On identifying and representing Aspects

Saqib Iqbal, Gary Allen
School of Computing and Engineering,
The University of Huddersfield, Huddersfield HD1 3DH, UK

Abstract - *Identification of cross-cutting concerns (Aspects) in the earliest phases of software development has gained in popularity over recent years. Many approaches have been suggested for identifying and representing Aspects in abstraction and design structures. Since these approaches are still relatively immature, shortcomings such as overlooking or not properly locating Aspects have been noted in almost all of these approaches. This paper discusses some of these methods and suggested approaches, and provides a constructive critique on Aspects as Use Cases, View-Point based system of identifying Aspects, and Use Cases as Concerns. This paper also suggests a model-oriented approach for identifying and representing Aspects throughout the development life cycle.*

Keywords: Aspect-Oriented Programming (AOP), Aspect Identification, Aspect Representation, Aspect Modeling

1 Introduction

Many essential system requirements, such as efficiency, security, fault-tolerance, and synchronization of threads, are difficult to handle, especially in the implementation of large-scale systems. The slightest mishandling of any of these requirements can result in a big problem, or even disaster in the case of safety critical systems. Such requirements are rarely limited to one module or unit of a system, rather their implementation spreads over a set of modules or sub-modules. Their implementation, therefore, also involves more than one programming unit. Hence their code is scattered and tangled across the whole system, and that is why they are known as cross-cutting (or Aspectual) concerns of the system. The complex, yet important nature of these Aspects has forced software engineers to address them separately from the base program. Aspect-oriented programming (AOP) [5] has been proposed as a programming paradigm to handle these cross-cutting concerns. Since its inception, a lot of work has been carried out on the better implementation of Aspects. There are plenty of tools and technologies such as AspectJ, AspectWorkz, and Spring available which implement Aspects differently according to the requirements and environments.

Aspects are usually handled in the implementation phase. Their identification usually relies on the strength of domain knowledge of the implementer. There has been very little work in identifying Aspects at the earlier stages of software development. Although some techniques have been proposed, like the AORE (Aspect-Oriented Requirements Engineering) model by Rashid et al [1] which builds on View Point Model [1], and the COSMOS [7] model by S. Sutton which proposes a technique to capture concerns in the early stages. The problem with both of these techniques is that they do not specifically capture cross-cutting concerns; rather they talk about general concerns of the system. Some of other related work can be found in [8], [10] and [12]. All these techniques and models either address modeling of Aspects or identifying general concerns including non cross-cutting concerns (non Aspects).

There are some approaches which represent Aspects in Use Case models. For example, Saiki and Keya propose generating a use case model for Non functional requirement (NFR or Aspects) [9]. Araújo and Coutinho proposed developing a vision document based on a viewpoint-oriented method to separate Aspects from basic concerns [11], and Araujo et al. have proposed extensions in UML showing use cases in the use case model and suggested techniques to implement Aspects as use cases [2]. In this paper, we have tried to provide a positive critique on some of these approaches for identifying and representing Aspects. We have also suggested a model-oriented approach for handling Aspects at the earlier stages of software development.

The rest of the paper is organized as follows; Section 2 is about the background and critical analysis of the contemporary approaches proposed for identifying and representing Aspects. Section 3 is on the proposed Aspect model and its breakdown in UML diagrams and Section 4 gives conclusion of the paper and describes the next step in research.

2 Background

Since the conception of the term “cross-cutting concerns”, most of the effort has been put in developing strategies and tools for their implementation. As a result, many efficient tools (e.g. AspectJ) have emerged and served the purpose. Before implementing cross-cutting concerns, the main goal is to identify them. In [13] the term “Early Aspects” was introduced for the first time which has now become a de-facto term for identification of cross-cutting concerns at the early stages of software development. Many approaches have

been proposed for requirements engineering of early Aspects. Some of the most prominent ones are:

a) Viewpoint-based approaches with Arcade model [13][1]

b) Goal-oriented approaches [4]

c) Use case- and scenario-based Approaches [3][6][14] and Aspectual use case driven approaches [21].

d) Multi-dimensional separation of concerns [7] by Cosmos [15][16]

e) Concern-Oriented Requirements Engineering [17][19]

f) Aspect-Oriented Requirement Engineering for Component-Based Software Systems [18]

g) Theme/Doc approach [20]

We now discuss three of these approaches, Viewpoint-based, Use case-based and Concern-Oriented requirement engineering in detail and analyze their strengths to capture Aspects.

2.1 Viewpoint-based Approaches with Arcade Model

Viewpoint-based approaches give a fairly new way of representing and abstracting requirements. We might also call them representing requirements on the roles of stakeholders, which makes pretty good sense because each role's perspective and usage is different from that of others. Capturing the right perspective can result in requirement satisfaction and ease of use. In [1], requirements have been presented in PREview-like viewpoints which compose requirements in XML based notations. Aspects have also been represented in the same XML notation along with corresponding viewpoints. The following example, taken from [1], presents viewpoint-based representation. It demonstrates an extract of a Portuguese toll collection system in which a device called gizmo is installed in a car and is activated to pay tolls as the car passes the toll gate:

```
<?xml version="1.0" ?>
 <Viewpoint name="ATM">
  <Requirement id="1">
     The ATM sends the customer's card
     number,account number and gizmo
     identifier to the system for activation
     and reactivation.
   <Requirement id="1.1">
     The ATM is notified if the activation
     or reactivation was successful
     <Requirement id="1.1.1">
        In case of unsuccessful activation
        or reactivation the ATM is notified
        of the reasons for failure.
     </Requirement>
   </Requirement>
  </Requirement>
  <?xml version="1.0" ?>
<Concern name="Compatibility">
</Viewpoint>
```

```
<?xml version="1.0" ?>
<Concern name="Compatibility">
  <Requirement id="1">
    The system must be compatible
    with systems used to:
   <Requirement id="1.1">
     activate and reactivate gizmos;
   </Requirement>
   <Requirement id="1.2">
    deal with infraction incidents;
   </Requirement>
   <Requirement id="1.3">
    charge for usage.
   </Requirement>
  </Requirement>
</Concern>
```

This could be considered as well-represented early Aspects, but we may argue that the identification of Aspects still depends on the domain knowledge of the requirements engineer. One still has to look at the user stories and try to come up with a corresponding concern. This approach doesn't propose an automated or procedural way of identifying Aspects from the crude user requirements. There is always a probability with such an approach that some of the concerns are overlooked or may not be considered as concerns at this level, and if they then creep in later in the development cycle then the whole purpose of identifying them at the earlier stage is undermined.

2.2 Use Cases as Concerns

Use cases identify and partition system functionality. They describe the behavior of the system in response to the interaction of the user. They provide an easily understandable abstract model of the system at the earliest phases of analysis and design. The thing to remember here is that use cases only show the functionality of the system as performed by particular users. They do not show how that functionality is performed internally. In other words, use cases provide high level, black box view of the system. If we start thinking that use cases are the only functionality that we have to implement then we are looking only at the tip of iceberg. Ivor Jacobson in [6] tries to create a resemblance between Aspects and use cases. We agree with him that use cases are also concerns of the system. These are the major functionalities or behavior of the system, so they can be regarded as the major concerns, but there are some other concerns which we do not consider at the point of use case modeling. For instance, let's take the example of a Cell phone. Some of the use cases could be "Call Someone", "Receive a Call", "Save a Contact", etc. However, there are other important functionalities, such as Phone Book Management, Energy Management or I/O Management, that we cannot call use cases because these functions are performed internally and they are not initiated by the user of the cell phone.

2.3 Use Cases as Components

Ivor Jacobson in [6] states that implementation of use cases crosscut the set of components and component-based techniques fail to achieve use case modularity. A piece of code of a component may contain code of multiple use cases which will result in code tangling problem and similarly if we implement a use case, a set of components will constitute its implementation which is a crosscutting property.

Pawlak and Younessi also back these assumptions of Jacobson in [21]. They further propose that methodologies should be developed which could work only on abstraction, designing, composition and testing of use cases for developing the whole system. In this way, they hope to achieve modular and traceable implementation.

These assumptions sound very desirable if we look at the problems of cross-cutting concerns and tangling of code. However, if we look at the use cases, they tend to have cross-cutting nature themselves. Most of the use cases depend on other use cases, which we also show as separate use cases in inclusions and extensions. These kinds of use cases have dependency on base use cases. Extension use case can only be initiated if the base use case does not perform its basic functionality or performs in an erroneous way. This inter-dependent nature is not easy to justify in design if we consider use cases as separate modules. In implementation also, we will have to have redundant code calling to other use cases if we implement them as separate components. This will again go against the basic purpose of component-based development.

3 Aspect Modeling

In Aspect-Oriented Programming (AOP), we develop a base program while paying equal attention on the development of its cross-cutting concerns (Aspects). We have a number of process models in use for object-oriented development of a program, such as the sequential model, iterative model, spiral model etc. When we extend our object-oriented development to include the development of its cross-cutting concerns, we need an extended version of the model to address the controlled flow of development processes. Aspects need to be developed along with the base functionality, but with different treatments on every level of the development life cycle. Figure 1 shows a proposed model based on this concept of handling Aspects in a process model approach, representing its role and evolution at every level of the development cycle.

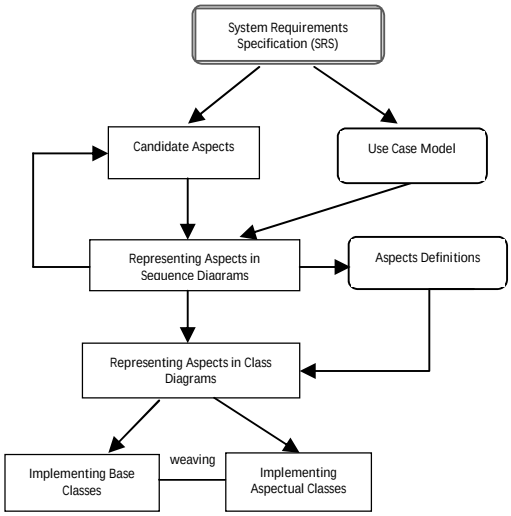

Figure1. AOSD Model

3.1 Identifying Aspects

There are some Aspects which are considered by default with every application, such as performance, security, and fault-tolerance. These Aspects can be pointed out in the System Requirements Specification (SRS). There are also application-related Aspects, such as security Aspects for safety critical systems, fault-tolerance Aspects for systems which are supposed to be running all the time, and synchronization Aspects for systems containing multiple running threads. We can identify these Aspects as candidate Aspects from requirements of the system during the requirements engineering phase. Once we have listed some of the candidate Aspects we can point out those use cases which may have interaction with these cross-cutting concerns (Aspects). For example in an ATM system shown in Figure 2, we can point out that the use case "withdraw cash" will require involvement of a "logging" Aspect, so we can highlight this use case to show that its further design and implementation is going to be affected by the cross-cutting concerns, and it should be handled differently compared to other use cases.

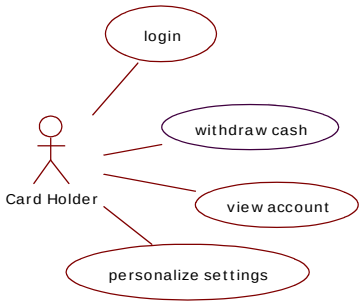

Figure 2. Use case diagram of an ATM system

3.2 Representing Aspects

Proceeding further with the highlighted use cases which have interaction with Aspects, we can now show the communication (message passing) between objects of the base program, and objects of the cross-cutting concerns (Aspects). From the above example, if we draw a sequence diagram of use case "withdraw cash" with Security and Access Control Aspects, we can show the interaction between them by identifying the insertion points of Aspects during the flow of messages. Figure 3 shows how the sequence diagram for the "withdraw cash" use case can be extended with the representation of Aspects and their interaction with the base program. This kind of representation can help not only in representing the Aspects but can also help in identifying characteristics of the Aspects, such as Insertion Points/Join Points, Extension Points and the Points of Calling (BeforeCall and AfterCall).

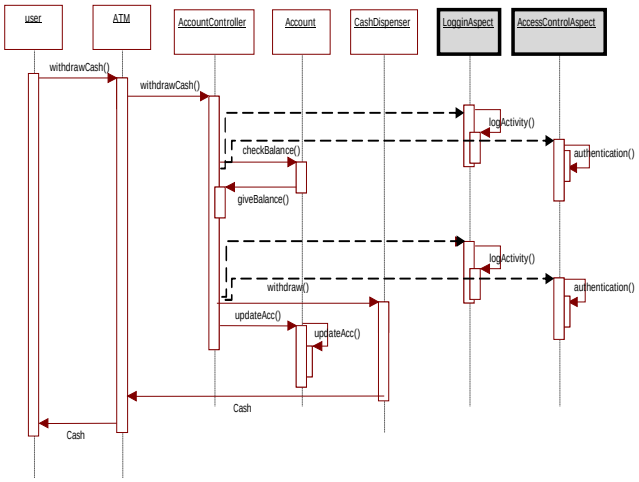

Figure 3. Sequence Diagram of the "withdraw cash" use case with Aspect representation

Such sequence diagrams show the flow of message passing which can later help the software engineer to develop the "advice" definitions of the Aspects. The diagrams also help in identifying the insertion points of the Aspects within a module, which will help with the implementation of the joint points later in the development phase.

Table 1. Aspect attributes

Sequence Diagram 1	
Use Case	withdraw cash
Aspect(s) involved	logging
Insertion Point(s)	checkAccount
Extension Point(s)	verify session

Sequence diagrams showing the interaction of Aspects with base objects in a modular scenario also help us to identify the relationships between Aspectual classes and Base classes. This can then be represented in an extended Class Diagram, a shown in Figure 4.

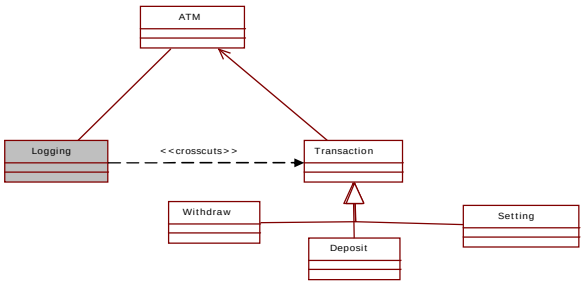

Figure 4. Class diagram with Aspectual Classes

4 Conclusion and Future Work

We have discussed some of the current approaches for identifying and representing Aspects. We have surveyed the literature related to these approaches and provided a constructive critique on the more popular ones. We have presented some of the shortcomings in these approaches, which we feel do not help in identifying and representing Aspects as separate but integrated entities of the system. Finally, we have outlined a proposed Aspect Model which hierarchically shows how Aspects can be identified and modeled alongside the base program, from abstraction to implementation. The emphasis of this model is upon identifying and representing Aspects at all levels of the system development life cycle. The model is followed by identification and representation of Aspects in Use Case Diagrams, Sequence Diagrams, and Class Diagrams of the system. In future, we are planning to work on finding out suitable approaches for identification of Aspects and developing model composition techniques for Aspect-Oriented Programming.

5 References

[1] A. Rashid, A. Moreira, and J. Araujo, "Modularization and Composition of Aspectual Requirements," presented at 2nd International Conference on Aspect Oriented Software Development (AOSD), Boston, USA, 2003.

[2] J. Araujo, A. Moreira, I. Brito, and A. Rashid, "Aspect-Oriented Requirements with UML," presented at Workshop on Aspect-Oriented Modeling with UML (held in conjunction with the International Conference on Unified Modeling Language UML 2002), 2002.

[3] I. Jacobson and P.-W. Ng, Aspect-Oriented Software Development with Use Cases: Addison Wesley Professional, 2005.

[4] Y. Yu, J. C. S. d. P. Leite, and J. Mylopoulos, "From Goals to Aspects: Discovering Aspects from Requirements Goal Models," presented at International Conference on Requirements Engineering, Kyoto, Japan, 2004.

[5] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. XEROX PARC Technical Report, SPL97-008 P9710042. February 1997.

[6] I. Jacobson, "Use Cases and Aspects—Working Seamlessly Together," Journal of Object Technology, vol. 2, pp. 7-28, 2003.

[7] S. M. Sutton, "Concerns in a Requirements Model - A Small Case Study," presented at Early Aspects 2003 Workshop: Aspect-Oriented Requirements Engineering and

Architecture Design (held with AOSD 2003), Boston, USA, 2003.

[8] M. Katera and S. Katz. Architectural views of Aspects. In Proceedings of the International Conference on Aspect-oriented Software Development, pages 1–10, 2003.

[9] M. Saeki and H. Kaiya. Transformation Based Approach for Weaving Use Case Models in Aspect-Oriented Requirements Analysis. In The 4th AOSD Modeling With UML Workshop, Oct. 2003.

[10] X. Wang and Y. Lesperance. Agent-oriented requirements engineering using congolog and i*. In Submitted to AOIS-2001, Bi-Conference Workshop at Agents 2001 and CAiSE'01. 2001.

[11] Araujo, J., Coutinho, P.: Identifying Aspectual use cases using a viewpoint-oriented requirements method. In: Early Aspects 2003: Aspect Oriented Requirements Engineering and Architecture Design, Workshop of the 2nd Intl. Conference on Aspect-Oriented Software Development, Boston, MA (2003)

[12] J. Castro, M. Kolp, and J. Mylopoulos, ''towards requirements-driven information systems engineering: the TROPOS project,'' Information Systems, vol. 27, pp. 365–389, 2002.

[13] A. Rashid, B. Tekinerdo˘gan, A. Moreira, J. Ara´ujo, J. Gray, J. G. Wijnstra, and P. Clements. Early Aspects: Aspectoriented requirements engineering and architecture design. In Workshop at AOSD-2002, 2002.

[14] J. Whittle and J. Araujo, "Scenario Modelling with Aspects," IEE Proceedings- Software Special Issue, vol. 151, pp. 157-172, 2004.

[15] S. Sutton and I. Rouvellou, "Modeling of Software Concerns in Cosmos," in Proceedings. 1st Int' Conf. on Aspect-Oriented Software Development (AOSD-2002), G. Kiczales, Ed., 2002, pp. 127-133.

[16] S. M. Sutton and I. Rouvellou, "Concern Modeling for Aspect-Oriented Software Development," in Aspect-Oriented Software Development, R. E. Filman, T. Elrad, S. Clarke, and M. Aksit, Eds.: Addison-Wesley, 2004, pp.479-505

[17] A. Moreira, J. Araujo, and A. Rashid, "Multi-Dimensional Separation of Concerns in Requirements Engineering," presented at Requirements Engineering Conference (RE 05), Paris, France, 2005.

[18] J. Grundy, "Multi-perspective specification, design and implementation of software components using Aspects," International Journal of Software Engineering and Knowledge Engineering, vol. 20, 2000.

[19] A. Moreira, J. Araujo, and A. Rashid, "A Concern-Oriented Requirements Engineering Model," presented at Conference on Advanced Information Systems Engineering (CAiSE'05), Porto, Portugal, 2005.

[20] E. Baniassad and S. Clarke, "Finding Aspects in Requirements with Theme/Doc," presented at Workshop on Early Aspects (held with AOSD 2004), Lancaster, UK, 2004.

[21] Pawlak, R; Younessi, H.; "On Getting Use Cases and Aspects to Work Together", in Journal of Object Technology, vol. 3, no. 1, January-February 2004, pp. 15-26. http://www.jot.fm/issues/issue_2004_01/column2