http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 6 (3): 332-337, 2007
ISSN 1812-5638

Towards a Requirements Model for Crosscutting Concerns

Abdelkrim Amirat
Laboratoire LINA CNRS FRE 2729, Université de Nantes,
2, Rue de la Houssinière, BP 92208, 44322 Nantes Cedex 03, France

Abstract: Techniques such as use cases, viewpoints and goals help achieve separation of stakeholders concerns but ensuring their consistency with global requirements and constraints is largely unsupported. The work on early aspects, therefore, complements these approaches by providing systematic means for handling such concerns. In this study we focus on a methodology to elicit the crosscutting concerns aspects in the early life phases of software development generally and especially during requirements analysis. Early aspects cannot be localized and tend to be scattered over multiple early life cycle modules. This reduces the modularity of the artifacts and might consequently lead to serious maintenance problems and low degree of reusability. Unfortunately, conventional aspect oriented software development approaches have mainly focused on identifying the aspects at the programming level and less attention has been taken on the impact of crosscutting concerns at the early phases of the software development.

Key words: Aspects, concerns, functional requirements, non-functional requirements

INTRODUCTION

Aspect-Oriented Software Development (AOSD) advocates the separation of crosscutting concerns during the software development. However, most research in this area has focused only on the design and implementation phases of the software lifecycle (Kiczales *et al*., 2001). The early aspects initiative refers to crosscutting properties at the requirements and architecture level. Requirements are classified into Functional Requirements (FRs) and Non-Functional Requirements (NFRs). While FRs refers to business logic, NFRs correspond to aspects such as synchronisation, scheduling, performance, security, reliability, real-time constraints and so on. By separating NFRs from FRs, maintainability, reusability and traceability of requirements are improved. The best way to deal with crosscutting requirements is to separate them from other requirements and model them independently. This modularization avoids tangled representations in the requirements document and facilitates requirements evolution. Use cases were first proposed by Jacobson (1992), as a requirement elicitation method and later embraced by most of the object-oriented methods. Developers find it an easy technique to structure the requirements of a system and to facilitate the communication with the stakeholders (Sommerville, 2001). Hence the crosscutting structure of use cases is part of what makes them useful and we can use Multi Dimensional Separation Of Concerns (MDSOC) (Ossher and Tarr, 2001) to decompose cleanly a system, use case by use case, over all the development models (i.e., use case model, analysis model, design model and implementation model), so that use cases remain separate all the way down to code (Jacobson, 2003).

Background: The object oriented paradigm uses classes to model and implement FRs, but lacks suitable concepts to express NFRs (Harrison and Ossher 1993). This situation is due to a number of reasons: NFRs express quality attributes not artefacts, they act upon other components or objects, or they can be thus considered as a global property of an application and thus can affect several requirements. Several approaches in the literature are pointed as the solution to deal with the NFRs; however, the aspect oriented software development approach allows the crosscutting concerns to be better encapsulated as aspects.

Use cases are the first artefacts to be established from requirement in the software development life cycle. They describe interactions between the system and its environment and thus capture the functional and non functional requirements of the system. A complete set of use cases specifies all different ways to use the system and therefore defines all behaviour required from the system (Kimour and Mestati, 2004). Those requirements

are described in terms of actors and use cases. The implementation of use cases in traditional object oriented programming leads to a situation such as: (1) a given component will contain the code coming from the implementation of several use cases (refereed to as code tangling), (2) a set of components will be required to implement a given use case, consequently, the use case implementation is not well modularised (refereed to as scattering code). As a result it is hard to get reusable and maintainable components as unit of requirements. AOP is a programming technique to have separated coding of functional parts from non-functional ones, called aspects and to weave them into final implementation. Applying this idea to requirements specification enhances the modularity of components.

Jacobson (2003) AOSD approach, each use case is realized using an object model defined for it alone and implementation for this use case is generated from the object model. Only the essential classes, attributes and methods needed to implement the use case are created, even though it is expected that other use cases may require conceptually similar classes with other or additional attributes and methods. The weaving process will handle the merging of those classes into a consistent whole-the deployed system (Suzuki and Yamamoto, 1999).

Overview of the approach: To situate our approach, we define a high-level process to identify, model and validate aspectual and non-aspectual behaviour. The approach can be summarized in the following four steps: (1) identify functional concerns; (2) identify and specify the crosscutting aspects; (3) apply the weaving process; (4) identify and resolve conflicts between aspects. The order in which functional and non-functional concerns are identified depends on the stakeholder views of the future system and the dynamics of the communication between the developer and the customers. Figure 1 depicts this mode.

Requirements identification

Identifying and specifying functional requirements: Functional requirements refer to the functionality or services basic concerns which the system is expected to provide, describing interactions between the system and its environment, while not focusing on implementation details.

The first step in this phase is a profound analysis (identification of actors and use cases) of the problem. Based on this analysis it is possible to build the use case diagram which capture FRs. For each use case occurrence, one template will be associated and will describe the use case in detailed way. Table 1 presents the template used to describe each use case. It was constructed by combining several proposals (Brito and Moreira, 2003;

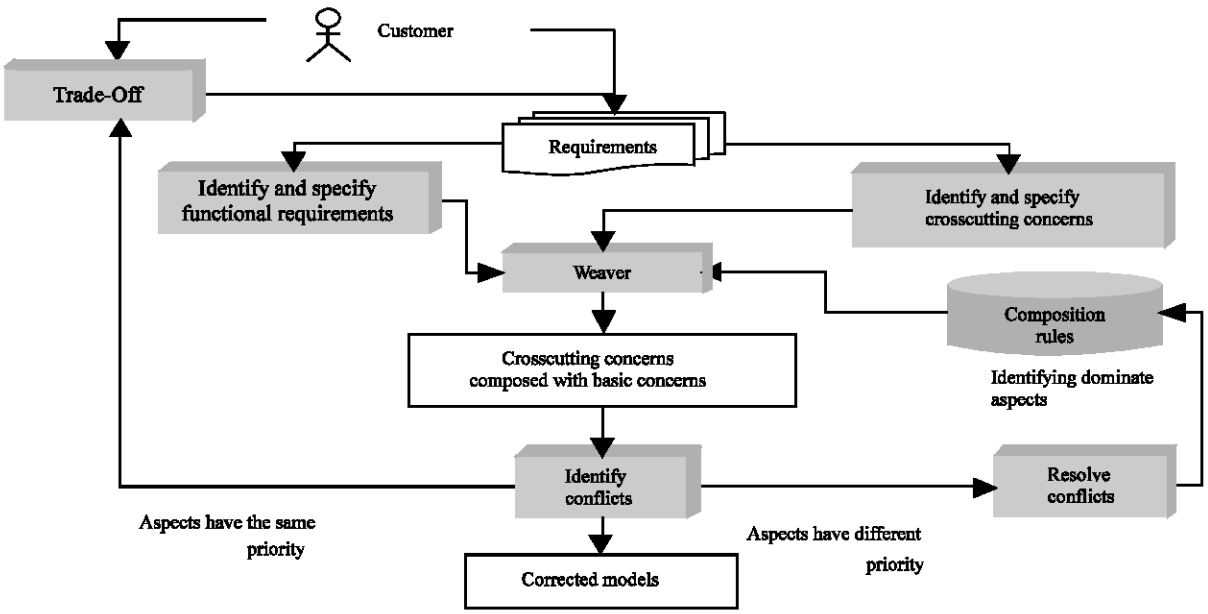

Fig. 1: A model for requirements analysis

Moreira *et al*., 2002; Rashid *et al*., 2003) in order to facilitate a detailed description of FRs and enrich theirs FRs analysis.

Identifying and specifying crosscutting concerns: Non-functional requirements are global properties crosscutting concerns that can influence part or the whole system. Several approaches can be used to identify non-functional requirements such as those used to identify goals. At this level of abstraction a crosscutting concern is a candidate aspect because it may be mapped later into an aspect, but it can also be mapped into a function. The template given in Table 2 represents a second variant of the template suggested in (Brito and Moreira, 2003) to capture the specification of non-functional requirements. We can use XML as a definition language to specify these requirements basic and crosscutting concerns. Notice that the information contained in these templates (Table 1 and 2) must be filled iteratively and incrementally.

Weaving crosscutting concerns with the basic ones: To achieve the weaving we firstly use a MP-Matrix Matching Point Matrix, representing the relationships between the stakeholders requirements (e.g., actors) and the model elements (e.g., use cases), to identify the set of matching points (Join point in AspectJ) of each candidate aspect based on the row (Where) in Table 2. This can be illustrated in MP-Matrix where each cell is filled with the list of candidate aspects (denoted CA_i), each filled cell represents a match point (denoted MP_i) (Table 3). Secondly we use C-Matrix Contribution Matrix, representing the contribution of each candidate aspect to the others, in order to derive the priority among aspects.

Table 1: Template to specify basic concerns

Name	Name of the basic concern
Primary actor	Name of the principal actor
Stakeholders	Users of the concern
Preconditions	Condition to be satisfied before the execution of the concern
Post conditions	Condition to be satisfied after the execution of the concern
Main scenario	The main scenario describing the functionality of the concern
Requirements	List of requirements describing the concern

Table 2: Template to specify crosscutting concerns

Name	Name of the crosscutting concern
Description	Short description
Priority	Importance of the concern (1: very low, 2: low, 3: medium, 4: high, 5: very high)
Decomposition	Concerns can be decomposed into simpler ones
Where	Matching point in model elements requiring the concern.
Requirements	List of requirements describing the crosscutting concern.
Contribution	Nature of contribution of each crosscutting concern to the others

Table 3: Matching point matrix

Concern / Stakeholder	$Concern_1$	…….	$Concerns_m$
$Stackeholder_1$	CA_1,CA_2 (MP_A)	…..	CA_3 …..
	……	….	
$Stackeholder_n$	CA_2,CA_4 (MP_B)		CA_k,CA_1 (MP_X)

This process is achieved by attributing weights (i.e., real value within the interval [0.1]) to represent the priority of each aspect in relation with others. The contribution can take a positive, negative or null value (no contribution). The following operators are adopted to identify how each candidate aspect affects the concerns it cuts across.

Overlap: The candidate aspect is applied before or after the concern it cuts across.

- before: the candidate aspect is applied before the concern it cuts across,
- after: the candidate aspect is applied after the concern it cuts across.

Override: The behavior described by the candidate aspect substitutes the behavior defined by the concern. (This operator represent the around qualifier in AspectJ without proceed ()).

Wrap: the behavior described by the concerns is enveloped by the behavior described by the candidate aspect. (This operator represent the around qualifier in AspectJ with proceed ()).

In the next step we use MP-Matrix, C-Matrix and the previous operators in order to establish the composition rules. The composition rules express the sequential order into which each candidate aspect must be composed (composition rules correspond to the definition of pointcuts in AspectJ). The weaving process use those composition rules to compose the basic concerns with the different aspects in all match points specified in MP-Matrix. Naturally those composition rules would be reconsidered if conflicts arise from the composition.

Identifying and resolving conflicts: When composing crosscutting concerns into a requirements model, conflicts may arise and must be resolved. Therefore, a decision has to be made in terms of which crosscutting concerns should have the maximum priority (i.e., should be composed first). If the priority attributed to each candidate aspect is different, the conflict problem is solved by identification of the dominant candidate aspect (Amirat and Laskri, 2005; Amirat *et al*., 2006).

However, if at least two candidate aspects have the same priority, a trade-off must be negotiated with the customer to modify the priority in order to resolve the conflict.

Dominant aspect identification: If two or more crosscutting concerns exist in a given match point, with negative contribution and the same priority, we start by analysing two concerns first to identify the dominant concern, then take the dominant and analyse it with a third concern and so forth, until all the crosscutting concerns are taken into consideration. The result is the concern with highest priority. Next we need to identify the second dominant crosscutting concern among the remaining concerns and so on until we get a dependency hierarchy between all the concerns.

Composition rules: The composition rule defines the order in which the concerns will be applied in a particular match point. It takes the form: <concern> <operator> <operator>. The operators we propose are (Before, After, Parallel, Around () without Proceed, Around with Proceed).

Example: Our aim here is to validate the approach and ideas using a simple version of the subway system. The top-level requirements for the subway system are as follows: To use the subway, a client has to own a card that must have been credited with some amount of money. A card is bought and credited in special buying machines available in any subway station. A client uses this card in an entering machine to initiate her/his trip. When she/he reaches the destination, the card is used in an exit machine that debits it with an amount that depends on the distance travelled. If the card has not enough credits the gates will not open unless the client adds more money to the card. The client can ask for a refund of the amount in the card by giving it back to a buying machine.

Step 1: Identify and specify functional concerns: Based on the requirements given above, let us consider the simple situation where only the actor Client is handled and we could think that the system has to offer the following functional concerns: BuyCard, LoadCard, RefundCard, EnterSubway and ExitSubway.

The corresponding use cases diagram to specify those concerns is illustrated in Fig. 2, where the use cases EnterSubway, ExitSubway and LoadCard have been refined to factorize the common functionality ValidateCard.

Step 2: Identify and specify non-functional concerns: Non-functional concerns can be identified based on the NFRs analysis for example we can elicit:

- Response time: The system needs to react in a short amount of time;
- Accuracy: Only right amounts should be debited from, or credited to a card;
- Multi-access: Several passengers can use the system concurrently;
- **Availability:** as the system and the machines must be available when the subway is open;
- **Security:** the card information must be protected against illicit actions.

In order to specify those concerns we can use hierarchy graph showing the interdependency between them (Fig. 3). Notice that a subject matter is added between brackets to the concern name. A subject matter is the topic addressed by the non-functional concern.

In this task we should also identify contributions between concerns, the priorities of each one and the list sub concerns required to accomplish the behavior of a

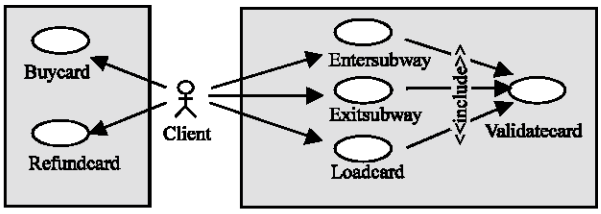

Fig. 2: The use case diagram of the subway system

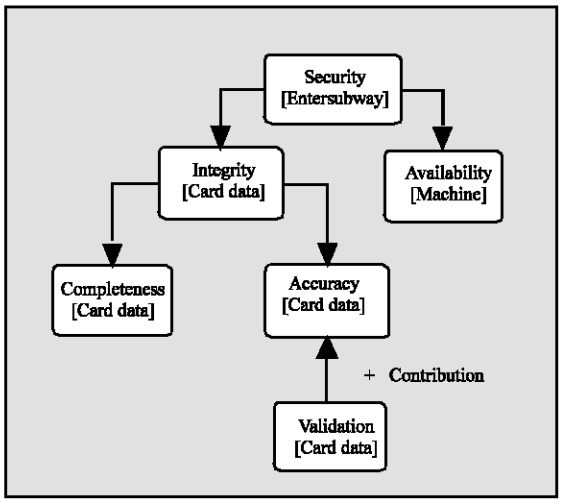

Fig. 3: Security specification for enter Subway

given concern. In our case the integrity and availability are both needed to achieve Security Finally at the end of this task, the template of each concern is complete. We give in Table 4 the complete template for the response time concern.

Step 3: Identify crosscutting concerns: Looking at rows required concerns we can identify crosscutting concerns. For example, Response time crosscutting because it is required in EnterSubway and ExitSubway. Other crosscutting concerns are: ValidateCard, Availability, Multi-access, Security and Accuracy.

Step 4: Compose concerns: The goal here is to compose all concerns to obtain the whole system. During this process conflicts may be identified. If conflicts arise we should resolve them in the following steps.

- Identify match points: This is accomplished by building the match points table. In our example in can quote that we can have tow matching points MP_A and MP_B (Table 5).
- Identify Conflicts: In our case study the MP_A match point has five concerns; Availability and Response Time contribute negatively to each other.
- Identify dominant concern: To solve the possible conflicts identified above, we need to verify the priority of each concern involved. Let us only deal with Availability and response time witch have the same priority (very important) after a trade-off with the stakeholders, the priority of response time is decreased thus Availability becomes the dominantconcern in MP_A.
- Composition rules: A composition rule for MP_A could be defined using the operators described above as follows:

Security. Availability before ((ValidateCard before inter Subway)||
Response Time || Security. Integrity. Accuracy) before Security. Integrity.

We could informally say that after the successfully satisfaction of Availability (before operator), Response Time, EnterSubway and Accuracy must synchronize and be satisfied in parallel (||: Parallel Operator). Notice that EnterSubway can only be satisfied after the successful satisfaction of ValidateCard (before operator). Only after this will Integrity be satisfied. (We use "." notation to represent a sub-concern of a given concern).

Related works: Researchers have suggested different approaches for modeling aspects at higher levels of abstraction. Several approaches have been proposed for separating concerns at code and design level, but few at the requirements level. The work done on separation of concerns in the requirements engineering community e.g., viewpoint, use cases and analysis models does not explicitly focus on crosscutting concerns.

Table 4: Response time template

Name	Response time
Description	The system has to react in time in specific situations, for instance to enter the subway, exit the subway
Priority	Very high
Decomposition	<none>
Where	Entersubway, Exitsubway.
Requirements	Entersubway, Exitsubway.
Contribution	(-) Security, (-) Multi-access

Table 5: Identification of match points

Concern / Stakeholder	Enter subway	Buycard	Validate card	....
Client	RT, S, AC, AV, V (MP_A)	S, AC, AV (MP_B)	AC	....

(AC: Accuracy; RT: Response Time; S: Security; AV: Availability; V: Validatecard)

Seaki and Kaiya (2004) discuss transformation based weaving aspects in requirements analysis. The requirements analysis process is based on goal oriented method and use case modeling one. Weaving can be formalized with transformation rules to derive the use cases and their structure that are satisfied with non-functional requirements, the approach use crosscutting tables to design how transformation should be applied.

Sampaio *et al.* (2005) describe an approach that uses corpus-based Natural Language Processing (NLP) techniques to effectively enable the identification of early aspects in a semi-automated way. The technique proposed describe how different sources of requirements (e.g., interviews, natural language descriptions of the system, etc) can be automatically mined to help the requirements engineer quickly identify and build a structured aspect-oriented model of the system's requirements.

Finally Rashid *et al.* (2002, 2003) propose an approach for modularizing and composing crosscutting concerns. The approach involves identifying requirements using stakeholders' viewpoint, use cases/scenarios, goals or problem frames. The approach basically uses a set of matrices consisting of the viewpoints and concerns represented in XML to define the composition rules.

CONCLUSIONS

Generally scattering and tangling do not occur only in implementation artifacts. They emerge in other artifacts throughout the development process. For this reason, it is necessary to apply the separation of crosscutting concerns in all development stages. As a result, the comprehensibility, maintainability and reusability of software system artifacts are improved. Since requirements direct the software development, they are crucial for quality. Both functional and non-functional requirements shall be identified as soon as possible and their elicitation must be accurate and complete.

This study shows how to discover non-functional requirements that represent software quality attributes in order to identify the separate aspects, as required by the AOSD approaches. The main ideas presented here are those of match point, conflicting aspect, dominant aspect and composition rule. A composition rule is defined for each match point and when a conflicting situation emerges, in a match point, we must resolve it by identifying the dominant candidate aspect.

Our future work will focus on developing concrete algorithms to achieve an automatic composition process based on XML scheme representation of requirements and interactive system that helps detecting and resolving conflicts.

REFERENCES

Amirat, A. and M.T. Laskri, 2005. Modular implementation of aspectual requirements. Proceedings of the international Arab conference on information and technology ACIT'05, Amman, Jordan, pp: 159-163.

Amirat, A.D. Meslati and M.T. Laskri, 2006. An aspect-oriented approach in early requirements engineering. Proceedings of the 4th ACS/IEEE International Conference on Computer Systems and Applications AICCSA' 06, Sharjah, UAE, pp: 224-227.

Brito, I. and A. Moreira, 2003. Towards a composition process for aspect-oriented requirements. Proceeding of AOSD'03 workshop on early Aspects: Aspet Oriented Requirements Engineering and Architecture Design, March 17, Boston USA.

Harrison, W. and H. Ossher, 1993. Subject oriented programming: A critic of pure object. Proceeding of the Conference on Object Oriented Programming Systems Languages and Applications, Wash. D.C., pp: 411-428.

Jacobson, I., 1992. Object-oriented Software Engineering -Use Case Driven Approach, Addison-Wesley, Reading Massachusetts.

Jacobson, I., 2003. Use cases and aspects-working seamlessly together. J. Object Technol., 2: 7-28.

Kiczales, G., E. Hilsdale, J. Hugunin, M. Kersten, J. Pal and W.G. Griswold, 2001. An overview of aspect J. Proceedings of 15th European conference on object-oriented programming, budapest, pp: 327-355.

Kimour, M.T. and D. Mestati, 2004. An Approach to Building Object Models with UML In embedded systems. J. CIT, 12: 223-235.

Moreira, A., J. Araujo and I. Brito, 2002. Crosscutting Quality Attributes for Requirements Engineering. 14th international Conference on Software Engineering and Knowledge Engineering (SEKE'02), ACM Press, Italy, 2002.

Ossher, H. and P.L. Tarr, 2001. Hyper/J: Multi-Dimensional Separation of the Concerns for Java. International Conference on Software Engineering, ACM, pp: 734-737.

Rashid, A.P. Sawyer, A. Moreira and J. Araujo, 2002. Early Aspects: A model for aspect-oriented requirements engineering. In: IEEE Joint International Conference on Requirements Engineering, Essen, Germany, IEEE, CSP, pp: 199-202

Rashid, A., A. Moreira and J. Araujo, 2003. Modularization and composition of aspectual requirements. Proceeding of 2nd International Conference on AOSD'03, Boston, Mar, ACM Publishing, pp: 11-20.

Sampaio, A., N. Loughran, A. Rashid and Rayson, 2005. mining aspects in requirements. Proceeding of the 4th workshop on AOSD, AOSD'05. Chicago, Illinois, USA.

Seaki, M. and H. Kaiya, 2004. Transformation Based Approach for Weaving Use case Models in Aspect-Oriented Requirements Analysis. ACM.

Sommerville, I., 2001. Software Engineering. Addition-Wesley, 6th Edn.

Suzuki, J. and Y. Yamamoto, 1999. Extending UML with Aspects: Aspect support in the design phase. Proceeding of the 3rd ECOOP Aspect Oriented Programming Workshop, Lisbon, Portugal.