REAssistant: a Tool for Identifying Crosscutting Concerns in Textual Requirements

Alejandro Rago*†1, Claudia Marcos*‡2, J. Andrés Diaz-Pace*†3
*Instituto Superior de Ingeniería de Software (ISISTAN-UNICEN)
Tandil, Buenos Aires, Argentina
†CONICET, Argentina
‡CIC, Buenos Aires, Argentina
1 2 3{arago,cmarcos,adiaz}@exa.unicen.edu.ar

***Abstract*—Use case modeling is very useful to capture requirements and communicate with the stakeholders. Use cases normally have textual specifications that describe the interactions between the system and external actors. However, since use cases are specified from a functional perspective, concerns that do not fit well this decomposition criterion are kept away from the analysts' eye and might end up intermingled in multiple use cases. These *crosscutting concerns* (CCCs) are generally relevant for analysis, design and implementation activities and should be dealt with from early stages. Unfortunately, identifying such concerns by hand is a cumbersome and error-prone task, mainly because it requires a semantic interpretation of textual requirements. To ease the analysis of CCCs, we have developed an automated tool called *REAssistant* that is able to extract semantic information from textual use cases and reveal candidate CCCs, helping analysts to reason about them before making important commitments in the development. Our tool performs a series of advanced NLP analyses based on the UIMA framework. Analysts can define concern-specific queries in the tool to search for CCCs in the requirements via a flexible SQL-like language. In this article, we briefly discuss the technologies behind the tool and explain how an end user can interact with *REAssistant* to analyze CCCs in use case specifications. A short video explaining the main features of the tool can be found at https://youtu.be/i3kSJil_2eg. The REAssistant tool can be downloaded from https://code.google.com/p/reassistant.**

I. Introduction

Most software systems have certain concerns that are key for the success of a project [1]. These concerns are often related to business goals of the system (e.g., profit, market opportunities or brand positioning, etc.) and quality attributes (e.g., performance, fault tolerance or security, etc.) [2]. Since many requirements modeling techniques (e.g., use cases) are based on a functional decomposition criterion, some concerns are likely to be hidden in textual specifications, tangled with functionality and scattered across documents. These concerns are referred to as *crosscutting concerns* (CCCs) [1]. For example, an access control policy (part of a security quality attribute) can subtlety appear in multiple use cases, and might be overlooked by analysts and architects during the system design. Since requirements are commonly documented in natural language, analysts and developers must peruse textual specifications to reveal CCCs of interest for further analysis. Still, searching for latent concerns in requirements is a difficult and time-consuming task, mainly due to the semantics and ambiguities of natural language.

In this context, is useful for analysts to rely on tool support for processing requirements and identifying CCCs. Such a tool should be able to quickly gather a list of candidate CCCs from the text and present it to the analysts (e.g., integrity, synchronization, access control, etc.). Then, it is up to the analysts to inspect the list to determine which CCCs are actually relevant. There are several concern mining tools available that use *Natural Language Processing* (NLP) techniques and domain-specific dictionaries (e.g., taxonomies of quality attributes) [3]–[5]. Unfortunately, these tools have trouble to identify portions of functionality implicitly (or indirectly) affected by CCCs because they have poor semantic capabilities when processing textual requirements.

To overcome these limitations, we have developed the *REAssistant* (REquirements Analysis Assistant) tool [6]. Our tool supports the search of latent CCCs by relying on advanced NLP modules and domain knowledge about use cases. *REAssistant* uses an annotation-based representation of use cases that holds lexical, syntactical, semantical and domain information of the text. Furthermore, our tool is equipped with a NLP pipeline assembled with the UIMA framework that decorates the use cases with annotations [7]. The pipeline performs several linguistic analyses in the text, such as: dependency parsing, semantic role labeling and domain actions classification [6]. To find candidate concerns, the tool provides customizable concern-specific rules that can query the annotations generated earlier to extract not only CCCs but also their crosscutting relations (i.e., requirements affected by CCCs). The rules take advantage of the so-called "domain actions", which are a taxonomy of domain-neutral classes applicable to use cases. Finally, our tool is implemented as a set of Eclipse plugins that provide mechanisms for the analysis of concerns, including special views for visualizing CCCs at different levels of granularity.

The rest of this article is organized in 3 sections. Section II explains the concern discovery problem with a motivating example. Section III briefly discusses the architectural design of the tool and its components. Finally, Section IV provides a quick tour of the working of *REAssistant* from the viewpoint of an analyst.

Figure 1. Concern impacts in use cases

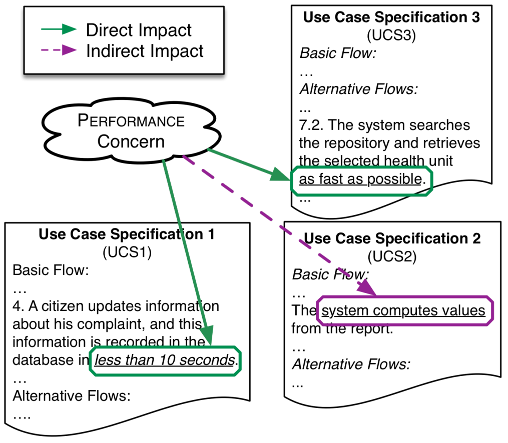
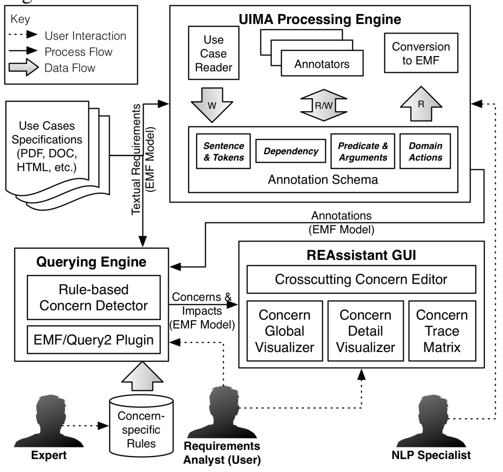

Direct Impact
Indirect Impact
PERFORMANCE Concern
Use Case Specification 3 (UCS3)
Basic Flow:
...
Alternative Flows:
...
7.2. The system searches the repository and retrieves the selected health unit as fast as possible.
...
Use Case Specification 1 (UCS1)
Basic Flow:
...
4. A citizen updates information about his complaint, and this information is recorded in the database in less than 10 seconds.
...
Alternative Flows:
....
Use Case Specification 2 (UCS2)
Basic Flow:
...
The system computes values from the report.
...
Alternative Flows:
...

Figure 2. Overview of the Architecture of *REAssistant*



II. Revealing Latent Concerns in Use Cases

Identifying CCCs in use case specifications demands a careful manual inspection by analysts, as well as a dosage of domain experience. There are three main activities that analysts should do: i) finding candidate concern(s), ii) determining the "real" concerns and specifying them, and iii) identifying the points of the specification (e.g., use case steps) affected by each concern, i.e., finding its crosscutting relations or impacts. Let us consider the excerpts from three use cases shown in Figure 1. The sentences of UCS1 and UCS3 qualify functionality with phrases such as "in less than 10 seconds" and "as fast as possible", which are hints of a PERFORMANCE concern. Thus, an analyst can interpret that there is a PERFORMANCE CCC at play, and search for performance-related words to quickly expose the concern. We call these explicit references in the text *direct impacts* of the concern. Several tools currently support keyword-based searches to mine them [3]–[5].

However, there might be other use cases implicitly affected by the same concern as well. For instance, an experienced analyst could determine that one of the steps of UCS2, referring to a "computation", is also constrained by the PERFORMANCE CCC. This relation can be discovered after making a semantic analysis of the text, rather than a lexical or syntactical analysis. We call these implicit relations in the text *indirect impacts* of the concern. Indirect impacts are usually harder to detect because they require an interpretation of the semantics in textual requirements. From an automation viewpoint, indirect impacts can be (approximately) detected by uncovering associations between specific concerns and "abstract" actions (e.g., compute, calculate, perform, execute) expressed in use cases, because such associations often hold the key for recognizing concern impacts. However, it is the analysts' responsibility to determine if a sentence is truly affected by a CCC. The role of tool support is then to recommend potential CCCs and let the analyst make the final decisions. Unfortunately, existing tools for mining concerns have problems to detect indirect impacts, because their semantic-level features are limited.

III. Architecture of *REAssistant*

REAssistant is built on the Eclipse IDE as a set of plugins that support both the linguistic analysis of textual use cases and the execution of search rules for identifying concerns [6]. Figure 2 shows the main components of the architecture, namely: *UIMAProcessingEngine*, *QueryingEngine*, and *REAssistant-GUI*. The communication among these components take place through files that contain (serialized) EMF[1] models. Use cases are first imported into the *UIMAProcessingEngine* via the *UseCaseReader* component, which gathers text from varied sources, such as PDF, DOC, HTML files, or directly from the use case editor bundled in *REAssistant*. Then, the text is stored in the *AnnotationSchema*, which is a shared data structure that allows the communication of text analytic modules. Once imported, a pipeline of *Annotators* takes the text from use cases and breaks them into individual sentences (e.g., behavior steps), and automatically generates different annotations for these sentences. There are two kind of annotators. The first set of annotators run a series of NLP tasks that include standard linguistic analyses (e.g., stemming, POS tagging). The second set of annotators runs more complex analyses (e.g., semantic role labeling) for extracting the predicate structure of the sentences and for mapping these predicates to domain actions. The computation of domain actions is performed with a special classifier reported in [6]. Furthermore, a NLP specialist can configure annotators via UIMA before running the pipeline. The resulting annotations are later exported to an EMF model. The *QueryingEngine* is equipped with the concern-specific searching rules that were defined beforehand by experts. By analyzing the use cases and their annotations, the *QueryingEngine* executes the searching rules on the text of use cases. At last, the *REAssistantGUI* features different views for browsing candidate CCCs and their impacts. This component provides edition and visualization support for the analyst to explore and refine the concerns found.

[1]http://www.eclipse.org/modeling/emf/

Figure 3. Annotations in a use case sentence

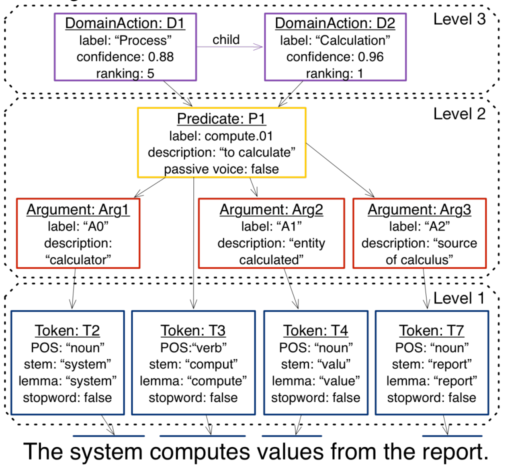


Figure 4. Rule syntax for searching CCCs

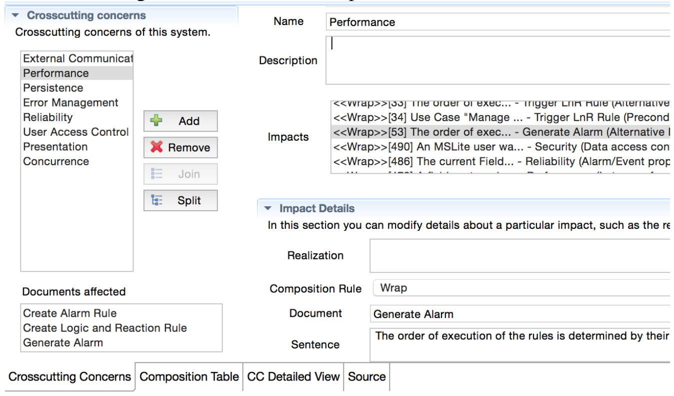
```
Rule for Performance Concern

Direct Rule (Query #1)
select S from [#Sentence#] as S, [#Token#] as T
where for T(lemma='response' or lemma='second' or
lemma='time' or stem='delay' or
lemma='throughput' or lemma='latency' or
lemma='deadline')
where T.begin >= S.begin where T.end <= S.end

Indirect Rule (Query #2)
select S from [#Sentence#] as S,
[#DomainAction#] as DA
where for DA(label='Calculation')
where DA.begin >= S.begin where DA.end <= S.end

Indirect Rule (Query #3)
select S from [#Sentence#] as S,
[#DomainAction#] as DA, [#Token#] as T
where for DA(label='Process')
where for T(lemma='result' or lemma ='value')
where T.begin >= S.begin where T.end <= S.end
where DA.begin >= S.begin where DA.end <= S.end
```

Figure 5. Concern editor provided in *REAssistant*



Crosscutting concerns of this system.

External Communicat
Performance
Persistence
Error Management
Reliability
User Access Control
Presentation
Concurrence

Name: Performance

Description:

Impacts:
<<Wrap>>[33] The order of exec... - Trigger LnR Rule (Alternative
<<Wrap>>[34] Use Case "Manage ... - Trigger LnR Rule (Precond
<<Wrap>>[53] The order of exec... - Generate Alarm (Alternative I
<<Wrap>>[490] An MSLite user wa... - Security (Data access con
<<Wrap>>[486] The current Field... - Reliability (Alarm/Event prop

In this section you can modify details about a particular impact, such as the re

Realization:

Composition Rule: Wrap

Document: Generate Alarm

Sentence: The order of execution of the rules is determined by their

Documents affected

Create Alarm Rule
Create Logic and Reaction Rule
Generate Alarm

REAssistant leverages on the UIMA framework[2] [6], [8]. UIMA is an extensible architecture for building analytic applications that process unstructured information. The architecture of our tool makes extensive use of the annotation mechanisms provided by UIMA. An annotation identifies and labels a specific region of a text document. Figure 3 shows a linguistic analysis of a use case step from a requirements specification. The annotations of level 1 correspond to tokens. Direct impacts would be typically discovered by analyzing information at this level. The annotations of levels 2 and 3 provide richer information, such as the predicate structure and domain actions, respectively. Indirect impacts can be discovered by querying information at level 3.

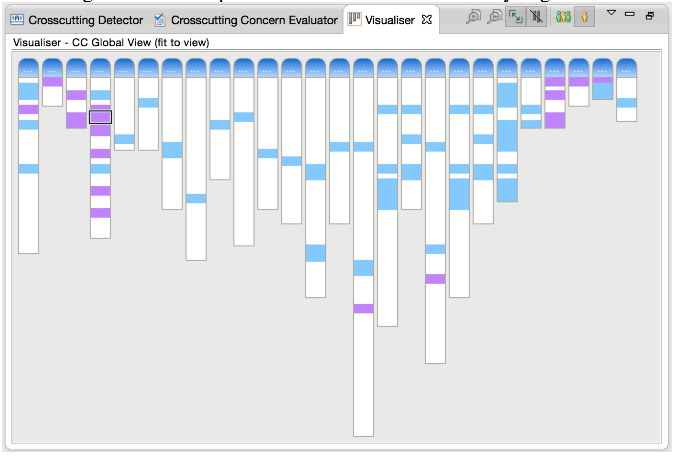
The *QueryingEngine* is implemented on top of the EMF Query2[3] project, which serves as an SQL-like language for searching through EMF models. The rule syntax is simple to understand and powerful enough to express concern-related queries. In addition, we have developed an abstraction layer that allows analysts to seamlessly incorporate UIMA-generated annotations in the queries. There are two types of rules: i) *direct rules*, responsible for detecting a CCC; and ii) *indirect rules*, for detecting domain actions that are potentially related to that concern. Direct rules are focused in finding explicit references to a particular CCC, for example, the word "server" or "database". Complementary, indirect rules are focused in finding more subtle associations that come from a semantic interpretation of the use cases. Figure 4 illustrates a PERFORMANCE rule composed of three queries. Query #1 would find parts of the text related to PERFORMANCE through the analysis of token lemmas such as "response" and "second", similarly to keyword-based approaches. Queries #2 and #3 make use of domain actions to reveal indirect impacts, looking for actions such as "calculation" and "process". For more information about the architecture of the tool, the NLP pipeline and the concern ruleset, the reader is referred to [6]. In this publication, we also report on the results of an empirical evaluation of *REAssistant* with three case-studies.

[2]http://uima.apache.org/
[3]http://www.eclipse.org/modeling/emf/downloads/?project=query2

IV. *REAssistant* IN ACTION

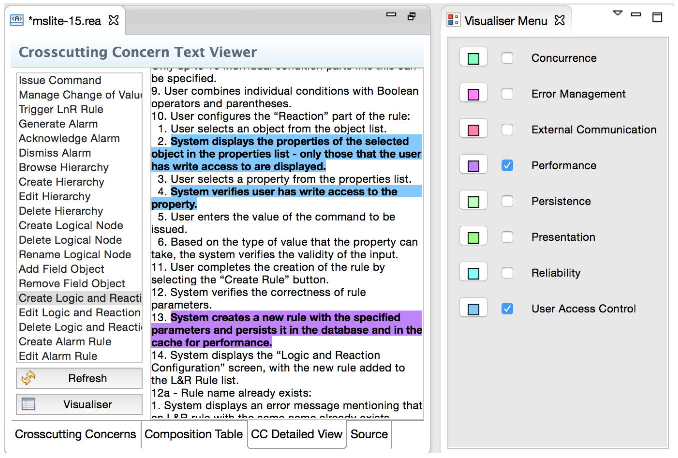
The *REAssistant* tool offers analysts functionality for editing use cases, performing a linguistic analysis of the use cases, and applying searching rules for identifying CCCs. In this section, we discuss the operation of the tool from the perspective of an analyst who is using it and explain how she/he interacts with the tool in the concern identification and analysis process (see a video at https://youtu.be/i3kSJil_2eg). Initially, the analyst needs to provide the text of use case specifications. Our tool has a form-based editor that handles the documentation of use cases and stores them in a persistent file with extension "ucs". The internal structure of "ucs" files are based on a standard use case template that contains sections to describe actors, main flow, alternative flows, supplementary requirements, etc. Once the "ucs" file is complete, analysts can automatically run a series of NLP analyses on the use cases. From the user's viewpoint, the linguistic analyses will produce all kinds of meta-information for the use cases in the form of layers of *annotations*, which are later stored in a persistent file with extension "uima". This file holds the results of the semantic analysis of the text.

Figure 6. Views provided in *REAssistant* for analyzing CCCs



Crosscutting Detector
Crosscutting Concern Evaluator
Visualiser
Visualiser - CC Global View (fit to view)

(a) Global view



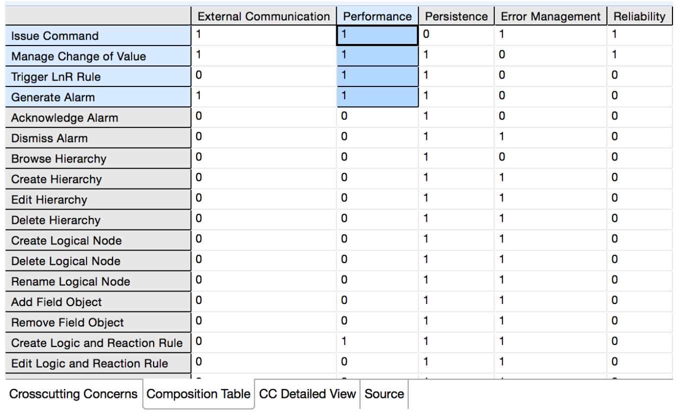
Crosscutting Concern Text Viewer

Issue Command
Manage Change of Valu
Trigger LnR Rule
Generate Alarm
Acknowledge Alarm
Dismiss Alarm
Browse Hierarchy
Create Hierarchy
Edit Hierarchy
Delete Hierarchy
Create Logical Node
Delete Logical Node
Rename Logical Node
Add Field Object
Remove Field Object
Create Logic and Reacti
Edit Logic and Reaction
Delete Logic and Reacti
Create Alarm Rule
Edit Alarm Rule

be specified.
9. User combines individual conditions with Boolean operators and parentheses.
10. User configures the "Reaction" part of the rule:
1. User selects an object from the object list.
2. **System displays the properties of the selected object in the properties list - only those that the user has write access to are displayed.**
3. User selects a property from the properties list.
4. **System verifies user has write access to the property.**
5. User enters the value of the command to be issued.
6. Based on the type of value that the property can take, the system verifies the validity of the input.
11. User completes the creation of the rule by selecting the "Create Rule" button.
12. System verifies the correctness of rule parameters.
13. **System creates a new rule with the specified parameters and persists it in the database and in the cache for performance.**
14. System displays the "Logic and Reaction Configuration" screen, with the new rule added to the L&R Rule list.
12a - Rule name already exists:
1. System displays an error message mentioning that

🟩	☐	Concurrence
🟪	☐	Error Management
🟥	☐	External Communication
🟪	✓	Performance
🟩	☐	Persistence
🟩	☐	Presentation
🟦	☐	Reliability
🟦	✓	User Access Control

(b) Detailed view



	External Communication	Performance	Persistence	Error Management	Reliability
Issue Command	1	1	0	1	1
Manage Change of Value	1	1	1	0	1
Trigger LnR Rule	0	1	1	0	0
Generate Alarm	1	1	1	0	0
Acknowledge Alarm	0	0	1	0	0
Dismiss Alarm	0	0	1	1	0
Browse Hierarchy	0	0	1	0	0
Create Hierarchy	0	0	1	1	0
Edit Hierarchy	0	0	1	1	0
Delete Hierarchy	0	0	1	1	0
Create Logical Node	0	0	1	1	0
Delete Logical Node	0	0	1	1	0
Rename Logical Node	0	0	1	1	0
Add Field Object	0	0	1	1	0
Remove Field Object	0	0	1	1	0
Create Logic and Reaction Rule	0	1	1	1	0
Edit Logic and Reaction Rule	0	0	1	1	0

(c) Traceability view

After the text is processed, users can open a new editor to conduct analyses for the CCCs and their relations with the use cases. The editor will create a persistent file with extension "rea". Figure 5 shows a snapshot of this editor, where the analysts are free to accept, modify or delete any of the concerns detected, based on their understanding of the requirements. In order to identify CCCs, users just have to press a button labeled "Rule Mine CCC" to execute the predefined queries loaded in *REAssistant* with the rule-based engine. The queries codify knowledge about concerns and how they relate semantically to natural language expressions, and were defined by experienced analysts to cover a wide range of software domains. Anyway, our tool has an editor in which analysts can customize the rules at any time. Let us assume that the analyst selects the rules associated to the PERFORMANCE concern. The execution of the rules will mark the sentences that are potentially crosscut by the concern. The tool can display the crosscutting relations using different colors on the text and at two levels of granularity: at the level of use cases (global view, Figure 7a), or at the level of behavior steps for a given use case (detailed view, Figure 7b). There is also another view within the concern editor that computes a traceability matrix among the use cases and the concerns. In this way, the analyst can easily get insights on: how a given concern impacts on the use cases, whether a concern is well-modularized (in terms of a narrow set of use cases), or how a given use case gets affected by several concerns.

REFERENCES

[1] A. Moreira, R. Chitchyan, J. Araujo, and A. Rashid, Eds., *Aspect-Oriented Requirements Engineering*. Springer Berlin Heidelberg, 2013, vol. XIX.
[2] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed., ser. SEI Series in Software Engineering. Addison-Wesley Professional, October 2012.
[3] E. Baniassad, P. Clements *et al.*, "Discovering early aspects," *IEEE Software*, vol. 23, no. 1, pp. 61–70, 2006.
[4] A. Sampaio, A. Rashid, R. Chitchyan, and P. Rayson, "EA-Miner: towards automation in aspect-oriented requirements engineering," *Transactions on Aspect-Oriented Software Development III*, pp. 4–39, 2007.
[5] A. Rago, C. Marcos, and A. Diaz-Pace, "Uncovering quality-attribute concerns in use case specifications via early aspect mining," *Requirements Engineering*, vol. 18, no. 1, pp. 67–84, March 2012. [Online]. Available: http://dx.doi.org/10.1007/s00766-011-0142-z
[6] ——, "Assisting requirements analysts to find latent concerns with REAssistant," *Automated Software Engineering*, June 2014.
[7] D. Ferrucci and A. Lally, "UIMA: an architectural approach to unstructured information processing in the corporate research environment," *Natural Language Engineering*, vol. 10, no. 3-4, pp. 327–348, 2004.
[8] A. Rago, C. Marcos, and A. Diaz-Pace, "Identifying duplicate functionality in textual use cases by aligning semantic actions," *Software and Systems Modeling*, August 2014.