Doc. 3CI further automates the identification of crosscutting requirements by avoiding the manual input of key entities and actions. This tool relies on various NLP techniques to determine the dependencies among requirements, mostly derived from the analysis of dominant verbs. The EAMiner tool [19] is another approach for aspect mining that

performs POS and semantic tagging of textual requirements. The semantic tagger categorizes each word according to a semantic English-grammar taxonomy. Each semantic tag is associated to non-functional requirements gathered from the NFR framework [6]. Each word tagged with a semantic category related to an NFR is proposed as a non-functional early aspect.

REAssistant has similarities with the approaches above but also has distinctive features. A shared characteristic is the application of NLP techniques for analyzing requirements. Yet, the approaches just discussed have not employed advanced NLP tasks, such as dependency parsing or semantic role labeling. The queries of *REAssistant* can be seen as an evolution of Rosenhainer's queries. The main difference here is that our approach uses a richer query language, and we can benefit from the semantic information provided by annotations.

In parallel with the early-aspects work, other authors have investigated different ways to derive abstract requirements models from text, which can later facilitate tasks such as consistency checking, or abstract concept extraction. Drazan & Menzl [8] analyzed the diverse ways of writing use cases, and developed NLP-based patterns to extract information from use cases. Each pattern represents grammar structures that are frequent in use cases steps, and the patterns can be adjusted to the complexity of the sentences. Kamalrudin and Grundy [13] have recently presented a tool approach that checks requirements for inconsistency. This work is based on the notion of "essential use cases" and interactions, in which elements of natural language requirements are linked to their corresponding abstractions. The tool can determine if the use case model is complete, consistent and correct, by matching the extracted essential use case model against a library of acceptable essential use case patterns. In our approach, the annotations-based representation can be seen as a lightweight model of use cases, but it does not prescribe the types of entities and relationships that a use case structure should have. Sinha et al. [20] developed a pipeline of UIMA annotators for extracting characteristics of the behavioral model from textual use cases. The design is close to our UIMA solution, although it differs in the technical details of the analysis modules and in the model generated by the pipeline. Sinha's pipeline generates a behavioral model that is intended to support completeness, structural and flow checks. To this end, the concepts of semantic actions and context information are introduced. As we discussed in section 2.1, these concepts are the basis for our hierarchy of domain actions. Furthermore, we believe that domain actions work at a higher abstraction level than the essential use case interactions of Kamalrudin's approach.

5 Conclusions

In this article, we presented a novel tool called *REAssistant* that brings support for processing textual requirements documents and for extracting useful information from them. In particular, our tool offers several modules to discover latent crosscutting concerns from use case specifications. A number of advanced NLP and RE-aware modules are used for analyzing use case steps and annotate them.

These annotations enable the search for crosscutting concerns through queries that operate at the level of annotations. The technical contributions of our work are the assembly of diverse text analytics modules into a single (automated) analysis pipeline and the integration of a querying language to search through their outputs.

We have carried out a preliminary evaluation of the *REAssistant* tool on two case studies, obtaining encouraging results. Our approach showed a very good recall, meaning that it was able to find most of the concerns hidden and scattered in the case studies. The precision was not as good as the ones achieved by human analysts. Yet, it was still within an acceptable range when compared to the precision of EAMiner. It should be noticed that our tool is not intended to replace the analyst but rather to assist her with a quick picture of candidate crosscutting concerns. Furthermore, the ability of *REAssistant* to find latent crosscutting concerns at the sentence level was very useful for the analysts. The evaluation of the case-studies also revealed some limitations of the approach. First, the quality of the text given as input to the tool impacts on the results of the NLP pipeline. Second, despite the benefits of having concern queries expressed in semantic terms (i.e., annotations), the tool outputs can be affected by the way these queries are written. Third, the classifier of domain actions might have a bias towards certain actions, which in turn favor the detection of certain concerns over others. Also *REAssistant* does not take into account the importance (i.e., priority) of the mined concerns for the system.

As future work, we plan to evaluate *REAssistant* with more case-studies. Having additional case-studies will serve us to enrich the current queries and maybe to arrange them in rules focused on specific concerns. In particular, we are investigating how to apply NLP techniques for spotting quality-attribute concerns, business goals and architecturally-significant requirements. We speculate that these new "types of concerns" can be incorporated with few tool modifications, thanks to the UIMA framework. If the concern identification is performed on multiple text documents, we are interested in exploring the traceability links between the concerns extracted from each document.

Acknowledgments. We would like to thank to the attendees of the "Aspect-Oriented Software Development" course (given at UNICEN University), who helped with the analysis of case studies.

References

1. Ali, B., Kasirun, Z.: 3ci: A tool for crosscutting concern identification. In: Computational Intelligence for Modelling Control & Automation, 2008 International Conference on. pp. 351–355. IEEE (2008)
2. Baeza-Yates, R., Ribeiro-Neto, B., et al.: Modern information retrieval, vol. 463. ACM press New York. (1999)
3. Baniassad, E., Clements, P., Araujo, J., Moreira, A., Rashid, A., Tekinerdogan, B.: Discovering early aspects. Software, IEEE 23(1), 61–70 (2006)

4. Bell, R.: Course registration system. http://sce.uhcl.edu/helm/RUP_course_example/courseregistrationproject/indexcourse.htm (2011)
5. Bjorkelund, A., Hafdell, L., Nugues, P.: Multilingual semantic role labeling. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task. pp. 43–48. CoNLL '09, Association for Computational Linguistics, Stroudsburg, PA, USA (2009), http://dl.acm.org/citation.cfm?id=1596409.1596416
6. Chung, L., do Prado Leite, J.: On non-functional requirements in software engineering. Conceptual modeling: Foundations and applications pp. 363–379 (2009)
7. Cockburn, A.: Writing effective use cases, vol. 1. Addison-Wesley Reading, MA (2001)
8. Drazan, J., Mencl, V.: Improved processing of textual use cases: Deriving behavior specifications. SOFSEM 2007: Theory and Practice of Computer Science pp. 856–868 (2007)
9. Ferrucci, D., Lally, A.: Uima: an architectural approach to unstructured information processing in the corporate research environment. Natural Language Engineering 10(3-4), 327–348 (2004)
10. Frakes, W., Baeza-Yates, R.: Information retrieval: data structures and algorithms. Prentice Hall PTR (1992)
11. Gildea, D., Jurafsky, D.: Automatic labeling of semantic roles. Computational Linguistics 28(3), 245–288 (2002)
12. Greenwood, P.: Tao: A testbed for aspect oriented software development. http://www.comp.lancs.ac.uk/~greenwop/tao/ (2011)
13. Kamalrudin, M., Hosking, J., Grundy, J.: Improving requirements quality using essential use case interaction patterns. In: Software Engineering (ICSE), 2011 33rd International Conference on. pp. 531 –540 (may 2011)
14. Manning, C., Schutze, H., MITCogNet: Foundations of statistical natural language processing, vol. 59. MIT Press (1999)
15. Mich, L., Franch, M., Novi Inverardi, P.: Market research for requirements analysis using linguistic tools. Requirements Engineering 9, 151–151 (2004), http://dx.doi.org/10.1007/s00766-004-0195-3, 10.1007/s00766-004-0195-3
16. Rago, A., Abait, E., Marcos, C., Diaz-Pace, A.: Early aspect identification from use cases using nlp and wsd techniques. In: Proceedings of the 15th workshop on Early aspects. pp. 19–24. ACM (2009)
17. Rosenhainer, L.: Identifying crosscutting concerns in requirements specifications. In: Proceedings of OOPSLA Early Aspects. Citeseer (2004)
18. Sampaio, A., Greenwood, P., Garcia, A., Rashid, A.: A comparative study of aspect-oriented requirements engineering approaches (2007)
19. Sampaio, A., Rashid, A., Chitchyan, R., Rayson, P.: Ea-miner: towards automation in aspect-oriented requirements engineering. Transactions on aspect-oriented software development III pp. 4–39 (2007)
20. Sinha, A., Paradkar, A., Kumanan, P., Boguraev, B.: An analysis engine for dependable elicitation of natural language use case description and its application to industrial use cases. IBM Research Report RC24712 (2008)
21. Tsoumakas, G., et al.: Multi label classification: An overview. International Journal of Data Warehousing and Mining 3(3), 1–13 (2007)
22. Zhang, H., Ben, K.: Architectural design of the health watch system with an integrated aspect-oriented modeling approach. In: Computer Design and Applications (ICCDA), 2010 International Conference on. vol. 1, pp. V1–624 –V1–628 (2010)