Adapting the NFR Framework to Aspect-Oriented Requirements Engineering

Geórgia Maria C. de Sousa, Ismênia G. L. da Silva, Jaelson Brelaz de Castro

Centro de Informática – Universidade Federal de Pernambuco (UFPE)

Caixa Postal 7851, CEP: 50.732-970, Recife – PE – Brasil

{gmcs,igls,jbc}@cin.ufpe.br

Abstract. One of the most important principles in Software Engineering is separation of concerns. At first, the research towards applying that principle throughout the software development process has provided structured and object-oriented methods. However, when using those methods it is difficult to achieve separation of concerns such as security, performance, reliability, persistence, distribution, etc., the so-called crosscutting concerns. Hence, Aspect-Oriented Paradigm has emerged to address those issues. Similar to what happened with structured and object-oriented paradigms, in the last years, the Software Engineering community has been interested in propagating the Aspect-Oriented Paradigm to early stages of the software life cycle. The purpose of this paper is to give a contribution to Aspect-Oriented Requirements Engineering, adapting the NFR-Framework in order to improve the mapping of crosscutting non-functional requirements onto artifacts at later development stages and to make better the composition of those requirements with non-crosscutting ones.

Key-words: aspect-oriented requirements engineering, NFR Framework, non-functional requirements

1. Introduction

In Software Engineering, there are some central principles that should be applied throughout the software development process, from requirements to implementation: modularity, abstraction, separation of concerns, anticipation of change, etc.

Separation of concerns (SOC) allows us to deal with different issues of a problem individually so that we concentrate on each one separately. The main advantages of applying that principle are: (i) to decrease the complexity of the software development by concentrating on different issues separately; (ii) to support division of efforts and separation of responsibilities [Ghezzi et al., 1991] and (iii) to improve the modularity of software systems artifacts.

The software engineer should be equipped with appropriate methods and specific techniques that help him/her to apply the separation of concerns throughout the software development process. Those techniques are usually based on the adopted programming paradigm. In object-oriented methods, for example, the separated concerns are modeled as objects and classes; in structural methods, they are represented as procedures [Aksit et al., 2001]. Those approaches are well suited to most types of concerns related to a system's primary functionality, but they fail when dealing with concerns such as security, performance, reliability, persistence, distribution, etc., typically high-level non-functional requirements. The specification for those concerns cannot be clearly captured into one of the available building blocks and, thus, is spread throughout or tangled with the

specification for the primary functionality; therefore, those concerns are so-called crosscutting concerns.

In short, when using either the procedural or object-oriented programming paradigm, it is not possible to achieve the separation of crosscutting concerns in the design and implementation levels [Feng et al., 2001]. This fact makes the software difficult to understand, develop, evolve and maintain [Baniassad et al., 2002].

Similar to what happened with others approaches for separation of concerns, the first initiatives on separation of crosscutting concerns, also called of Advanced Separation of Concerns (ASOC), have focused on implementation-oriented phases of the life cycle, including Subject-Oriented Programming and Design [Harrison and Ossher, 1993; Clarke et al., 1999], Composition Filters [Bergmans and Aksit, 2001], Multidimensional Separation of Concerns [Ossher and Tarr, 2001] and Aspect-Oriented Programming (AOP) [Kiczales et al., 1997].

In particular, AOP employs special abstractions known as aspects to encapsulate (and thereby separate) crosscutting concerns. Any separation of concerns mechanism must also include powerful integration mechanisms, to permit the integration of separate concerns [Ossher and Tarr, 2001]. Thus, it is necessary to determine, in the aspect specification, in which points and how the crosscutting concern should be integrated (composed) with the components it affects[1].

In the last years, the Software Engineering community has been interested in propagating the Aspect-Oriented Paradigm to early stages of the software life cycle to facilitate the modeling of aspects in the design and implementation phases by means of: (i) the determination of the mapping of crosscutting requirements onto artifacts at later development stages and (ii) the understanding about how a crosscutting concern affects others requirements.

However, current works in Aspect-Oriented Requirements Engineering research area [Rashid et al., 2002; Araujo et al., 2002; Rashid et al., 2003; Brito and Moreira, 2003] express non-functional requirements (NFRs) in such a way that makes it difficult to compose and to map crosscutting concerns onto artifacts at later development stages. They are expressed as abstracts attributes that cannot be objectively verified such as security, performance, availability, etc. In this manner, the mapping and composition of these abstract crosscutting NFRs do not take in consideration the real modeling of aspects at later development stages since, in fact, those NFRs will need to be "operationalized"[2] in the design and implementation phases to ensure they are verifiable [Sommerville, 1995]. Hence, we advocate that dealing with NFR *operationalizations* instead of abstract NFRs is more adequate to make the mapping from crosscutting requirements and to elaborate the composition of these requirements with non-crosscutting ones.

It is worth mentioning that the NFR Framework [Mylopoulos et al., 1992; Chung et al., 2000] provides, among other features, a way to model the *operationalization(s)* of abstract non-functional requirements. In this context, we propose an adaptation of the NFR

[1] In this paper, *to affect* denotes how a crosscutting concern is related with other artifact, i.e., it means to say that the crosscutting concern need to be applied in some point of another artifact's implementation.

[2] To "operationalize" a requirement means providing more concrete and precise mechanisms (e.g. operations, processes, data representations, constraints) to solve a problem [Chung et al., 2000].

Framework in order to that framework can be used in the Aspect-Oriented Requirements Engineering (AORE) process. The objectives of this work are: (i) to improve the mapping and composition of crosscutting non-functional requirements, so that they can reflect and contribute to the modeling of aspects at later development stages; (ii) to indicate that, with few adaptations, existing techniques, like NFR Framework, can be used in association with the Aspect-Oriented Software Development.

This work is organized as follows. In Section 2 we briefly present the background of our approach, describing the main concepts used in Aspect-Oriented Software Development (Section 2.1) and also providing a review the NFR Framework (Section 2.2). In Section 3 we review related works. Section 4, in turn, presents the proposed adaptation of NFR Framework that will be illustrated by a case study in Section 5. Lastly, in Section 6 we present our conclusions and future works.

2. Background

This section presents the bases for our proposal by defining some key concepts used in Aspect-Oriented Software Development and by providing a brief overview of the NFR Framework.

2.1 Concepts used in Aspect-Oriented Software Development

The research in Aspect-Oriented Software Development is still at the beginning and there is no consensus about key concepts commonly used. For this reason, it is important to determine the ontology considered in this work. It is the following:

- Concern: vague declaration, generally corresponding to high-level strategic goal for the system being developed [Kotonya and Sommerville, 1998];
- Crosscutting concern: feature that in common paradigms cannot be cleanly encapsulated in one development artifact and hence it is spread throughout or tangled with other(s) entities [Rashid, 2001].
- Component: represents a concern that in common paradigms can be cleanly encapsulated in an entity (i.e. class, method, procedure, API). Components tend to be units of the system's functional decomposition, such as image filters, bank accounts, data converter and Graphical User Interface (GUI) units [Kiczales et al., 1997];
- Aspect: abstraction that encapsulates the specification of a crosscutting concern and where the match points and the composition rules of a crosscutting concern are defined;
- Match Point: is where the crosscutting concern should join its behaviour with the components it cuts across [Brito and Moreira, 2003];
- Composition Rule: expresses the sequential order in which each aspect must be composed with other(s) component(s), i.e., a composition rule specifies how a crosscutting concern needs to be applied in the match point. To make this composition, three operators are provided [Moreira et al., 2002]:
    i. Overlap: indicates that the aspect is applied *before* or *after* the component(s) it transverses.

ii. Override: indicates that the aspect superposes the component it transverses. This means that the behaviour described by the aspect substitutes the behaviour defined by the component.

iii. Wrap: the aspect "encapsulates" the component it transverses. This means that the behaviour described by the component(s) affected are enveloped by the behaviour described by the aspect.

2.2 Review of the NFR Framework

Non-functional requirements 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) [Kotonya and Sommerville, 1998]

The main objective of the NFR Framework [Mylopoulos et al., 1992; Chung et al., 2000] is to represent, organize and analyze non-functional requirements (NFRs). This framework is process-oriented in the sense of providing techniques for justifying design decisions during the software development process. It is goal-driven since it treats non-functional requirements as goals to be achieved. However, different from traditional goal-oriented approaches [Dardenne et al., 1993; Anton, 1996], this framework uses the notion of *softgoal*, which represents a goal that has no clear-cut criteria to determine whether they've been satisfied or not. A *softgoal* is considered *satisficed*[3] when there is sufficient positive evidence and little negative evidence against it [Mylopoulos et al., 2001].

The NFR Framework deals with the following key concepts:

- *Softgoal*: basic unit for representing non-functional requirements. There are three kinds of *softgoals*: NFR *softgoals* (or NFRs), *operationalizing softgoals* and claim *softgoals*. The first one represents high-level non-functional requirements to be *satisficed*. *Operationalizing softgoals* are possible solutions (operations, processes, data representations, etc.) or design alternatives which help to achieve the NFR *softgoal*. Lastly, *claim softgoals* justify the rationale and explain the context for a *softgoal* or interdependency link. Each *softgoal* has an associated *NFR type* and one or more topics to indicate, respectively, the meaning and the information item of the *softgoal* (e.g. Security [CardData], Authenticate [Account]). In the case of *claim softgoals*, the type is always `Claim` and the topic is a statement. Figure 1 presents the *softgoals* graphical representations adopted by NFR Framework.

Figure 1. *Softgoals* Graphical Representations

- *Interdependencies:* indicate refinements of *softgoals* and the contributions of offspring *softgoals* towards the achievement of its parent. There are basically two types of contributions describing how the offspring contributes to *satisfice* its

[3] A *softgoal* rarely can be completely satisfied. From here on, we will use the term *to satisfice* [Chung et al., 2000] to indicate that the goal satisfying is accomplished within acceptable limits.

parent. The first one decomposes a *softgoal* in a group of offspring by means of AND/OR contribution. The other type of contribution relates a single offspring to a parent and it can assume the following values: *surely negative* ("--" or BREAK), *surely positive* ("++" or MAKE), *partially negative* ("-" or HURT) and *partially positive* ("+" or HELP). This last type of contribution can be related to different *softgoals* hierarchies (*implicit interdependency or correlation*). Figure 2 shows the graphical representations of interdependencies adopted by NFR Framework;

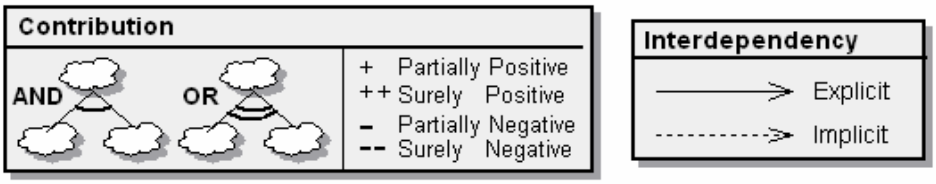

Figure 2. Interdependencies Graphical Representations

- *Softgoal Interdependency Graph (SIG)*: graph where *softgoals* and their interdependencies are represented. An example of a SIG for performance NFR in a credit card system is illustrated in Figure 3.
- *Catalogues:* group an organized body of design knowledge about NFRs (types, development techniques and correlations among *operationalizing* and NFR *softgoals*) that can be used in different application domains to compose the SIG.

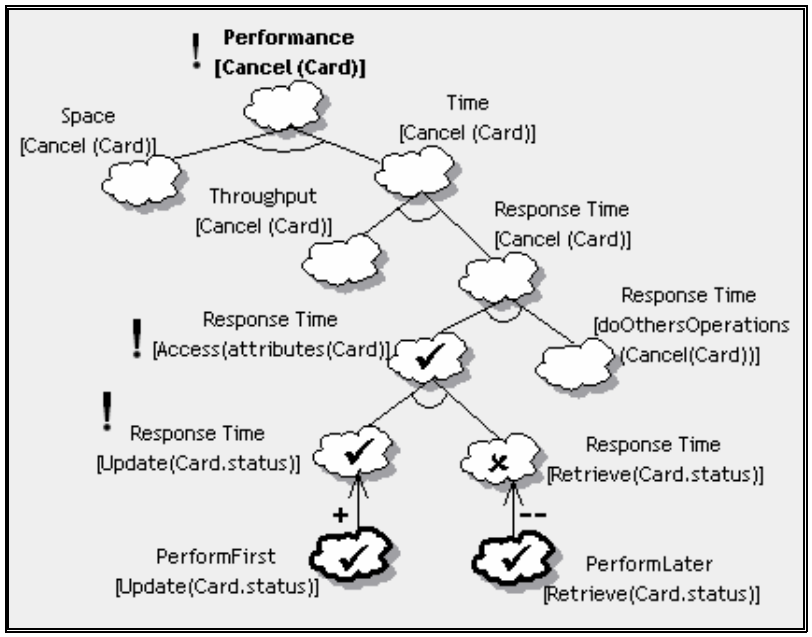

Figure 3. An Example of a SIG (adapted from [Chung et al., 2000**])**

The process of dealing with the NFR Framework (see Figure 4) starts with an identification of functional requirements and high-level non-functional requirements that the system should meet. Non-functional requirements should be represented as NFRs *softgoals* in the top of the SIG and they should be iteratively refined into more specific ones. At some point, when the NFRs *softgoals* have been sufficiently refined, it will be possible to *operationalize* these non-functional requirements and then choose specific solutions for the system. During refinement and *operationalization* steps, contributions and

possible conflicts should be established, defining the impact of *softgoals* to each other and identifying priorities (indicated by "!" or "!!").

An important consideration of the NFR Framework is that design decisions should be supported by well-justified arguments (design rationale) by means of *claim softgoals*.

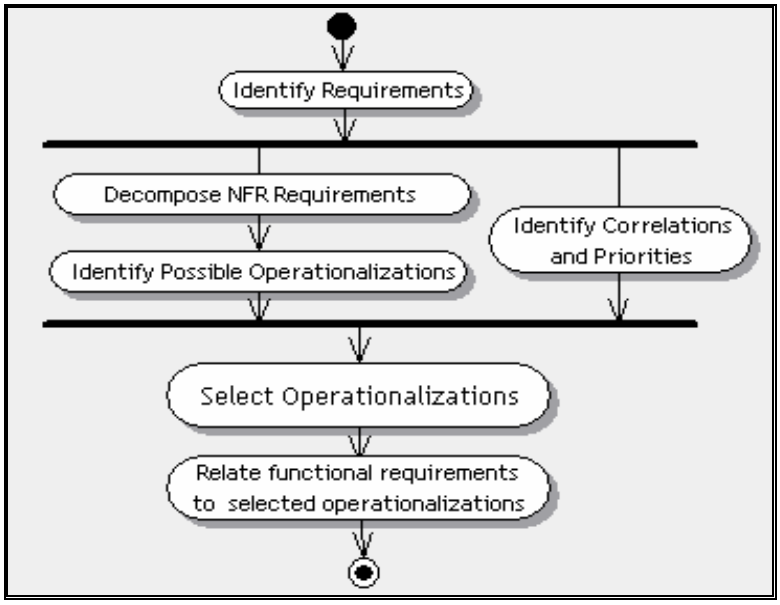

Figure 4. Process of NFR Framework (adapted from [Chung and Nixon, 1995] and [Chung et al., 2000])

Last but not least, it is possible to relate graphically in a SIG functional requirements to design specification of NFRs *operationalizations*. In order to do that, first it is necessary to link the chosen *operationalizations* to a description of the target system (represented in a rectangle) and then link this description to the functional requirements (represented in an oval). This mechanism is generally little explored since the most part of requirements techniques handle functional and non-functional requirements separately. The graphical representation of the components used in this step is exhibited in Figure 5.

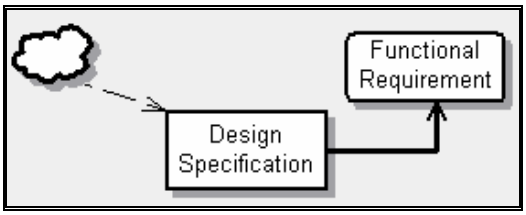

Figure 5. Relating Decisions to Functional Requirements

3. Related Work

One of the first publications that took Aspect-Oriented Programming into consideration to Requirements Engineering was [Grundy, 1999]. In that work, it was presented an Aspect-Oriented Component Engineering methodology in which after analyzing the system requirements, aspects are identified for each component. Those aspects determine the provided/required services by the component, and thus allow a better understanding and reasoning about component data, functionality, constraints and inter-relationships.

Later, Rashid et al. (2002) proposed a generic model for Aspect-Oriented Requirements Engineering (AORE). It is composed of the following activities: (i) identify and specify concerns and requirements; (ii) identify candidate aspects; (iii) specify and prioritize aspects; and (iv) specify aspect dimensions, i.e., determine the aspect influence on later development stages and identify its mapping onto a function, decision or aspect. The objective of that model was to accomplish the separation of crosscutting properties since the early stages of the development process in order to identify the mapping and influence of requirement level aspects onto artifacts at later development stages.

A refinement of the generic AORE model has been presented in [Rashid et al., 2003], including two new activities: aspect composition and conflict handling (its general diagram is represented in Figure 6). This new AORE model intends to compose (by means of composition rules) and modularize crosscutting concerns. The main argument of the authors is that the modularization of crosscutting concerns makes it possible to establish initial trade-offs between the candidate aspects. Therefore, it will be possible to give a better support for negotiation and subsequent decision-making among stakeholders, as well as facilitate the analysis of the impact of the crosscutting concerns on the artifacts produced at the next stages of the development process.

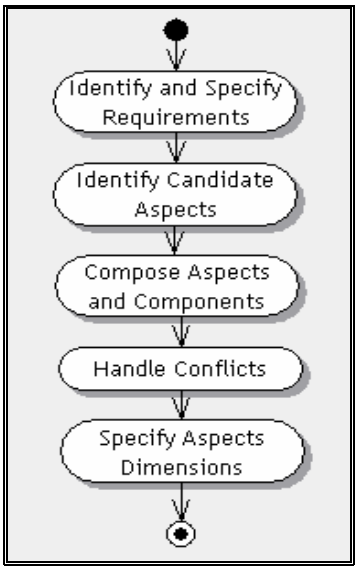

Figure 6. AORE Generic Model [Rashid et al., 2003**]**

In [Moreira et al., 2002] a simplified model is presented to support the general AORE process described in [Rashid et al., 2002]. The functional requirements are represented using UML diagrams and the quality attributes are described through templates. The composition of quality attributes with the functional requirements is accomplished using extensions of the use case and sequence diagrams.

An extension of the UML notation adopted by [Moreira et al., 2002], including the composition rule operators described in Section 2.1, was proposed in [Araujo et al., 2002] to make the composition of crosscutting quality attributes with functional requirements. Moreover, crosscutting concerns are specified using templates and an activity responsible for identifying and resolving conflicts is added to the process.

The main contributions given by [Brito and Moreira, 2003] were: (i) introduction of the match point concept and (ii) use of composition rules. That work presents an extension

of the process proposed in [Moreira et al., 2002], giving emphasis for the composition of functional requirements and candidate aspects.

We believe that it is more adequate to deal with NFR *operationalizations* in the context of Aspect-Oriented Requirements Engineering because they better reflect how the crosscutting concern will be implemented and therefore improving the composition and the mapping of crosscutting requirements onto artifacts at later development stages. Nevertheless, those previous AORE models proposed in literature [Rashid et al., 2002; Araujo et al., 2002; Brito and Moreira, 2003; Rashid et al., 2003] fail to address the issue of non-functional requirements, especially because they deal with abstract NFRs instead of NFR *operationalizations*.

4. Adaptation of NFR Framework

Our work modifies and extends the NFR Framework to Aspect-Oriented Requirements Engineering (AORE) in order to improve the composition and the mapping of crosscutting non-functional requirements onto artifacts at later development stages. We propose a novel approach, based in AORE generic models [Rashid et al., 2002; Rashid et al., 2003] (see Table 1), but we stress the following differences:

(i) We explicitly deal with NFR *operationalizations* in the mapping and composition activities instead of abstract declarations of NFRs;

(ii) We consider that each NFR *softgoal* is a concern;

(iii) Although there can be functional crosscutting concerns, in this paper crosscutting concerns will be limited to non-functional ones;

(iv) We decided that the activity *Identify Crosscutting Concerns (or Identify Candidate Aspects)*, presents in the most part of previous AORE models, is not necessary to be performed in the context of non-functional concerns. The reason for this decision is that non-functional concerns are naturally crosscutting, since they place restrictions on how the user requirements are to be met and, thus, they are always tied up with functional requirements [Cysneiros et al., 2001];

(v) Since aspects are only identified after the activity *Specifying the Mapping and Influence (Specify Aspects Dimensions)* [Rashid et al., 2002; Rashid et al., 2003], we recommend that, different from previous approaches, the activity of aspects composition to be performed after the activity *Analyze the Mapping*;

(vi) NFR *operationalization* results from typical crosscutting concerns, hence if we were to map a NFR *operationalization* onto a function or a procedure (as proposed by [Rashid et al., 2002; Rashid et al., 2003]), the NFR *operationalization* would be probably spread and/or tangled with others components at later development stages. Therefore, to preserve the separation of concerns principle we advocate that the NFR *operationalizations* should be mapped or onto an architectural decision or onto an aspect;

(vii) It is not necessary to include an activity responsible for handling conflicts because the NFR Framework has already dealt with that in the decisions evaluation procedure by means of interdependencies, correlations and priorities.

Table 1 - Correlation between the activities of the AORE generic model and the activities of the NFR Framework adaptation

AORE GENERIC MODEL	NFR FRAMEWORK ADAPTATION
Identify and specify requirements	Identify requirements
	Decompose NFR requirements
	Identify possible *operationalizations*
	Select *operationalizations*
Identify candidate aspects	- - Not necessary - -
Compose aspects and components	Compose identified aspects with functional requirements
Handle conflicts	Identify correlations and priorities
Specify aspects dimensions	Analyze the mapping of NFR *operationalizations*

As illustrated in Figure 7, the first five activities of the proposed adaptation of NFR Framework correspond to the same first five steps of NFR Framework [Mylopoulos et al., 1992; Chung et al., 2000]. After accomplishing them, we have the selected *operationalizations* that meet the non-functional concerns initially identified.

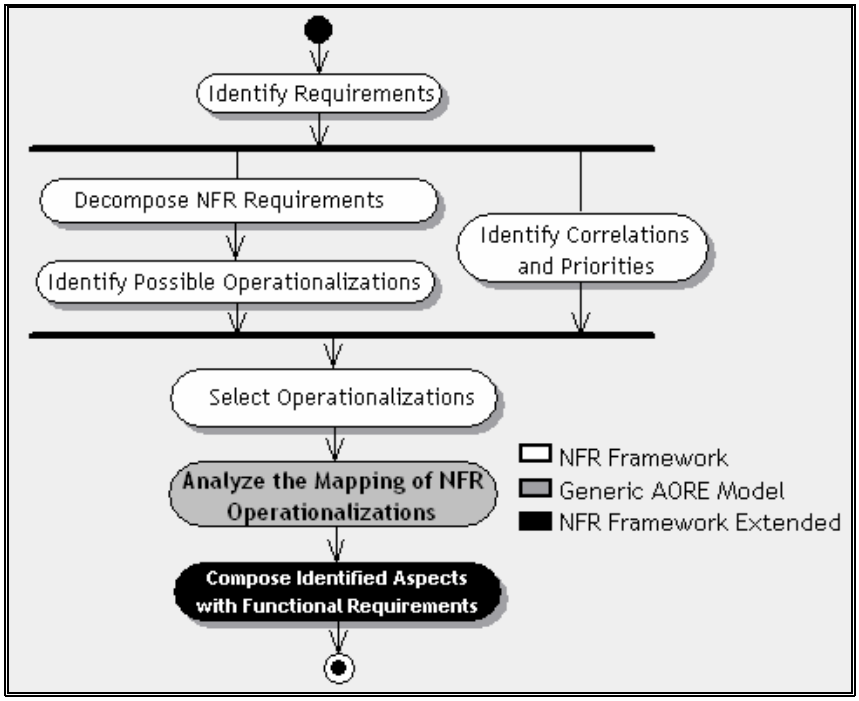

Figure 7. Proposed adaptation of NFR Framework

The next activity, originated from the generic AORE Model [Rashid et al., 2002], is responsible for analyzing what is the mapping of these *operationalizations* onto artifacts at later development stages: architectural decision or aspect. If an *operationalization* is related to the manner how the components are organized in the architecture, then it is mapped onto an architectural decision (e.g., the *operationalizing softgoal* "Duplicate the server" to meet the concern of availability); if not, they are mapped onto an aspect.

Last but not least, the composition of the identified aspects with the functional requirements is performed. This activity is an extension of the NFR Framework activity

"Relate functional requirements to selected *operationalizations*", but with a difference: the composition rule operators *overlap, override* and *wrap,* described before (Section 2.2), should be considered.

5. Applying the Approach to Case Study

We apply our approach to an Internet Banking System since non-functional requirements are determinant for the success of this kind of system [Patricio et al., 2003]. The main objective of an Internet Banking System is to allow bank clients to perform banking transactions through the Internet. In the sequel, we follow the steps prescribed in Section 4.

IDENTIFY REQUIREMENTS

Having as input existing system information, stakeholder needs, organizational patterns, regulations and domain information, and using any requirements elicitation technique, the developer can specify the following high-level requirements for this kind of system[4]:

- Functional Requirements: to allow (i) query transactions (account balance and account statement) and (ii) financial transactions (transfers, bill payments, etc.);
- Non-Functional Requirements: security, availability, user-friendless;

DECOMPOSE NFR REQUIREMENTS

One important concern when building information systems to be used on the Internet is information *security*, i.e., protecting information against unauthorized access. According to NFR catalogues [Chung et al., 2000] and domain information, the developer can decompose this concern in three others ones: *Confidentiality, Integrity* and *Availability*.

Confidentiality is a priority concern since the bank institutions have obligation to guard client information against unauthorized divulgation. Avoiding interruption of service (Availability) is other important concern because it contributes considerably for the client satisfaction. Since site intrusions are common on Internet, Integrity, i.e., guarding transactions against unauthorized update or falsification, is also an important concern. Integrity can be subdivided in Completeness and Accuracy. The former means wholeness of the data being maintained by the system; the latter is related to the correspondence between values in the system and what they are supposed to represent.

Those decompositions can be visualized in Figure 8.

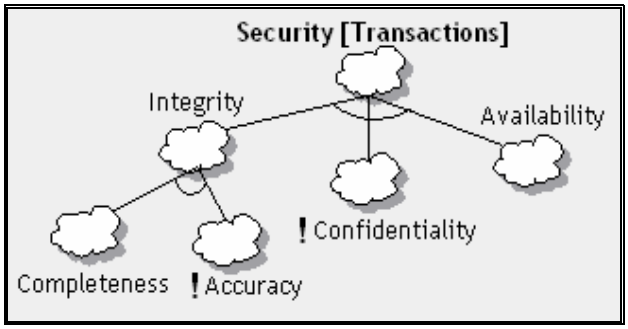

Figure 8. Security Decomposition

[4] We reduced the number of requirements to simplify the case study.

IDENTIFY POSSIBLE OPERATIONALIZATIONS

In this step, the developer should analyze possible *operationalizations* for each one of the NFR *softgoal* offsprings (see Figure 9).

Beginning by 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 softgoal*s: *Identification* and *Authentication* (Single for query transactions or Multiple for financial transaction). The *Internet Password Request satisfices* the *SingleAuthentication softgoal* and in conjunction with *OtherAuthentication softgoal satisfices MultiAuthentication softgoal*. Lastly, the *OtherAuthentication softgoal* can be *satisficed* either by *PersonalDataValidation* or else by *AdditionalPasswordRequest*. However, the *OtherAuthentication satisficing* contributes negatively for *user-friendly access* concern.

Analyzing the *Accuracy* concern, two possible alternatives can help to decrease the risk of frauds: (i) to limit the value of financial transactions and (ii) to install a firewall to protect the database server of badly-intentioned intruders. There is also a positive correlation between the *Authentication operationalization* and the *Accuracy softgoal*. In similar fashion, *Data Encryption operationalization* helps to achieve the *Accuracy softgoal*.

At last, possible solutions that contribute positively to *satisficing* the *Availability softgoal* are *Duplication of the Server* and *Mirroring the Database*.

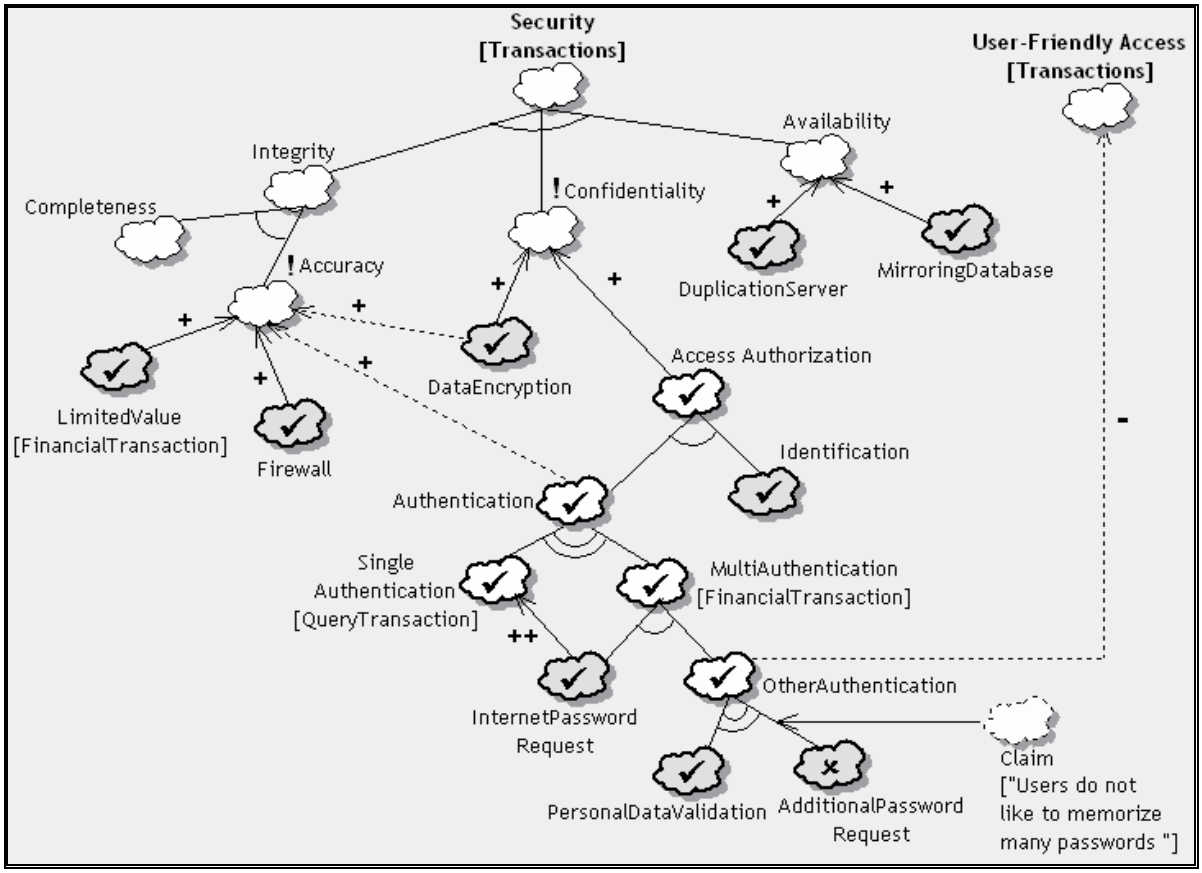

Figure 9. Representation of *Operationalizing Softgoal*s for Security Concern

In Figure 9 all those *operationalizing softgoal*s (in a gray color) can be visualized.

SELECT *OPERATIONALIZATIONS*

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 (✓) or rejecting (✗) each of them.

In our case study, the only rejected *operationalizing softgoal* was *Additional Password Request.* The reason for that is represented in a claim *softgoal* being related to the client difficulty to memorize many passwords.

The decisions and their impact in the sense of *satisficing* or not the parent *softgoal*s are presented in Figure 9.

ANALYZE THE MAPPING OF NFR *OPERATIONALIZATIONS*

So far, graphs started with top abstract NFRs and they resulted in *operationalizations* being selected. In this current activity we analyze the mapping of each selected *operationalizing softgoal* with respect to artifacts to be generated in later stages. Table 2 shows the outcome of that analysis.

Table 2. Mapping of NFR *Operationalizations*

NFR CONCERN		NFR *OPERATIONALIZATION*	MAPPING
SECURITY	ACCURACY	*Limited Value*	Aspect
		Firewall	Architectural Decision
	CONFIDENTIALITY	*Data Encryption*	Aspect
		Identification	Aspect
		Internet Password Request	Aspect
		Personal Data Validation	Aspect
	AVAILABILITY	*Duplication Server*	Architectural Decision
		Mirroring Database	Architectural Decision

It is important to emphasize that in previous AORE works this mapping is done from abstract non-functional requirements or NFR concerns (first column) instead of NFR *operationalizations* (second column). For instance, they make the mapping from a security concern onto an aspect [Rashid et al., 2002; Rashid et al., 2003]. However, how we can perceive in Table 2 all those *operationalizations* are related to a security concern and, even so, there are some *operationalizations* mapped onto aspects and others mapped onto architectural decisions. Furthermore, considering that the objective of the mapping is to perform the aspect analysis earlier, our kind of mapping from *operationalizations* better reflects how the aspects will be treated at later development stages.

Therefore, as shown in Table 2, our mapping is richer than if we were dealing only with NFR concerns because it better reflects how the design and implementation of these concerns will be addressed.

COMPOSE IDENTIFIED ASPECTS WITH FUNCTIONAL REQUIREMENTS

This is the last activity of our proposal. Now, we graphically relate the identified aspects to functional requirements they affect. In doing so we use the composition rule operators previously described (Section 2.1).

Figure 10 shows how the chosen *operationalizations* are linked to descriptions of design specifications as well as the functional requirements affected by them.

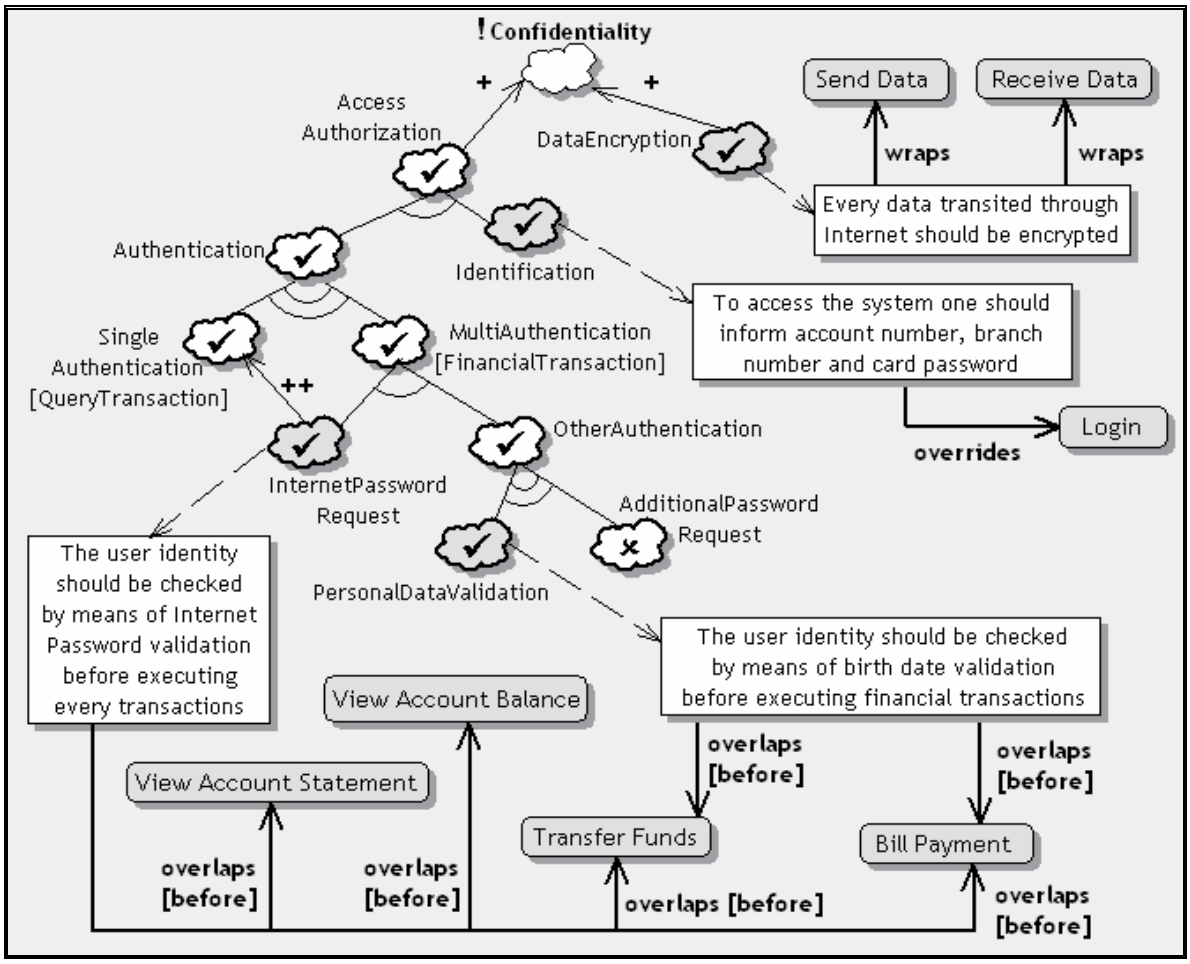

Figure 10. Composing Aspects Identified with Functional Requirements Using Composition Rule Operators

The novelty here is the use of the operators: overlaps, overrides and wraps. They are required in order to introduce the aspect semantics in the design decision links. We can perceive in Figure 10 that, compared with previous approaches, this kind of composition improves the AORE process because it reflects the real modeling and implementation of non-functional aspects at later development stages.

For instance, making an analogy with the ontology presented in Section 2.1, we can say that the *Internet Password Request operationalization* (non-functional aspect) affects the functional requirements (components) *View Account Balance, View Account Statement, Transfer Funds and Bill Payment;* and that aspect should be applied before (composition rule) executing every transactions (match point). In similar fashion, the *Data Encryption operationaliozation* (non-functional aspect) affects the functional requirements (components) *Send Data* and *Receive Data*; and that aspect should change the data

(composition rule) during the execution of those functions (match point) by means of cryptography.

Unfortunately, the choice about which operator will be used and which functional requirement will be affected is based in the software engineer experience.

6. Conclusion and Future Work

Aspect-Oriented Paradigm (AOP) is an evolution, not a revolution, on previous software development paradigms. For that reason, it is natural the attempt to adapt existing software development methodologies and techniques to include AOP concepts. In this context, this paper has proposed an adaptation of NFR Framework [Mylopoulos et al., 1992; Chung et al., 2000] to Aspect-Oriented Requirements Engineering [Rashid et al., 2002; Araujo et al., 2002; Rashid et al., 2003; Brito and Moreira, 2003] in order to improve the mapping and the composition of crosscutting requirements onto artifacts at later development stages.

It was necessary to make two adaptations to the NFR Framework process: (i) to include an activity responsible for analyzing the mapping from the *operationalizations* onto later artifacts; and (ii) to modify the activity that links functional requirements to design decisions in order to put aspects semantics in that relationship by means of the composition rule operators.

Our proposal uses non-functional requirements (NFR) *operationalizations* [Chung et al., 2000], instead of abstract declarations of NFRs, in the mapping and composition of crosscutting NFRs. The presented case study indicates that our approach provides richer mapping and composition than previous approaches since it better reflects how crosscutting concerns will be manipulated in later stages. Of course the price to be paid is the inherent complexity of our approach because besides the NFR Framework notation, we propose the inclusion of composition rule operators in its graph so that this model can be used in Aspect-Oriented Requirements Engineering.

This work focus on non-functional product requirements instead of non-functional process requirements since there are not evidences in the literature that the last one can be encapsulated on an aspect. Furthermore, we do not deal with non-functional requirements like persistence and distribution because they are very dependent on implementation characteristics; for that reason we believe that they should be better specified in the aspect-oriented design.

Our future work will focus on evaluating the proposed approach and on improving the composition between crosscutting requirements and non-crosscutting ones.

7. References

Aksit, M.; Tekinerdogan, B. and Bergmans, L. (2001), "The Six Concerns for Separation of Concerns", in Proceedings of ECOOP 2001 Workshop on Advanced Separation of Concerns, Budapest, Hungary, June 18-22.

Anton, A. (1996) “Goal-based Requirements Analysis,” Proc. 2nd IEEE Int’l Conf. Requirements Engineering, CS Press, Los Alamitos, Calif., pp. 136–144.

Araújo, J.; Moreira, A.; Brito, I. and Rashid, A. (2002) "Aspect-Oriented Requirements with UML", Workshop: Aspect-oriented Modeling with UML, UML 2002, Dresden, Germany.

Baniassad, E.; Murphy, G.; Schwanninger, C. and Kircher, M. (2002) ”Managing

Crosscutting Concerns During Software Evolution Tasks: an Inquisitive Study", Proceedings of the 1st international conference on Aspect-oriented software development, April 22-26, Enschede, The Netherlands

Bergmans, L. and Aksit, M. (2001) "Composing Crosscutting Concerns Using Composition Filters". Commun. ACM, 44(10): 51–57, Oct.

Brito, I. and Moreira, A. (2003) "Towards a Composition Process for Aspect-Oriented Requirements". Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, March 17 - Boston, USA.

Chung, L. and Nixon, B. (1995) "Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach". In Proceedings of the IEEE 17th International Conference on Software Engineering (ICSE), Seattle, April 24-28, pp. 25-37.

Chung, L; Nixon, B.; Yu, E. and Mylopoulos, J. (2000) "Non-Functional Requirements in Software Engineering",Boston: Kluwer Academic Publishers.ISBN 0-7923-8666-3.

Clarke, S.; Harrison, W.; Ossher, H. and Tart, P. (1999) "Towards Improved Alignment of Requirements, Design, and Code". Conf. on Object-Oriented Programming, Systems, Languages, and Applications, Denver, Colorado. ACM SIGPLAN Notices, v. 34, n. 10, pp. 325-339.

Cysneiros, L.; Leite, J. and Neto, J. (2001) "A Framework for Integrating Non-Functional Requirements into Conceptual Models". Requirements Engineering Journal – Vol 6, Issue 2 Apr., pp: 97-115.

Dardenne, A; Lamsweerde, A and Fickas, S. (1993) "Goal-Directed Requirements Acquisition", Science of Computer Programming, Vol. 20, 3-50.

Feng, L.; Marcus, A. and Schaffer, K. (2001) "An Overview of Aspect Oriented Programming", Term Report, Kent State University, Department of Computer Science.

Ghezzi, C.; Jazayeri, M. and Mandrioli, D. (1991) "Fundamentals of Software Engineering". Prentice Hall, ISBN0-13-820432-2.

Grundy, J. (1999) "Aspect-Oriented Requirements Engineering for Component-based Software Systems", 4th IEEE International Symposium on RE, IEEE Computer Society Press, pp. 84-91.

Harrison, W. and Ossher, H. (1993) "Subject-Oriented Programming (a Critique of Pure Objects). Conf. on Object Oriented Programming: Systems, Languages, and Applications.

Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J.-M. and Irwin, J. (1997) "Aspect-Oriented Programming". In Proceedings of ECOOP '97, Springer-Verlag.

Kotonya, G. and Sommerville, I. (1998) "Requirements Engineering: Processes and Techniques". Wiley, ISBN 0-471-97208-8.

Moreira, A.; Araújo, J. and Brito, I. (2002) "Crosscutting Quality Attributes for Requirements Engineering", 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002), ACM Press, Italy, July.

Mylopoulos, J.; Chung, L. and Nixon, B. (1992) "Representing and Using Non-Functional Requirements: A Process-Oriented Approach". IEEE Transactions on Software Engineering, Vol. 18, No. 6, June, pp. 483-497.

Mylopoulos, J.; Chung, L.; Liao, S.; Wang, H. and Yu, E. (2001) “Exploring Alternatives during Requirements Analysis”. IEEE Software Jan/Feb, pp. 2-6.

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.

Patrício, L., Falcão e Cunha, J. and Fisk, R. (2003) “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.

Rashid, A. (2001) “Editorial Aspect -Oriented and Component-Based Software Engineering”. IEE Proc. Software: Special Issue on Aspect-Oriented and Component-Based Software Engineering, 148(3)(June).

Rashid, A. Moreira, A. and Araujo, J. (2003). “Modularisation and Composition of Aspectual Requirements”. 2nd International Conference on Aspect-Oriented Software Development, ACM, pp. 11-20.

Rashid, A.; Sawyer, P.; Moreira, A. and Araújo, J. (2002) "Early Aspects: a Model for Aspect-Oriented Requirements Engineering", IEEE Joint Conference on Requirements Engineering, Essen, Germany, September.

Sommerville, I. (1995). “Software Engineering”, 5th Ed., Addison-Wesley, pg. 132.