Towards a Goal-Oriented Requirements Methodology Based on the Separation of Concerns Principle

Geórgia Maria C. de Sousa and Jaelson Brelaz de Castro

Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Caixa Postal 7851, CEP: 50.732-970, Recife – PE – Brasil
{gmcs,jbc}@cin.ufpe.br

Abstract. One of the most important principles in Software Engineering is the separation of concerns. When this principle is correctly applied, it helps to promote comprehensibility, maintainability and reusability of software system artifacts. However, often, crosscutting requirements are specified in a scattered and tangled fashion. Therefore, in order to produce better requirements specifications, this paper presents a Goal-oriented REquirements Methodology founded on the Separation of Concerns principle. We named this methodology GREMSoC. It intends to better manipulate with crosscutting requirements in Requirements Engineering process providing a way to represent crosscutting requirements apart from the requirements they affect and to specify the composition between them in a noninvasive way. A case study of an Internet Banking System illustrates the use of this methodology.

1. Introduction

The growing complexity of software and the demand for rapid development have increased the importance of reusability, maintainability and comprehensibility of software system artifacts. In order to achieve these qualities, some Software Engineering principles can be applied throughout the software development process, from requirements to implementation: modularity, abstraction, rigor and formality, separation of concerns and anticipation of change [1].

Separation of concerns (SoC) means dealing with different issues of a problem individually so that it is possible to concentrate on each one separately [40]. The main advantages of applying this principle are: (i) decrease the complexity of the software development by concentrating on different issues separately; (ii) support division of efforts and separation of responsibilities [1]; and (iii) improve the modularity of software systems artifacts. Furthermore, when the SoC principle is correctly applied, the software artifacts tend to be cohesive and loosely coupled. The consequences of this are manifold: an artifact can be understood in isolation; the software artifacts can be reused; and changes to one artifact have a limited effect on others artifacts [2].

Nevertheless, due to the intrinsic relationship between some requirements, especially between non-functional and functional ones, sometimes references to the specification of one requirement is scattered across multiple artifacts (scattering) and one requirement artifact contains references to multiple requirements (tangling). For instance, each non-functional requirement (NFR) is normally repeated in every use

case that the NFR affects [3,4]. This kind of representation makes it difficult to keep all the requirements updated and in the correct place. Moreover, the scattering and tangling in requirements artifacts makes it difficult not only the requirements evolution and maintenance, but also the requirements reuse and comprehension [5].

Therefore, in order to improve reusability, maintainability and comprehensibility of requirements specifications, we advocate that requirements should be represented according to the separation of concerns principle.

This work describes the first initiative towards a Goal-oriented REquirements Methodology founded on the SoC principle. We named this methodology GREMSoC. It provides a way to represent crosscutting requirements[1] apart from the requirements they affect and to specify the composition between them in a noninvasive way.

This work is organized as follows. In Section 2, we briefly present the background of our approach, describing the main concepts used in Requirements Engineering area. Section 3 presents the GREMSoC methodology that is illustrated by a case study in Section 4. We review related work and compare them with our proposal in Section 5. Finally, in Section 6, we present our conclusions and future work.

2. Background

This section presents key concepts used in our methodology, from goals and requirements to characteristics and approaches to deal with functional and non-functional requirements.

2.1 Goals and Requirements

A goal is a high-level objective that the system under consideration should achieve [6]. In turn, a requirement is a description of a system service or constraint needed by a user to achieve a goal [7]. Thus, a requirement specifies how a goal should be accomplished by a proposed system [8]. Goals whose satisfaction cannot be established in a clear-cut sense are named *softgoals*[2] [9].

Requirements Engineering (RE) is the branch of Software Engineering concerned with the identification of the goals to be achieved by the system, the operationalizations of such goals into requirements, and the assignment of responsibilities for the resulting requirements to agents such as humans, devices, and software [10,11]. The RE process generally includes the following intertwined activities: domain analysis, elicitation, analysis and negotiation, specification and validation.

Besides the services and the quality restrictions that a system should provide, a requirements specification also should include the application domain information and the organizational context where the system will be applied [13]. From this point of view, we can identify three types of requirements: functional, non-functional and organizational requirements. It is important to distinguish the types of requirements

[1] A crosscutting requirement is the one that affects more than one requirement in the system

[2] To indicate that a softgoal satisfying is accomplished within acceptable limits, from here on, we will use the term *to satisfice* [12] rather than the term *to satisfy* with *softgoal*s.

since they have different characteristics and therefore, by means of this distinction, we can choose the adequate methods and techniques that should be used during Requirements Engineering process.

The traditional approaches used in software development are driven by functional requirements and their focus is on achieving the desired functionality of the system. However, a broader view of software development is one that goes beyond the description of system functionalities, also including organizational and non-functional requirements in a requirements specification [13].

2.2 Functional Requirements

Functional requirements (FRs) capture the intended behavior of the system in terms of services, tasks or functions the system is required to perform [14].

This kind of requirement is generally specified by means of inputs, processing, outputs, controls, exceptions and entities. The more common techniques for specifying functional requirements are: use cases, scenarios, data flow diagrams and state transition diagrams. Currently, use case [15] is the most used practice to capture and represent functional requirements, especially in object-oriented software development.

A use case describes a set of interactions between actors and the system necessary to deliver the service that satisfies the user goal. It also includes possible alternative sequences that may satisfy the goal, as well as sequences that may lead to failure in completing the service because of exceptional behavior, error handling, etc [16].

2.3 Non-Functional Requirements

Non-functional requirements (NFRs) are requirements that impose restrictions on the product being developed (product requirements), on the development process (process requirements), or they specify external constrains that the product/process must meet (external requirements) [17]. These constraints usually narrow the choices for constructing a solution to the problem.

The distinction between functional and non-functional requirements may cause confusion. Some NFRs characteristics are used to distinguish them from functional requirements:

- NFRs are focused on how the software must perform something instead of focused on what the software must do [18];
- NFRs “cross-cut” software functionality [19];
- NFRs express constraints or conditions that need to be satisfied by functional requirements and/or design solutions [9]
- Different from functional requirements that can fail or succeed, NFRs rarely can be completely met: their satisfying is accomplished within acceptable limits [12].

Most approaches to systematically deal with non-functional requirements focus on specific NFRs such as security [20] and performance [21]. In turn, the NFR Framework [9,12,22] can be applied to a variety of non-functional requirements. In this approach, NFRs are treated as *softgoal*s to be satisficed by means of operationalizations (operations, data representations, architectural decisions, etc).

Although non-functional requirements are crucial for system development success, they are seldom analyzed and, even when they are considered, they are generally poorly documented:

1. They are often stated in requirements specifications just as abstract, vague and informal declarations such as: "the system should have good security, performance, confidentiality, usability". This kind of declaration makes it difficult to analyze and to verify how to meet the non-functional requirement. Moreover, it may be ambiguous since different interpretations about the real meaning of the NFR are possible. For those reasons, we advocate that abstract declarations of NFRs need to be broken down into smaller components and then converted into operationalizations that together contribute for achieving these NFRs.
2. Normally, non-functional requirements are not documented in a specific artifact. On the contrary, they are declared repeatedly in each functional artifact affected by them. This fact contradicts the separation of concerns principle and, consequently, the requirements comprehension, evolution and reuse are damaged.

3. The GREMSoC Methodology

The GREMSoC methodology purpose is to promote an approach to improve reusability, maintainability and comprehensibility of requirements specifications by means of the separation of concerns principle. The outline of GREMSoC can be visualized in Fig. 1.

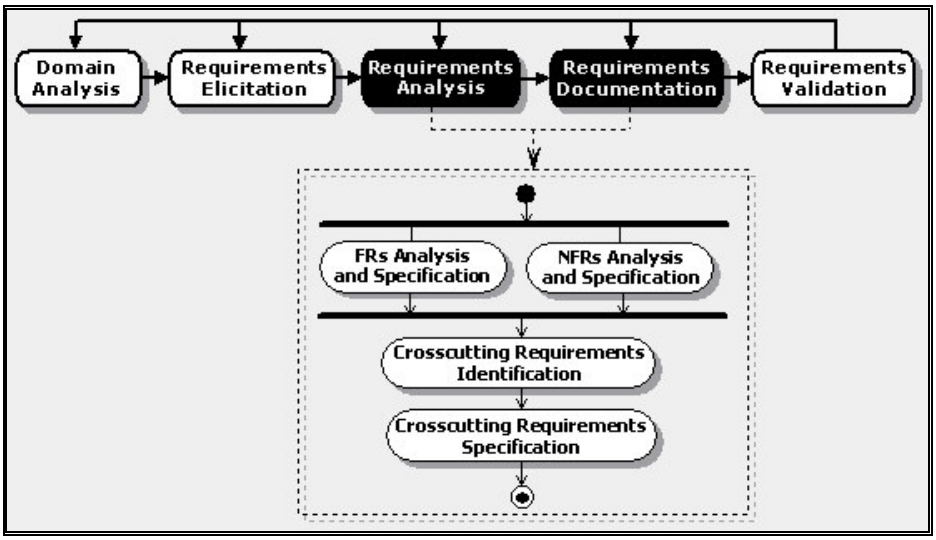

Fig. 1. Outline diagram of GREMSoC Methodology

GREMSoC intends to better manipulate with crosscutting requirements in analysis and documentation activities since, in the most part of current approaches, these requirements are specified in a scattered and tangled fashion. This methodology

separately considers both the analysis and the specification of functional and non-functional requirements. Moreover, the relationships between crosscutting and non-crosscutting requirements are documented apart.

In general, every SoC approach includes the following activities: identify concerns; separate concerns; represent separated concerns; and compose concerns [23]. Since our methodology is founded on the separation of concerns principle, it is interesting to show the correspondence between the common activities of SoC approaches and the GREMSoC activities. Table 1 presents this correspondence and furthermore, it presents the selected techniques for each proposed activity. It is worth mentioning that, since GREMSoC is a goal-oriented methodology, all selected techniques are goal-oriented.

Table 1. Relating the common activities of SoC approaches with the GREMSoC activities and their selected techniques/representations

SoC Activity	GREMSoC Activity	Selected Technique/ Representation
CONCERNS IDENTIFICATION	Goal-Based Requirements Elicitaion	—
CONCERNS SEPARATION	Functional Requirements Analysis	Use Cases with Goals [16, 24]
	Non-Functional Requirements Analysis	NFR Framework [9, 12, 22]
CONCERNS REPRESENTATION	Functional Requirements Specification	Use Case Specification [25]
	Non-Functional Requirements Specification	Softgoal Interdependencies Graph [12] and Operationalization Specification
CONCERNS COMPOSITION	Crosscutting Identification	—
	Crosscutting Specification	Composition Table

A concern is a vague declaration, generally corresponding to a high-level goal for the system being developed [17]. Hence, in our approach, the identification of concerns is related to goal identification. This activity is facilitated when stakeholders explicitly state the goals or they are declared in preliminary material available. Nevertheless, most often, they are implicit and thus, the goal elicitation has to be undertaken [6]. Although this activity is beyond the scope of our work, we will cite some ways to accomplish it:

- Searching for intentional keywords in the preliminary documents provided, interview transcripts, etc. [10].
- Analyzing the current system and, from this analysis, to list the goals that solve/reduce the problems and deficiencies found in the current system [6].

The following subsections outline the approaches adopted by GREMSoC to separate, represent and compose concerns.

3.1 Functional Analysis and Specification

We adopt the Cockburn's approach [16,24] for the functional analysis due to two reasons: it is goal-oriented and use-case driven. The steps of this approach are shown as a sequence of activities in Fig. 2.

Fig. 2. Steps of Cockburn's Approach

The first step of Cockburn's approach is identifying the actors and their functional goals that the system will support. A use-case specifies how a functional goal will be reached. Eventually it also specifies the fail conditions to reach the goal and how these conditions will be handled [24]. Then, the next step is specifying the main success scenario for each use cases that will accomplish a user functional goal. After that, it will be necessary to identify the failure conditions that could occur in the main success scenarios, without considering how the system must handle them all. The last step is specifying how the system is supposed to respond to each failure. This step can reveal a new actor or a new goal that needs to be supported.

Table 2. Basic Use Case Template (adapted from [25])

USE CASE # N - <name >	
GOAL IN CONTEXT	<a longer statement of the goal, if needed>
PRECONDITIONS	<What we expect is already the state of the world>
PRIMARY ACTOR	<A role name for the primary actor, or description>
MAIN SUCCESS SCENARIO	
STEP	**ACTION**
<step>	<action description>
EXTENSIONS	
STEP	**BRANCHING ACTION**
<step>	<condition causing branching> : <action>
SUB-VARIATIONS	
STEP	**BRANCHING ACTION**
<step>	<list of variations>

Although use cases are part of UML [27], there is no standard template for specifying them. In this work, we use a simplified version of Cockburn's Basic Use-Case Template [25] (see Table 2).

3.2 Non-Functional Analysis and Specification

The NFR Framework [9,12,22] was chosen for the non-functional analysis because it deals with NFRs in a systematic way. In this approach, NFR are treated as *softgoals* to be achieved, not in an absolute sense, but in a sufficient or satisfactory way. The NFR *softgoals* and their interdependencies are graphically represented in a Softgoal Interdependency Graph (SIG).

In the NFR Framework, abstract and subjective NFRs (e.g. security, performance) should be represented at the top of the SIG. Then, each one should be iteratively refined into more specific *softgoal*s. At some point, when the NFRs *softgoal*s have been sufficiently refined, it will be possible to operationalize these *softgoal*s, i.e. providing more concrete and precise mechanisms (e.g. operations, processes, data representations, architectural decisions, etc.) to achieve it. The next step is choosing which operationalizations the system will adopt. During refinement and operationalization steps, contributions and possible conflicts should be established, defining the impact of *softgoal*s to each other and identifying priorities (indicated by "!" or "!!").

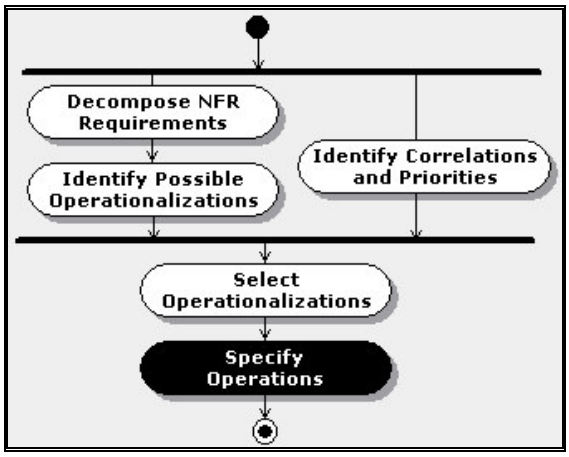

Fig. 3. GREMSoC approach to analyze and specify NFRs

Fig. 3 exhibits the sequence of activities adopted by the GREMSoC approach to analyze and specify non-functional requirements. All the activities are the same of NFR Framework, except the last one: *specify operations*. This activity was added in order to provide a detailed description of each selected operationalization (except architectural decisions) that should be done in a specific template (see Table 3).

Table 3. Operationalization Template (adapted from [25])

OPERATIONALIZATION # N - <NAME >	
NFR SOFTGOAL	<the softgoal hierarchy to which this operationalization contributes to satisfice>
GOAL IN CONTEXT	<a longer statement of the goal, if needed>
PRECONDITIONS	<what we expect is already the state of the world>
PRIMARY ACTOR	<a role name or description for the primary actor, if needed>
MAIN SUCCESS SCENARIO	
STEP	**ACTION**
<step >	<action description>
EXTENSIONS	
STEP	**BRANCHING ACTION**
<step >	<condition causing branching> : <action>
SUB-VARIATIONS	
STEP	**BRANCHING ACTION**
<step >	<list of variations>

3.3 Crosscutting Requirements Identification and Composition

At this point of the approach, the requirements have already been individually represented. However, there are requirements that need to be applied in some point of another requirements specification. They are called crosscutting requirements.

According to the separation of concerns principle, the composition between crosscutting and non-crosscutting requirements should be noninvasive. Therefore, besides a unit to encapsulate each concern, any SoC mechanism must also provide a composition mechanism apart to permit the integration of separate concerns and to give them some coherence [23]. This stage of GREMSoC approach aims to address this issue.

Firstly, it is necessary to identify among the requirements which of them are crosscutting. A requirement is crosscutting if it affects other requirements in such a way that the crosscutting requirement needs to be applied in some point of other requirements specification.

The next step is specifying how each crosscutting requirement affects the requirements it transverses. In order to do that, we provide a composition table (see Table 4). This table should be specified for each crosscutting requirement to indicate: (i) which artifacts it will affect (first column); (ii); when the composition should be done (second column); (iii) in which point of the artifact the crosscutting should be applied (fourth column); and (iv) how the composition should be done (third column).

To determine how a crosscutting requirement is applied in a particular point that it affects, we use the following composition rule operators [26]:

- Overlap: indicates that the crosscutting requirement should be applied before or after the step of the scenario it transverses;

- Override: indicates that the crosscutting requirement superposes the scenario's step it transverses. This means that the behaviour described by the crosscutting requirement substitutes the behaviour defined by the step;

The crosscutting requirement should be identified at the table's top. We stipulate that the identification of the requirement artifacts should follow this syntax: {OP| UC} #N <name>. OP indicates that the crosscutting requirement is an operationalization; UC indicates that it is a use-case. #N represents the artifacts number and <name> its name.

Table 4. Composition Table

CROSSCUTTING REQUIREMENT: {OP| UC} # N <NAME>			
AFFECTED REQUIREMENT	CONDITION (OPTIONAL)	COMPOSITION RULE OPERATOR	AFFECTED POINT
{OP| UC} # N <name>	condition of the composition[3]	{overlap.after | overlap.before | override}	Step of the Scenario

4. Applying the GREMSoC Methodology

We apply the GREMSoC methodology to an Internet Banking System, which is a well-known application domain whose success rests on the adequate treatment of non-functional requirements [28]. In the sequel, we will outline how the GREMSoC methodology can be used in the requirements analysis and documentation activities.

4.1 Functional Analysis and Specification

The main functional goal of an Internet Banking System is to allow bank clients to perform banking transactions through the Internet such as query transactions (account balance and account statement) and financial transactions (transfers, bill payments, etc.). Table 5 and Table 6 present the specification of two functional requirements.

Table 5. Use-Case Specification for *View Account Statement*

USE CASE # 01 - VIEW ACCOUNT STATEMENT	
GOAL IN CONTEXT	Visualize debits and credits' historic of an account in a p eriod
PRECONDITIONS	The account should have been identified
PRIMARY ACTOR	Bank Client
MAIN SUCCESS SCENARIO	
STEP	ACTION
1	The actor informs the period
2	The system exhibits a list of debits and credits and its respective dates, descriptions and document numbers.

[3] The default condition is always

EXTENSIONS	
STEP	**BRANCHING ACTION**
1a	The actor informs an invalid period: 1a1. An error message is exhibited 1a2. The user repeat step 1
SUB-VARIATIONS	
STEP	**BRANCHING ACTION**
1a	Invalid Period: Period exceed 120 days; The initial date is subsequent to the final date; The initial date or the final date is subsequent to today's date.

Table 6. Use-Case Specification for *Transfer of Funds*

USE CASE # 02 – TRANSFER OF FUNDS	
GOAL IN CONTEXT	To transfer funds from an user account to another account
PRECONDITIONS	The bank client account should have been identified
PRIMARY ACTOR	Bank Client (user)
MAIN SUCCESS SCENARIO	
STEP	**ACTION**
1	The user informs the value of the transfer, the target account and branch number.
2	The user confirm the informed data
3	The system debits the value from the user account and credits in the target account
4	A confirmation of the transaction success is exhibited
EXTENSIONS	
STEP	**BRANCHING ACTION**
1a	Target account does not exist: 1a1. The system exhibits an error message 1a2. The user repeat step 1 or cancel the operation
3a	The user account does not have enough funds: 3a1. The system exhibits an error message 3a2. The user repeat step 1 or cancel the operation

4.2 Non-Functional Analysis and Specification

One of the most important non-functional requirements when building information systems to be used on the Internet is security, i.e., protecting transactions against unauthorized access. Fig. 4 exhibits the successive decompositions and operationalizations to satisfice the security *softgoal*.

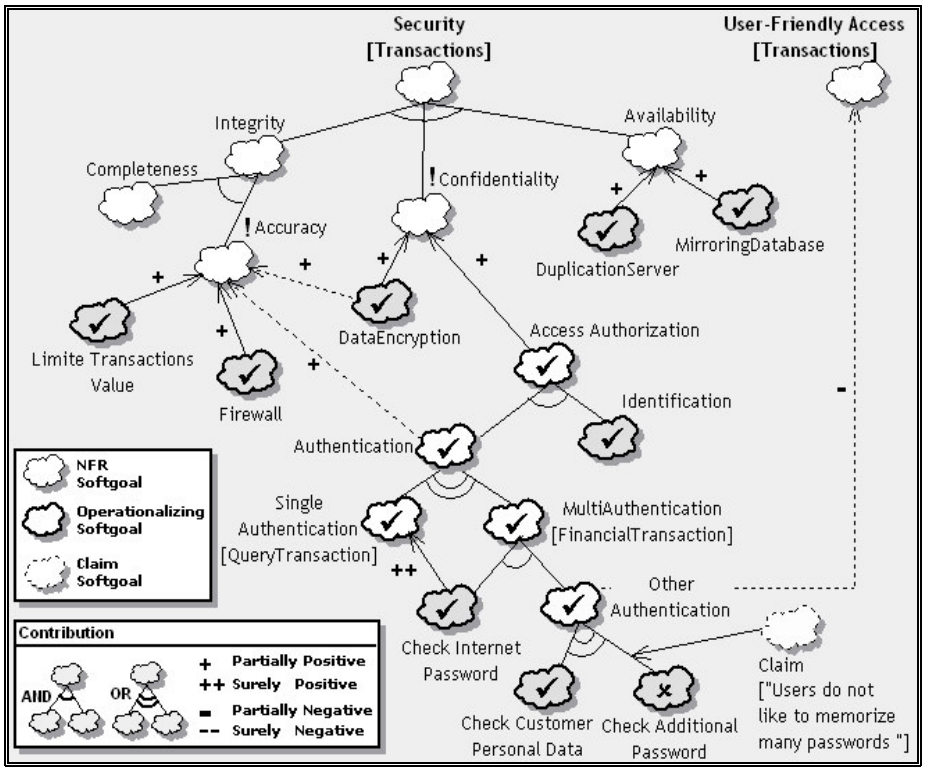

Fig. 4. Softgoal Interdependency Graph for Security [Transactions] Softgoal

Firstly, the security *softgoal* was decomposed in three others ones: *Confidentiality, Integrity* and *Availability*. Integrity, in turn, was subdivided in *Completeness* and *Accuracy*. At this point, the security *softgoal* have been sufficiently refined and then it will be possible to operationalize it (i.e. choose specific solutions for satisficing the offsprings of the security softgoal).

Taking the *Confidentiality softgoal*, two operationalizations can contribute positively to its satisficing: *Data Encryption* and *Access Authorization*. The former ensures that the information can only be deciphered by the system; the latter ensures that users are in fact whom they claim to be.

Access Authorization, in turn, can be decomposed in two others *operationalizing softgoals*: *Identification* and *Authentication* (Single for query transactions or Multiple for financial transactions). The *Check Internet Password operationalization* satisfices the *Single Authentication softgoal*. This operationalization in conjunction with *Other Authentication softgoal* satisfices *Multi Authentication softgoal*. Lastly, the *Other Authentication* softgoal can be *satisficed* either by *Check Customer Personal Data* or else by *Check Additional Password*. However, the *Other Authentication satisficing* contributes negatively for *user-friendly access* concern.

The same reasoning was applied for the *Accuracy* and *Availability softgoals*.

Considering that the possible solutions for the system are sufficiently detailed and that no other alternatives need to be analyzed, it is appropriate to select among

alternatives (bottom nodes of a SIG), accepting (v) or rejecting (x) each of them. In our case study, the only rejected *operationalizing softgoal* was *Check Additional Password.* The reason for that is represented in a claim *softgoal* being related to the client difficulty to memorize many passwords.

Table 7 exhibits the specification for *Check Internet Password* operationalization.

Table 7. Specification for *Check Internet Password* Operationalization

OPERATIONALIZATION # 01 – CHECK INTERNET PASSWORD	
NFR SOFTGOAL	SECURITY => Confidentiality => Access Authorization => Authentication => Single Authentication
GOAL IN CONTEXT	Check the actor identity by means of a request for his account's Internet Password
PRECONDITIONS	The bank client account should have been identified
PRIMARY ACTOR	The bank client
MAIN SUCCESS SCENARIO	
STEP	**ACTION**
1	The actor informs the Internet Password
2	The system compares the informed Internet Password with the account's Internet Password
3	An Ok status is returned
EXTENSIONS	
STEP	**BRANCHING ACTION**
2a	The informed Internet Password is wrong (more than three times): 2a1.The Internet Password is blocked and the goal fails
2b	The informed Internet Password is wrong (less than three times): 2b1. An error message is exhibited 2b2. The user repeat step 1 or cancel the operation
SUB-VARIATIONS	
STEP	**BRANCHING ACTION**
1	The actor can inform the Internet Password by means of: Numeric Keyboard; Virtual Keyboard

4.3 Crosscutting Requirements Identification and Composition

We have identified the following crosscutting requirements: *Limit Transactions Value, Data Encryption, Identification, Check Internet Password and Check Customer Personal Data*.

To exemplify how we identify crosscutting requirements, take in particular the *Check Internet Password Request* operationalization: it needs to be applied in two use-cases *View Account Statement* and *Transfer of Funds*, and then it is a crosscutting requirement. In common approaches, the composition between these artifacts would be done in an invasive way, i.e. there should be a reference to the *Check Internet Password* in both use-cases *View Account Statement* and *Transfer of Funds*. This kind of approach makes it difficult to understand, to maintain and to reuse those use-cases.

Note that, using this kind of approach, *View Account Statement* and *Transfer of Funds* use-cases cannot be understood in isolation since to do this it is also necessary to understand the *Check Internet Password* operationalization. Moreover, if the *Internet Password Request* is no longer necessary, for example, then all artifacts affected by it should be altered. Also, imagine, for example, that another system requires the same functionality provided by the *View Account Statement* use-case, but it does not need *the Internet Password Request*. In this case, the complete reuse of this use case will not be possible since the use case is tangled with *Check Internet Password* crosscutting requirement.

Our approach intends to address these issues. As was explained before (Section 3.3), in GREMSoC methodology the composition between a crosscutting requirement and the requirements artifacts it affects should be specified in a table apart. This fact helps to promote comprehensibility, maintainability and reusability of requirements artifacts. Furthermore, since the composition is done only in one artifact, it is easy to determine the range of artifacts that a crosscutting requirement affects and how it affects each one of them.

Table 8 illustrates the specification for the composition between the *Check Internet Password* operationalization and the artifacts it affects, according to our approach. This specification indicates that the *Check Internet Password* operationalization should be applied after the step 2 in *View Account Statement* use-case; and before the step 2 in *Transfer of Funds* use-case.

Table 8. Specification for the Composition of *Check Internet Password*

CROSSCUTTING REQ: OP #01 – CHECK INTERNET PASSWORD			
AFFECTED REQUIREMENT	CONDITION	COMPOSITION RULE OPERATOR	AFFECTED POINT
UC #01 - View Account Statement	always	overlap.before	Step 2
UC #02 - Transfer of Funds	always	overlap.after	Step 2

5. Related Works

There are two kinds of related works to our proposal: (i) works concerned with functional and non-functional requirements and their relationships; and (ii) research related to SoC approaches in Requirements Engineering process. Both of them are outlined in the following paragraphs.

5.1 Approaches to Integrate Functional and Non-Functional Requirements

An interesting strategy for dealing with non-functional requirements and integrating them into Entity-Relationship and Object-Oriented conceptual models was presented in [29,30]. In this strategy, the software development process is carried out through two independent cycles, one regarding the functional requirements of the system and the other the non-functional requirements. The integration of these both views is done

by representing the operationalizations found in the non-functional view inside of functional models such as entity-relationship and class diagrams.

A similarity of this approach and GREMSoC methodology is that both deal separately with functional and non-functional requirements. However, Cysneiros' approach does not apply the separation of concerns principle since the models produced by their approach have weak cohesion and strong coupling. The former occurs because the models contain data and methods that are not direct related with its functionality. The latter occurs due to the models mix non-functional and functional properties in a same entity and this fact makes it difficult to isolate each one.

5.2 SoC Approaches

Many Separation of Concern mechanisms have been proposed over the time, including procedures, objects, packages and so forth. In the last years, however, the research has been concerned in providing approaches for separation of crosscutting concerns, also called of Advanced Separation of Concerns (ASOC). Many ASoC approaches have been proposed such as Subject-Oriented Programming and Design [31], Composition Filters [32] and Multidimensional Separation of Concerns [33]. However, Aspect-Oriented Paradigm [34] has been the one that has attracted more research lately. In short, this paradigm employs special abstractions known as aspects to encapsulate (and thereby separate) crosscutting concerns; while non-crosscutting concerns are encapsulated in components such as classes or modules. The composition between aspects and components is accomplished by means of a mechanism named weaving.

Among the research works in Aspect-Oriented Requirements Engineering (AORE), we can cite: [35,36,26,37,38,39]. In the sequel, we will comment about some of them in particular.

Moreira et al. [26] and Araujo et al. [36] have proposed a model to identify and specify quality attributes that crosscut requirements, including their systematic integration into UML models at the requirements stage. The process proposed by these approaches consists of three main activities: identification, specification and composition of crosscutting requirements. GREMSoC methodology has two points in common with those approaches: (i) the process model and (ii) the composition rule operators used in the composition activity. However, whereas our approach makes the composition on an artifact apart, the approach of Moreira et al. [26] and Araujo et al. [36] makes the composition inside UML models, in an invasive way. Another difference among these works is the treatment of non-functional requirements: they deal with NFRs as abstract attributes such as security, performance, etc; GREMSoC, in turn, manipulates with non-functional requirements operationalizations [12] (i.e. operations, processes, data representations, constraints, etc) that contributes to satisfy an abstract NFR.

In our previous work [39], we have proposed an adaptation of the NFR Framework in order to improve the mapping and the composition of crosscutting requirements onto artifacts at later development stages. This work was based on AORE generic models presented in [35, 37]. The main contributions of this approach were: (i) the adaptation of an existing requirements technique to include concepts of Aspect-Oriented-Paradigm; and the use of NFR operationalizations [12], instead of abstract

declarations of NFRs, in the mapping and composition of crosscutting NFRs. This last contribution was incorporated in GREMSoC methodology.

There are some differences between our previous approach and GREMSoC one: the former is concerned only with non-functional crosscutting requirements and it was specifically developed for the Aspect-Oriented Software Development (AOSD); whereas GREMSoC handle with functional and non-functional crosscutting requirements and it can be used not only in the context of AOSD but also in others kinds of software development paradigms.

6. Conclusion

In software development, it is important to specify system artifacts with a clear separation of concerns. The motivation for this is manifold: the understanding and the maintenance of one artifact can be accomplished without having to know all of the details of the larger system, as well as the reuse of system artifacts is promoted.

However, sometimes it is difficult to apply the separation of concerns principle in the specification of requirements artifacts due to the strong relationship and the interdependencies among some requirements. This fact is true especially in the specification of non-functional requirements that are naturally crosscutting. Very often, they are scattered and tangled in functional specification.

The purpose of GREMSoC methodology is to promote an approach to solve this problem by selecting goal-oriented techniques to specify requirements separately and by providing a way to compose these requirements in an artifact apart. This fact, therefore, allows that crosscutting requirements are specified separately from the requirements they affect. Therefore, we advocate that using GREMSoC methodology the comprehensibility, the maintainability and the reusability of crosscutting requirements specification are improved.

Our approach relies on well-known techniques such as use-cases for functional requirements and the NFR Framework for non-functional requirements. Furthermore, the composition mechanism is quite simple.

Our future work will focus on applying the GREMSoC methodology in others case studies and evaluating the use of this methodology in Aspect-Oriented Software Development.

7. References

1. Ghezzi, C.; Jazayeri, M. and Mandrioli, D. "Fundamentals of Software Engineering". Prentice Hall, 1991. ISBN0-13-820432-2.
2. Hursch, W. and Lopes, C. "Separation of Concerns". Technical Report NU CCS -95-03, Northeastern University, 1995.
3. Jacobson I., Booch G., and Rumbaugh , J. "The Unified Software Development Process", Addison-Wesley, 1999. ISBN 0-201-57169-2.
4. Malan R. and Bredemeyer D. "Defining Non -Functional Requirements", 2001. Available at http://www.bredemeyer.com/papers.htm. Last access: Sept. 2003.
5. Baniassad, E.; Murphy, G.; Schwanninger, C. and Kircher, M. "Managing Crosscutting

Concerns During Software Evolution Tasks: an Inquisitive Study", Proceedings of the 1st International Conference on Aspect-Oriented Software Development, April 22-26, 2002, Enschede, The Netherlands

6. van Lamsweerde, A. "Goal-Oriented requirements Engineering: A Guided Tour", Proceedings of the 5th International Symposium on Requirements Engineering (RE' 01), IEEE CS Press, 2001, pp.249-261.
7. IEEE-Std. '610' "IEEE Standard Glossary of Sof tware Engineering Terminology". Institute of Electrical and Electronics Engineers, New York, 1990.
8. Anton, A."Goal -based Requirements Analysis," Proc. 2nd IEEE Int'l Conf. Requirements Engineering, CS Press, Los Alamitos, Calif., 1996, pp. 136–144.
9. Mylopoulos, J.; Chung, L. and Nixon, B. "Representing and Using Non -Functional Requirements: A Process-Oriented Approach". IEEE Transactions on Software Engineering, Vol. 18, No. 6, June 1992, pp. 483-497.
10. van Lamsweerde, A. "Requirements Engineerin g in the Year 00: A Research Perspective". Invited Keynote Paper, Proc. ICSE'2000: 22nd International Conference on Software Engineering, ACM Press, 2000, pp. 5-19.
11. Castro, J.; Kolp, M. and Mylopoulos, J. "Towards Requirements -Driven Information Systems Engineering: The Tropos Project". In Information Systems, Vol. 27, Elsevier, Amsterdam, The Netherlands, 2002, pp. 365–389.
12. Chung, L; Nixon, B.; Yu, E. and Mylopoulos, J. "Non -Functional Requirements in Software Engineering".Boston:Kluwer Academic Publishers, 2000. ISBN 0-7923-8666-3.
13. Loucopoulos P. and Karakostas V. "System Requirements Engineering", McGraw Hill International Series in Software Engineering, 1995
14. Malan R. and Bredemeyer D. "Functional Requirements and Use Cases", 1999. Av ailable at http://www.bredemeyer.com/papers.htm. Last access: Sept. 2003.
15. Jacobson, I.; Christerson, M.; Johnson, P. and Overgaard, G. "Object -Oriented Software Engineering: A Use-Case Driven Approach". Addison -Wesley, 1992. ISBN 0-201-54335-0
16. Cockburn, A. "Writing Effective Use Cases". Addison -Wesley, 2000. ISBN: 0201702258.
17. Kotonya, G. and Sommerville, I. "Requirements Engineering: Processes and Techniques". Wiley, 1998. ISBN 0-471-97208-8.
18. Cysneiros, L. and Leite, J. "Non -functional Requirements: From Elicitation to Modelling Languages". In Proceedings of the 24th International Conference on Software Engineering, Tutorial Session, 2002, pp: 699 - 700, ISBN:1-58113-472-X.
19. Burge, J. and Brown, D. "NFRs: Fact or Fiction", Computer Science Technical Report, Worcester Polytechnic University, 2002, WPI-CS-TR-02-01.
20. Chung, L. "Dealing with Security Requirements During the Development of Information System". Fifth International Conference on Advanced Information System Engineering (CAiSE'93). Springer -Verlag, Berlin, 1993, pp. 5-30.
21. Nixon, B. "Dealing with Performance Requirements During the Development of Information Systems". In Proceedings of the RE' 93, IEEE International Symposium on Requirements Engineering, San Diego, California, 1993, pp. 42-49.
22. Mylopoulos, J.; Chung, L.; Liao, S.; Wang, H. and Yu, E. "Exploring Alternatives during Requirements Analysis". IEEE Software Jan/Feb 2001, pp. 2 -6.
23. Aksit, M.; Tekinerdogan, B. and Bergmans, L. "The Six Concerns for Separation of Concerns", in Proceedings of ECOOP 2001 Workshop on Advanced Separation of Concerns, Budapest, Hungary, June 18-22, 2001.
24. Cockburn, A. "Structuring Use Cases with Goals," Journal of Object-Oriented Programming, Sep/Oct, 1997, pp. 35-40, and Nov/Dec, 1997, pp. 56-62.
25. Cockburn, A. "Basic Use Case Template", 1998. Available at http://members.aol.com/acockburn/papers/uctempla.htm . Last access: Sept. 2003.
26. Moreira, A.; Araújo, J. and Brito, I. "Crosscutting Quality Attributes for Requi rements

Engineering", 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002), ACM Press, Italy, July, 2002.
27. Booch, G.; Rumbaugh, J. and Jacobson, I. "The Unified Modeling Language User Guide", Addison-Wesley, 1999. ISBN: 0201571684.
28. Patrício, L., Falcão e Cunha, J. and Fisk, R. "The Relevance of User Experience Requirements in Interface: Design – a Study of Internet Banking". 6th Ibero-American Workshop on Requirements Engineering and Software Environments - IDEAS' 2003, Asunción, Paraguay, 30th April – 2nd May, 2003.
29. Cysneiros, L.; Leite, J. and Neto, J. "A Framework for Integrating Non-Functional Requirements into Conceptual Models". Requirements Engineering Journal – Vol 6, Issue 2 Apr. 2001, pp. 97-115.
30. Cysneiros, L. and Leite, J. "Using UML to Reflect Non-Functional Requirements". Proc. of the 11th CASCON, IBM Canada, Toronto, Nov 2001, pp. 202-216.
31. Harrison, W. and Ossher, H. "Subject-Oriented Programming (a Critique of Pure Objects). OOPSLA'93, 1993, pp 411-428, ACM Press.
32. Bergmans, L. and Aksit, M. "Composing Crosscutting Concerns Using Composition Filters". Commun. ACM, 44(10): 51–57, Oct 2001.
33. Ossher, H and Tarr, P. (2001) "Multi-Dimensional Separation of Concerns and the Hyperspace Approach". Proc. Symposium on Software Architectures and Component Technology: The State of the Art in Software Development. Kluwer Academic Publishers.
34. Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J.-M. and Irwin, J. "Aspect-Oriented Programming". In Proceedings of ECOOP '97, Springer-Verlag.
35. Rashid, A.; Sawyer, P.; Moreira, A. and Araújo, J. "Early Aspects: a Model for Aspect-Oriented Requirements Engineering", IEEE Joint Conference on Requirements Engineering, Essen, Germany, September 2002, Pages 199-202.
36. Araújo, J.; Moreira, A.; Brito, I. and Rashid, A. "Aspect-Oriented Requirements with UML", Workshop: Aspect-oriented Modeling with UML, UML 2002, Germany.
37. Rashid, A. Moreira, A. and Araujo, J. "Modularisation and Composition of Aspectual Requirements". 2nd International Conference on Aspect-Oriented Software Development, ACM, 2003, pp. 11-20.
38. Brito, I. and Moreira, A. "Towards a Composition Process for Aspect-Oriented Requirements". Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, March 17, 2003, Boston, USA.
39. Sousa, G.; Silva, I. and Castro, J. "Adapting the NFR Framework to Aspect-Oriented Requirements Engineering". XVII Brazilian Symposium on Software Engineering, Manaus, Brazil, October 2003.
40. Dijkstra, E. "A Discipline of Programming". Prentice-Hall, 1976.