Modularization of Crosscutting Concerns in Requirements Engineering

Abdelkrim Amirat[1, 2], Mohamed Laskri[1], and Tahar Khammaci[2]
[1] Laboratoire de Recherche en Informatique, Université de Annaba, Algeria
[2]Laboratoire LINA CNRS FRE 2729, Université de Nantes 2, France

Abstract: *In spite of the generated benefits, Object-Oriented (OO) paradigm seems reaching its limits, regarding complexity reduction of current systems. In this context, the Aspect Oriented (AO) comes up as an alternative to reduce software development complexity while keeping OO advantages. Needs for investigating methodologies of AO Software Development have emerged a long with AO. As an example, Early Aspect (EA) aims to identify aspects on the early stages of software development, such as domain analysis requirements specification and architectural design. Being one of the newest software engineering paradigms, AO emphasizes that new studies and experiments should be carefully carried out, in order to establish improved methods, techniques and tools applicable to this new way of development. In this paper, we discuss a sequence of systematic activities toward an early consideration of specifying and separating crosscutting Functional Requirements (FRs) and Non-Functional Requirements (NFRs) by the adoption of use-cases to model systems. This approach would make it possible to identify and resolve conflicts between requirements earlier in the development cycle and can promote traceability of broadly scoped properties throughout system development, maintenance and evolution.*

Keywords: *Early aspects, requirements engineering, decomposition, composition, methodologies.*

Received May 30, 2006; accepted November 22, 2006

1. Introduction

Despite the success of Object-Orientation (OO) in the effort to achieve separation of concerns (requirements), certain properties in OO systems cannot be directly mapped from the problem domain to the solution space, and thus they cannot be localised in single modular unit [7]. This is due to the fact that functional decomposition is performed along the notion of class. A conflict tends to rise when we map an n-dimensional requirements space to a single dimensional design and implementation space while building development artefacts. This conflict constitutes the source of CrossCutting (CC) which imposes two symptoms on implementation: (1) code tangling and (2) code scattering. In OO development CC does not allow the benefits of Object-Oriented Programming (OOP) to be fully utilized. Developers are thus faced with a number of implications including poor traceability of requirements, strong coupling between functional components, low cohesion of modules, low degree of code reusability, and low productivity. As a consequence to the above, the quality of software is negatively affected.

There have been many approaches to Aspect-Oriented Requirements Engineering (AORE). Each approach attempts to capture and address a significant issue or issues relating to crosscutting in requirements engineering by providing a second axis of decomposition that enables separation of core functionality from crosscutting requirements [11]. In [1, 2], we discussed an AOSD model that constitutes of a sequence of systematic activities towards an early consideration of identifying, specifying and separating crosscutting non functional requirements starting from requirements elicitation.

Throughout the development process, stakeholders are in need to verify that they managed capturing and specifying all related crosscutting requirements properly. To achieve this target, we choose to extend our AOSD model in this paper by proposing a set of steps to deal with functional and non functional requirements simultaneously. This paper offers the following contributions:

- It proposes a new approach to identify, separate and compose requirements starting from early requirements elicitation to implementation phase.
- It provides a new mechanism to compose requirements that assist in integrating the captured main requirements with the crosscutting requirements.

The rest of this paper is organized as follows: Section 2 introduces some background information related to this research. Section 3 briefly summarizes the motivation and the proposal. In section 4, the main ideas of the paper are discussed. Section 5 presents related works on Early Aspects (EA), and in section 6 we conclude and discuss recommendations for future research.

2. Background

The contemporary non-AORE approaches have been developed to primarily deal with one type of concerns. For instance, PREview [15] and NFR [6] have underlined the importance of non-functional concerns and proposed means to ensure their fulfilment in a system. Problem frames [8] and use cases [9], on the other hand, have focused on ensuring the required functionality of a system. Recently, AORE approaches try to propagate the idea that all types of concerns are equally important and should be treated consistently, and non-discriminatively.

Moreover, the CC concerns have not been treated as separate units of modularity. For instance, the issues related to security in PREview [15] would be scattered across all viewpoints, as each viewpoint will have to specify the influence of security on it. AORE allow a broad crosscutting influence for both functional and non functional requirements and their modularisation. The issue of requirements elevel composition has not been extensively investigated before AO. Composability *(Weaving Process)* [4] the support for combining individual requirements into coarser-grained requirements is the central notion of AORE. Using AO terminology, this support should include:

- *JoinPoint model:* a well defined JoinPoint Model (Interaction Points: a set of points in the computational flow of the program in AspectJ[1]) exposes structured points through which requirements can be composed.
- *Composition semantics:* the composition semantics provides systematic meaning to the composition process.

Composability allows not only reviewing the requirements in their entirety, but also the detection of potential conflicts very early on in order to either take corrective measures or make appropriate decisions for the next development step. The composed requirements also become valuable sources of validation for the complete system.

3. Related Works

Recently, there has been growing interest in propagating the aspect paradigm to the earlier activities of the software development life cycle. A number of approaches to aspect-oriented design have been proposed.

In [18], the authors adopted model analysis to detect semantic conflicts between aspects. However, the approach is dedicated to serve the detection of direct conflicts only. Resolving conflicts is recommended through a process of correction and refinement of the model, which is not clearly investigated.

In [5] and [14], the composition of the concerns was defined as the last step of a proposed model for separation of concerns at requirements engineering using the formal method LOTOS. Resolving conflicts among concerns is recommended through negotiation with stakeholders which may not always be applicable except for developers. Defining the dominant concern at a matching point as recommended in this approach is not always applicable as well because of the dynamic behaviours of the system. In addition, it is not clear how to map the combination set defined in LOTOS to the next stages of the development.

In [3] and [13], composition of concerns is accomplished by extending UML models to integrate the candidate aspects to the functional behaviour. Although the composition process must be considered at the meta-level, these approaches only model certain NFRs in a way that is not necessarily applicable for other requirements.

In [12], the obliviousness property was adopted to model orthogonal aspects independently from each other and from the functional requirements. The usage of formal methods in these approaches (e.g., GAMMA, LOTOS, Time Temporal Logic) to specify the functional behaviour and the associated aspects helps to enable formal validation and facilitates a specification-driven design. On the other hand, the weaving process is not presented in a precise systematic way and it is limited to a specific type of requirements that can not necessarily be applicable for others.

4. Motivation and Proposed Approach

An effective software development approach must harmonise the need to build the functional behaviour of a system with the need to clearly model the associated Non Functional Properties (NFP). Most of the current approaches adopt the AOSD as an effective mechanism to handle NFPs at the early stage of the development process. This is mainly because the NFRs are considered as global properties of the system and they crosscut at different spots of it, thus they need to be treated within the context of AOSD which has been prompted as an approach to separate crosscutting concerns and improve the modularity in software system artefacts [6].

In our proposal we adopt use-case driven activities to model the system. We argue that use-cases tend to be more concrete in their representation of the system as they explicitly state series of interactions between actors and the system. Furthermore, their representations tend to be easy to map to the next phases in development. Use-cases are also widely used as part of the de facto standard of UML [10]. The different activities that define our approach are illustrated in Figure 1. In spite of the model sequence of activities, we emphasize the iterative and incremental nature of the development is implied even thought it is not explicitly captured in the diagram.

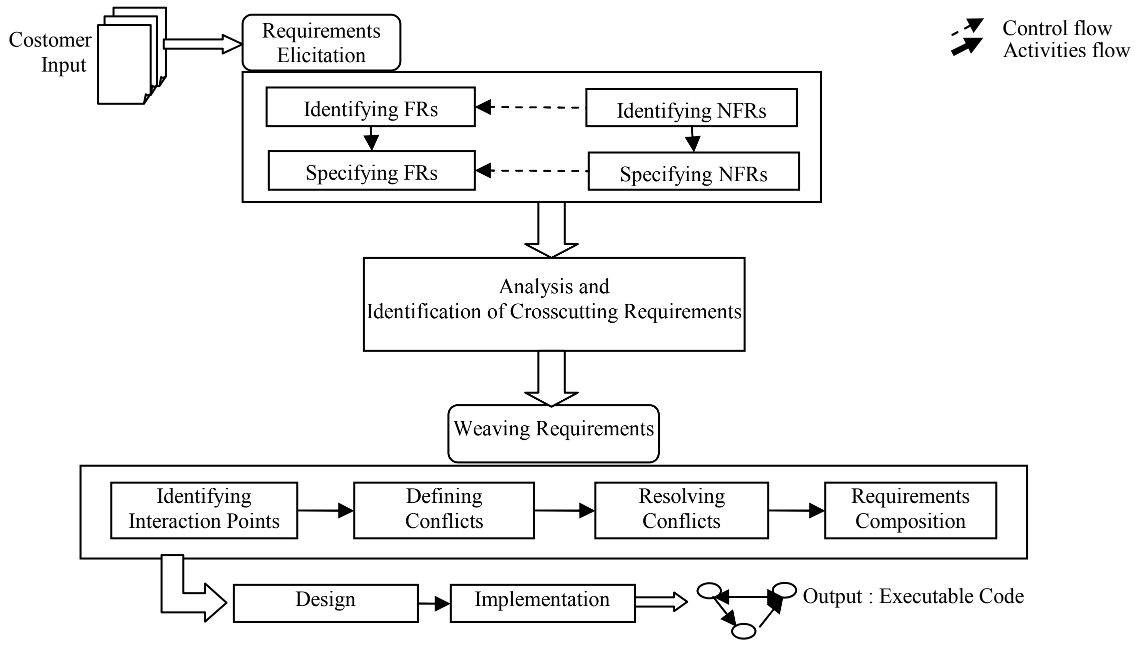

Figure 1. AOSD activities in the proposed approach.

5. The AOSD Approach

In this section we describe the different activities that define our proposed AO approach. The approach is defined in five phases: requirements elicitation, analysis and identification of CC requirements, weaving requirements, design and implementation. We use the term phase to describe a group of one or more activities within the AOSD approach. The phase is a mean to categorize activities based on the general target they tend to achieve.

5.1. Functional Requirements

By the end of the current activity, we illustrate with a graphical representation the interactions between the actors and the system in terms of a System Sequence Diagrams (SSD).

5.1.1. Identifying Functional Requirements

Functional requirements capture the intended behaviour of the system. This behaviour may be expressed as services, tasks or functions whose the system is required to perform. Those requirements can carried out using an existing requirements level separation of concerns mechanism such as view points [15], use cases [9], goals [6], or problem frames [8]. In our case we use use-case diagrams as starting point to summarize with graphical representation the high-level behaviour of the system: what the system does (*as a black box*), what lies outside the system and how it gets used. Identifying FRs is a process that involves discussions with stakeholders, reviewing proposals, building prototypes and arranging requirements elicitation meetings.

5.1.2. Specifying Functional Requirements

In this activity, we further refine the detailed functional behaviour of each use-case with textual description, graphical representation and formal specification. The outcome of this activity is the completion of a use-case description Table 1 which forms an extension to the fully-dressed format.

5.1.3. Identifying Crosscutting Functional Requirements

To identify the crosscutting nature of certain FRs we need to take into consideration the information contained in the row "*Related Use Cases*" in Table 1. If a use-case is repeated in multiple occurrences in the system functional descriptions, then it is a crosscutting, also we can find a use case depending on another one without being CC (i.e. the case of extends and includes relationship between use-cases).

5.2. Non-Functional Requirements

5.2.1. Identifying Non-Functional Requirements

Non functional requirements that are relevant to the problem domain are captured in parallel to the identification of FRs. Even though the elicitation of NFRs can be accomplished in a number of existing techniques, the most recognized technique is to use NFR catalogue [6] where each entry in the catalogue is cheeked whether it is applicable for the system or not.

[1]AspectJ^{TM} is an Aspect Oriented extension of Java developed by Kiczales [11] at Xerox Palo Alto Research Center, 2000.

5.2.2. Specifying Non-Functional Requirements

To specify non-functional requirements we adopt an extended version of our approach presented in [1, 2]. We propose the adoption of the matrix presented in Table 2 that relates the identified NFRs to the use cases they affect. In the case where an NFR (e.g., login) affects the system as a hole, the entire corresponding column must be check.

Table 1. Template to specify usec ases.

Use Case No.	**Unique to the Use-Case**
Name	The name of the use-case
Priority	Importance of the use-case
Actors	Primary and secondary actors
Precondition (Textual →Formal)	Description of the conditions to be satisfied before the use-case is executed
Main Scenario	A single and complete sequence of steps describing an interaction between a user and a system
Alternative Scenario	Extensions or alternative courses of the main scenario
Postcondition (Textual →Formal)	Description of the conditions to be satisfied after the use-case is executed
Related Use-Cases	Use-cases which the current use-case depend on

Table 2. Use-cases affected by NFRs.

	NFR$_1$	**NFR$_2$**	**….**	**NFR$_N$**
Use-Case$_1$	✓	✓		
Use-Case$_2$		✓		✓
….				
Use-Case$_n$	✓			

5.3. Weaving Requirements

The goal of this activity is to weave (i.e., compose) the functional requirements and non-functional requirements together. This is achieved in a series of four steps:

Step 1: Identifying the interaction points at which crosscutting requirements affect the system.
Step 2: Identifying possible conflicts among requirements at each interaction point.
Step 3: Resolving conflicts.
Step 4: Integrating requirements.

A. Identifying Interaction Points

Based on functional crosscutting and the correspondence between NFRs and use-cases which constitutes another form of crosscutting, we can identify interaction points in the system where crosscutting will manifest themselves. Otherwise the set of interaction points is defined by the sub set of the Use Cases (UC) affected by CC Requirements (CCR), as shown in Table 3.

Table 3. Interactions points in the system.

	CCR$_x$	**CCR$_y$**	**….**	**CCR$_z$**
Use-Case$_i$	✓			✓
Use-Case$_j$		✓		✓
….			….	
Use-Case$_k$	✓	✓	….	

B. Defining Conflicts

Rarely requirements manifest in isolation, and normally the provision of one crosscutting may affect the level of another. We refer to this mutual dependency as non-orthogonality [16, 17]. The dependency can be collaborative (positive) or damage (negative). We define function for mapping pairs of CCRs to values "+", "-", "?" or " ". The rules for assigning the signs to the pairs of CCRs are as follows:

$$F\,(CCR_i\,,\,CCR_j)\ \rightarrow \{\text{“+”}, \text{“-”}, \text{“?”}, \text{“ ”}\} \qquad (1)$$

The values "-", "+" or " " are assigned to the pair of CCRs originating from the set of NFRs that contribute respectively negatively, positively or do not interact at the same interaction point. The assignment is based on the expert's judgment of the developers. The "?" value indicates a lack of information on the contribution; this might be updated in later phases of the software development life-cycle, or a subsequent iteration.

We use Table 4 as symmetric matrix presentation of the previous mapping function. It is important to stress that any conflict defined at this step may not prove to be a real conflict when the system will be refined further during later phases.

Table 4. Aspect contribution matrix for the set of crosscutting requirements (CCR$_1$… CCR$_N$).

F:function	**CCR$_1$**	**CCR$_2$**	**….**	**CCR$_N$**
CCR$_1$		+		-
CCR$_2$	+			?
….				
CCR$_N$	-	?		

C. Resolving Conflicts

For each interaction point we analyse the set of crosscutting requirements and study the contribution among its elements. We are essentially interested in those elements "requirements" that have a mutual negative interaction. We solve conflict resolution by refining the set of the crosscutting requirements to eliminate the negative contribution, or assign a priority among these elements to determine the order of their execution.

D. Requirements Composition

In this step, we integrate requirements (FRs and NFRs) together to obtain the whole system, and we use UML diagrams at this high level of abstraction to model the composition. In the new structured use-case diagram, we use <<include>> stereotype for each NFR and have the set of initial crosscutting use-cases include the new ones.

6. Conclusion and Future Work

Tangling and scattering are symptoms that do not exclusively affect implementation, but they also propagate to early stages of the development process. Identifying and modelling crosscutting earlier has a great impact on the improving the general quality of the system and reducing complexity by (1) prompting understandability and reusability, (2) enhancing the process of detecting and removing defects, (3) reducing development time. In this paper, we discussed a sequence of systematic activities towards an early consideration of identifying, specifying and separating broadly scoped requirements that are traceable throughout system development process. We addressed both FRs and NFRs as candidate crosscutting requirements. To compose requirements, we provided a fine grained approach to define interaction points and relate them to the level of use-cases. Our approach makes it possible to early recognize and resolve conflicts within the activity of composing requirements. For future research, we plan to investigate how to formalize the specification of the NFRs and how to integrate them with formally specified FRs. We also plan to investigate how to formally resolve conflicts among requirements at interaction points with minimum contribution of stakeholders. Finally we work on a set of measurements that help in better designing strategies based on quantitative analysis and we plan to investigate how to formalize the traceability mechanism.

References

[1] Amirat A. and Laskri M., "Modular Implementation of Aspectual Requirements," *in Proceedings of the International Arab Conference on Information Technology (ACIT'05)*, Amman, Jordan, pp. 159-163, 2005.

[2] Amirat A., Meslati D., and Laskri M., "An Aspect-Oriented Approach in Early Requirements Engineering," *in Proceedings of the 4th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'06)*, Sharjah, UAE, pp. 224-227, 2006.

[3] Araujo J., Moreira A., Brito I., and Rashid A., "Aspect-Oriented Requirements with UML", *in Proceedings of the Workshop on Aspect Oriented Modelling with UML in Conjunction with 1st International Conference on Aspect-Oriented Software Development*, Enschede, Netherlands, 2002.

[4] Baniassad E., Clements P., Araújo J., Moreira A., Rashid A., and Tekinerdoğan B., "Discovring Early Aspects," *IEEE Software*, pp. 61-70, January/February 2006.

[5] Brito I. and Moreira A., "Integrating the NFR Framework in a RE Model," *in Workshop on Early Aspects, in conjunction with 3rd International Conference on Aspect Oriented Software Development*, Lancaster, UK, 2004.

[6] Chung L., Nixon B., Yu E., and Mylopoulos J., *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, 2000.

[7] Dijkstra E., *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[8] Jackson M., *Problem Frames: Analysing and Structuring Software Development Problems*, Addison-Wesley, 2001.

[9] Jacobson I., Chirsterson M., Jonsson P., and Overgaard G., *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992.

[10] Jacobson I., Booch G., and Rumbaugh J., *The Unified Software Development Process*, Addition-Wisley, 1999.

[11] Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J., and Griswold W., "An Overview of AspectJ," *in Proceedings of the ECOOP'2000*, Springer Verlag, 2000.

[12] Mousavi M. Russello G., Chaudron M., Reniers M., Basten T., Corsaro A., Shukla S., Gupta R., and Schmidt D., "Aspect+ GAMMA= AspectGAMMA: A Formal Framework for Aspect-Oriented Specification," *in Proceedings of the Workshop on Aspect-Oriented Modelling with UML in conjunction with 1st International Conference on Aspect-Oriented Software Development*, Enschede, Netherlands, 2002.

[13] Park D. and Kand S., "Design Phase Analysis of Software Performance Using Aspect-Oriented Programming," *5th Aspect Oriented Modelling Workshop in Conjunction with UML 2004*, Lisbon, Portugal, 2004.

[14] Rashid A., Moreira A., and Araujo J., "Modularisation and Composition of Aspectual Requirement," *in Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*, Boston, MA, pp. 11-20, 2003.

[15] Sommerville I. and Sawyer P., "PREview Viewpoints for Process and Requirements in Software Engineering," *Lancaster University*, Lancaster REAIMS/WP5.1/LU060, 1996.

[16] Sousa G., Soares S., Borb P., and Castro J., "Separation of Crosscutting Concerns from Requirements to Design: Adapting the Use Case Driven Approach," *Workshop Proceedings*, Lancaster, pp. 98-107, 2004.

[17] Tekinerdoğan B., Moreira A., Araújo J., and Clements P. "Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design," *Workshop Proceedings*, Lancaster, pp. 5-15, 2004.

[18] Tessier F., Badri L., and Madri M., "Towards a Formal Detection of Semantic Conflicts Between Aspects: A Model Based Approach," *in Proceedings of the 5th Aspect-Oriented Modelling Workshop, in conjunction with UML 2004*, Lisbon, Portugal, 2004.

Abdelkrim Amirat received his engineer and MSc in computer sciences from the University of Badji Mokhtar, Annaba, Algeria. Currently, he is a PhD student at the University of Nantes, France. His research interests include aspect oriented software development, requirement engineering, and software architecture.

Mohamed Laskri is a professor of computer science, he holds Doctorat 3ème cycle in computer science from France in 1987 and Doctorat d'état in computer science from Algeria in 1995. He is the leader of a research group in artificial intelligence in LRI laboratory, Algeria. His actually research includes artificial intelligence reasoning and its applications especially in image processing, multi-agent systems, interface engineering, and automatic processing of natural language.

Tahar Khammaci is an associate professor at University of Nantes since 1992. He obtained his MS and PhD degree from University of Nancy I, France. His area of interest includes software engineering, software architecture and automated software engineering. He supervised number of PhD and Master students. He has published number of books, chapters, and articles for International Journals and Conferences. He is member of IEEE Computer Society and ACM.