Aspect-Oriented Requirements with UML

João Araújo and Ana Moreira
Dept. Informática, FCT
Universidade Nova Lisboa
2829-516 Caparica, Portugal
+351 212948536
{ja,amm}@di.fct.unl.pt

Isabel Brito
Dept. de Engenharia
Instituto Politécnico de Beja
7800-050 Beja, Portugal
+351 284311540
isabel.sofia@estig.ipbeja.pt

Awais Rashid
Computing Department
Lancaster University Lancaster
LA1 4YR, UK
+44 1524 592344
awais@comp.lancs.ac.uk

ABSTRACT

Crosscutting concerns are responsible for producing spread and tangled representations throughout the software life cycle. Effective separation of such concerns is essential to improve understandability and maintainability of artefacts at the various software development stages. Aspect-oriented software development holds promise for the purpose. However, to date, most of the work in this area has concentrated on the implementation level. While the focus is shifting to earlier development stages such as design, very less work exists on separation of crosscutting concerns during requirements engineering.

The goal of this paper is to handle the separation of crosscutting concerns at requirements level using UML. To accomplish this we identify and specify crosscutting concerns in separate modules, so that localization and hence, reusability and maintainability can be promoted. The UML-based aspect-oriented requirements engineering mechanism has a two-fold impact. It makes it possible to identify trade-offs among broadly scoped properties early on in the development cycle hence providing decision support for the stakeholders involved. At the same time, being based on UML, the approach adheres to a de-facto industry standard hence making it suitable for incorporation in existing requirements engineering practices.

1. INTRODUCTION

Separation of concerns is a central software engineering principle that should be applied throughout the software development process, from requirements to implementation [6]. The basic idea is to handle one property of a system at a time. This involves identifying, encapsulating and using parts of a system related to a specific area of interest.

The concerns that we are interested in are those that crosscut, i.e. transverse, other concerns at the requirements level (e.g. response time, security). There are other crosscutting concerns that may appear during design and implementation due to limitations imposed by the technology chosen to implement a system (e.g. exception handling, synchronization). Multidimensional separation of concerns, composition filters, adaptive techniques and aspect languages are some of the approaches dealing with crosscutting concerns at design and implementation levels [1].

This work presents an approach to handle crosscutting non-functional concerns at the requirements stage[1]. Non-functional requirements are global properties of a system that constrain the functional requirements [3]. During the requirements elicitation, while identifying the user requirements, it happens that stakeholders describe their system in terms of functional and non-functional requirements. For example, a stakeholder may tell us that a bank account should support deposits and withdrawals and that the system should react to the account' owners requests in a short period of time. That is, on the one hand the user is describing the accounts functionality (deposits and withdrawals), on the other hand s/he is explicitly concerned with response time. Surely, those are two different types of requirements. Most approaches to requirements engineering and modelling are focused on identifying the business requirements of the systems (e.g. those that refer to an account functionality). However, we should not ignore the type of restrictions that a user is already imposing on the systems solution (in this case, response time).

This discussion highlights the need to have, on the one hand, approaches that treat crosscutting concerns (i.e. candidate aspects[2]) homogeneously from requirements to implementation; on the other hand, the need to equip requirements engineering methods with mechanisms to rapidly manage and understand the whole systems requirements. The goal of this paper is to handle the separation of crosscutting concerns at requirements level. The UML models [15] will be used as a basic notation to express the requirements. The integration of aspects in the UML would augment its power.

[1] Crosscutting concerns can also be functional [12]. However, this paper focuses on non-functional properties. Separation of crosscutting functional requirements will form the subject of a future publication.

[2] Crosscutting concerns at the requirements level are candidate aspects. They can be mapped onto design aspects, functions or architecture decisions. For more information please see [13].

This paper is organized as follows. Section 2 presents some related work on aspect–orientation. Section 3 applies our approach to aspect-oriented requirements using a case study. Finally, section 4 draws some conclusions and highlights some future work.

2. RELATED WORK

In the last couple of years there has been growing interest in propagating the aspect paradigm to the earlier activities of the software development life cycle. At the Aspect-Oriented Software Development conference, one workshop was organised on aspect-oriented requirements and architecture [17] and another on aspects and UML [16]. Integrating aspects with UML seems to be an obvious area of research. Indeed, this subject has raised interest in the UML community. The well-known work on incorporating aspects into the UML was accomplished by [5, 8, 9, 14].

Suzuki and Yamamoto proposed an extension to UML to support aspects, where an aspect is described as a classifier in the meta-model [14]. They extend UML with aspects to support the design activity. Also, they propose a XML based aspect description language to interchange aspect models between development tools such as CASE tools and aspect weavers.

Composition patterns is an approach to handle crosscutting requirements at the design level [5]. This approach promotes reusability and traceability to the following activities of the software development. This model is based on subject-oriented design and uses UML templates.

The aspect-oriented requirements engineering approach by Grundy is targeted to component based software development, where there is a characterization of diverse aspects of a system that each component provides to end users or other components [8]. This approach is too specific for component development, not showing evidence of its use in software development in general. Besides, the identification of aspects for each component is not clearly defined.

An UML compliant approach to handle quality attributes at the requirements activity of the software development process was proposed in [11]. The work we present here builds on this by extending the notation used and adding more to the identification and resolution of conflicting crosscutting concerns.

3. AN APPROACH TO ASPECT-ORIENTED REQUIREMENTS

3.1 Overview

The proposed approach is a UML-based realisation of the general aspect-oriented requirements engineering process presented in [13]. While [13] described a viewpoint-based implementation of the process, this paper will describe how such a process can be supported when engineering crosscutting requirements with the UML. A simplistic view of the aspect-oriented requirements engineering process is depicted in Figure 1.

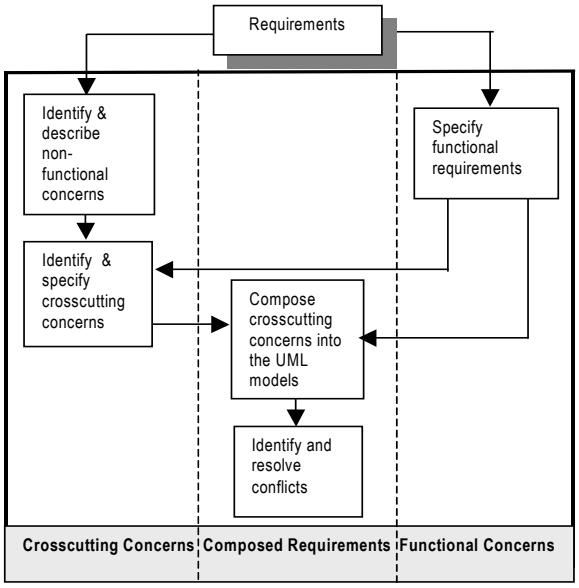

Figure 1. A model for aspect-oriented requirements

The process is partitioned vertically in three main parts:

- Crosscutting concerns: this handles first the non-functional requirements and then identifies which of those are crosscutting, i.e. which are candidate aspects. Candidate aspects[3] are specified using the template depicted in Table 1.
- Functional concerns: this performs a traditional specification of functional requirements, in this case, using an UML-like approach where the use case model is the main specification technique.
- Composed requirements: this starts by composing functional requirements (modelled using UML) with aspects; then it identifies and resolves conflicts that may arise from the composition process. We adopt the concepts of overlapping, overriding and wrapping, commonly used in various separation of concerns approaches [2, 5, 10, 18], to define the composition part of the model. This is accomplished as follows:
  - ♦ Overlapping: the requirements of the aspect modifies the functional requirements they transverse. In this case, the aspect requirements may be required before the functional ones, or they may be required after them.
  - ♦ Overriding: the requirements of the aspect superpose the functional requirements they transverse. In this case, the behaviour described by the aspect requirements substitutes the functional requirements behaviour.
  - ♦ Wrapping: the requirements of the aspect "encapsulate" the functional requirements they transverse. In this case, the behaviour described by

[3] For simplicity we will use "aspect" instead of "candidate aspect" from now on.

the functional requirements is wrapped by the behaviour described by the aspect requirements.

Table 1. Specification of crosscutting concerns

Crosscutting concern	<Name>
Description	<Executive description>
Priority	<Priority can be Max, Med and Min>
List of requirements	<Requirements that describe the concern>
List of models	<UML models influenced by the concern >

3.2 Applying the approach to a case study

The case study we have chosen is a simplified version of the real system implemented in the Portuguese motorways network [4]. The requirements are stated as follows:

"In a road traffic pricing system, drivers of authorised vehicles are charged at toll gates automatically. The gates are placed at special lanes called green lanes. A driver has to install a device (a gizmo) in his/her vehicle. The registration of authorised vehicles includes the owner's personal data, bank account number and vehicle details. The gizmo is sent to the client to be activated using an ATM that informs the system upon gizmo activation.

A gizmo is read by the toll gate sensors. The information read is stored by the system and used to debit the respective account.

When an authorised vehicle passes through a green lane, a green light is turned on, and the amount being debited is displayed. If an unauthorised vehicle passes through it, a yellow light is turned on and a camera takes a photo of the plate (used to fine the owner of the vehicle). There are three types of toll gates: single toll, where the same type of vehicles pay a fixed amount, entry toll to enter a motorway and exit toll to leave it. The amount paid on motorways depends on the type of the vehicle and the distance travelled."

3.2.1 Identify and describe non-functional concerns

The problem description presented in the previous section contains functional and non-functional requirements. However, in order to have a complete description of each non-functional requirements we need to discuss with the stakeholder the kind of restrictions that the system has to satisfy.

For example, from our case study, this will be the moment when we have to decide about the time in which a tollgate has to react when a vehicle uses the system. By further analysis of the requirements we can identify that this time can be calculated as a function of the speed limit allowed and the distance between the various elements that compose a tollgate (sensors to detect the vehicle and read the gizmo, light, display and camera). This leads us to determine where all those components should be physically located. For example, if we have to photograph the plate number and the driver then the camera has to be located in a different place from that if the plate number can be photographed from the back of the vehicle.

Also, it is very important that the light is turned green (or yellow) before the vehicle leaves the tollgate area. In a similar way, the amount to be charged later has to be displayed while the driver is able to see it. Therefore we can say, from the externally observable behaviour, that the tollgate has to react fast enough so that the driver (vehicle) can see the light and the amount that will be later charged. From this perspective, we can talk about "tollgate response time".

The "tollgate response time" concern can be described with a numbered set of requirements, as follows:

R1. "When a car crosses a toll-gate, the system has to read the identifier in time *t1*."

R2. "Unauthorized vehicles using the green lane, have their plate numbers photographed in time *t2*."

R3. "When a car crosses a toll gate, the system has to turn on the light in time *t3*."

R4. "When an authorized vehicle crosses the gate, the system has to display the amount in time *t4*."

The nature of these requirements, i.e. whether they cut across or not other concerns, can be better analyzed after we study in more detail the functional requirements. Other non-functional concerns are: security, multi-user system, compatibility, correctness, legal issues.

3.2.2 Specify functional requirements

We propose the UML use case model and interaction diagrams as description techniques to specify functional requirements. Analysing these set of requirements we can identify the following actors:

- Vehicle owner: this is responsible for registering a vehicle;
- Vehicle driver: this comprehends the vehicle and the gizmo installed on it;
- Bank: this represents the entity that holds the vehicle owner's account;
- System clock: represents the internal clock of the system that monthly triggers the calculation of debits.

The following are the use cases required by the actors listed above:

- Register vehicle: is responsible for registering a vehicle and its owner, and communicate with the bank to guarantee a good account;
- Pass single toll: is responsible for dealing with tolls where vehicles pay a fixed amount. It reads the vehicle gizmo and checks on whether it is a good one. If the gizmo is ok the light is turned green, and the amount to

be paid is calculated and displayed. If the gizmo is not ok, the light is turned yellow and a photo is taken.

- Enter motorway: checks the gizmo, turns on the light and registers an entrance. If the gizmo is invalid, a photo is taken.
- Exit motorway: checks the gizmo and if the vehicle has an entrance, turns on the light accordingly, calculates the amount to be paid (as a function of the distance travelled), displays it and records this passage. If the gizmo is not ok, or if the vehicle did not enter in a green lane, the light is turned yellow and a photo is taken.
- Pay bill: sums up all passages for each vehicle, issues a debit to be sent to the bank and a copy to the vehicle owner.

Figure 2 shows the use case diagram of the road traffic system.

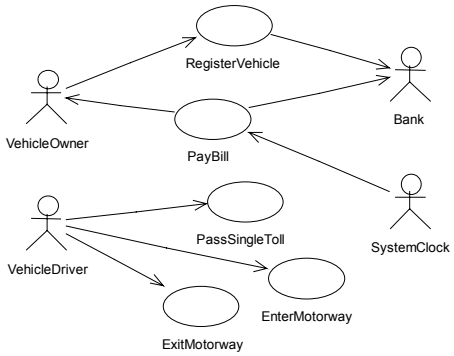

Figure 2. The use case diagram of the toll gate collecting system

We describe use cases using scenarios (primary and secondary) and each scenario is then further described using sequence diagrams. For the use cases PassSingleToll, EnterMotorway and ExitMotorway, we can identify at least two scenarios for each one; one to deal with authorised vehicles and another to deal with non-authorised vehicles.

Figure 3 shows a sequence diagram for the primary scenario. (For some non-functional requirements we may not need to "explode" the sequence diagram, i.e. show all the interactions that take place between objects inside the system.)

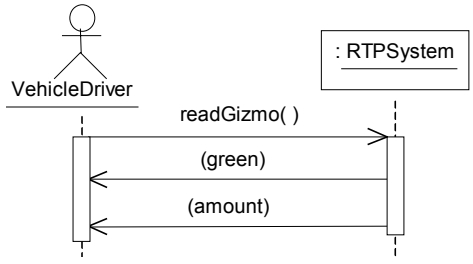

Figure 3. Sequence diagram for "authorized vehicle passing a single toll gate"

The idea is that the system is represented by the object RTPSystem. The road traffic pricing system reads the gizmo and, if this is a valid one, the actor VehicleDriver sees the light green and the amount to be paid in the display. This represents the externally visible behaviour of the system for that scenario.

3.2.3 Identify and specify crosscutting concerns

A non-functional requirement is crosscutting if it transverses, i.e. affects, more than one use case. For example, let us consider "response-time" when vehicles use the system. This non-functional requirement affects PassSingleToll, EnterMotorway and ExitMotorway. For this reason, "toll gate response time" is crosscutting. Table 2 illustrates this.

Table 2. Template specification for TollGateResponseTime

Crosscutting concern	Toll gate response time
Description	Tollgates should react before the driver leaves the toll gate area
Priority	Max
List of requirements	R1, R2, R3, R4
List of models	Use cases: PassSingleToll, EnterMotorway, ExitMotorway

3.2.4 Composing crosscutting concerns into the UML models

The criteria for integrating both functional and crosscutting requirements are: completeness and sufficiency. With completeness we guarantee that all the requirements needed to support composition are included in the aspect. With sufficiency we guarantee that every requirement in an aspect must have an impact in the composition process.

Let us take "toll gate response time" and compose it into the two UML models we have used to describe the functional requirements. For the use case diagram we can define a special use case with the stereotype <<TollGateResponse-Time>> (see Figure 4).

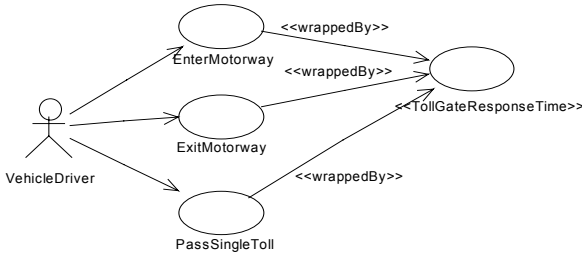

Figure 4. A use case composed with an aspect

This crosscutting concern wraps, using the stereotyped relationship <<wrappedBy>>, the use cases PassSingleToll, ExitMotorway and EnterMotorway. This means that the functional behaviour described by these use cases is wrapped by the behaviour described by the requirements of the aspect TollGateResponseTime.

To compose TollGateResponseTime with a sequence diagram we can be inspired by [7]. Figure 5 shows the real-time constraints composed into the scenario "authorized vehicles pass single toll".

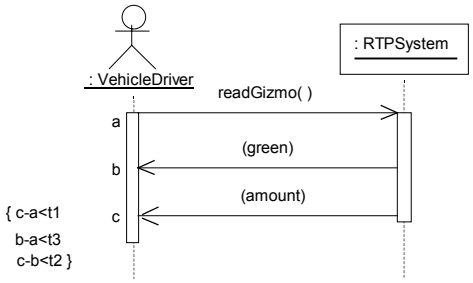

Figure 5. A sequence diagram composed with an aspect

Here, we included event identifiers within the sequence diagram (e.g. "a", "b" and "c"). These are used to reference the event that gives rise to the message. Event identifiers are then included in timing mark expressions to indicate the relative time between events. These expressions specify the timing constraints and are shown between curly braces. In the situation where we have several constraints (such as the one we are evaluating) it is possible to provide multiple timing constraints within the same timing mark expression, as shown in Figure 5.

According to Douglass, these expressions can be used to specify constraints using functions. (These constraints could be specified in OCL.)

3.2.5 Identifying and resolving conflicts

Composing a crosscutting concern into a requirements model may reveal in conflicts that have to be solved. It may well be that crosscutting concerns may cause contradictory situations in a system.

We have been looking at the situation where during the composition of crosscutting concerns with functional requirements conflicting behaviour may arise. For example, "Toll gate response time" and "security" are two crosscutting concerns that affect a toll gate. When trying to compose these concerns with the (same set of the) toll gate requirements a conflict will be found, as both crosscutting concerns contribute negatively to each other. Therefore, a decision has to be made in terms of which crosscutting concerns should have the maximum priority, i.e. should be composed first.

This suggests that what we should do is to first study the contribution from one crosscutting concern in relation to all the others. This contribution can be positive or negative [3]. If two (or more) crosscutting concerns contribute negatively to each other we are facing a conflicting behaviour if, and only if, these crosscutting concerns influence the same set of requirements. To resolve these kinds of conflicts, which affect the whole system or a part of it, a trade-off must be negotiated with the stakeholders. In this situation what we propose is to attribute priorities to the concerns and compose them according to these.

4. CONCLUSIONS AND FUTURE WORK

This paper proposes an approach to handle crosscutting concerns (i.e. candidate aspects) at the requirements level, using the UML. The approach is composed of three main parts: crosscutting concerns, functional requirements and composition. The first part handles first the non-functional requirements and then the crosscutting concerns. The second part performs a traditional specification of functional requirements, in this case, using an UML-like approach where the use case model and sequence diagrams are the main specification techniques. Finally, the third part handles first the composition of functional requirements (modelled using UML) and the crosscutting concerns and then it identifies and solves conflicts that may arise from the composition process. This, in turn, provides decision support for stakeholders making it possible for the requirements engineer to establish early trade-offs and balance the various conflicting, broadly scoped properties.

We are currently working on a method to help us identifying other types of conflicts that may arise both during the specification of crosscutting concerns and after the composition process. Part of this work will deal with the order in which the composition process should tackle the crosscutting concerns. We also aim to develop tool support for the UML-based aspect-oriented requirements engineering approach.

REFERENCES

[1] *Communications of ACM*. Special Issue on Aspect-Oriented Programming, **44** (10), 2001.

[2] Bergmans, L. M. J. and Aksit, M. "Composing Software from Multiple Concerns: A Model and Composition Anomalies", Multi Dimensional Separation of Concerns in Software Engineering Workshop, ICSE 2000, Limerick, Ireland, 2000.

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

[4] Clark, R., Moreira, A. "Constructing Formal Specifications from Informal Requirements", Software Technology and Engineering Practice, pp. 68-75, IEEE Computer Society, Los Alamitos, California, July 1997.

[5] Clarke, S., Walker, R.J. "Composition Patterns: An Approach to Designing Reusable Aspects", Proceedings of International Conference on Software Engineering, ICSE 2001, Toronto, Canada, 2001.

[6] Dijkstra, E.W. *A discipline of programming*, Prentice-Hall, 1976.

[7] Douglass, B.P. *Real Time UML - Developing Efficient Objects for Embedded Systems*, Addison-Wesley, 1998 (pp.80).

[8] Grundy, J. “Aspect-oriented Requirements Engineering for Component-based Software Systems”, 4th IEEE International Symposium on Requirements Engineering, IEEE Computer Society, Limerick, Ireland, 1999, pp. 84-91

[9] Ho, W.-M., Pennaneac’h F., Plouzeau, N. “UMLAUT: A Framework for Weaving UML-Based Aspect-Oriented Designs”, http://www.irisa.fr/pampa/UMLAUT/ download. Htm

[10] IBM Research. MDSOC: Software Engineering Using Hyperspaces, http://www.research.ibm.com/hyperspace/

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

[12] Rashid, A. and Sawyer, P. “Aspect-Orientation and Database Systems: An Effective Customisation Approach”, IEE Proceedings - Software, 2001, **148**(5): p. 156-164.

[13] 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, pp 199-202.

[14] Suzuki, J. and Yamamoto, Y. “Extending UML with Aspects: Aspect Support in the Design Phase”, AOP Workshop at ECOOP’99, Lisbon, Portugal, 1999.

[15] Unified Modeling Language, version 1.4. http://cgi.omg.org/, 2001.

[16] Workshop on “Aspect-oriented Modeling with UML”. ttp://lglwww.epfl.ch/workshops/aosd-uml/

[17] Workshop on “Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design”. http://trese.cs.utwente.nl/AOSD-EarlyAspectsWS/

[18] Xerox Parc, AspectJ home page, Technical report, http://www.aspectj.org/, 2000.