ASPECT-ORIENTED DESIGN MODEL

Saqib Iqbal, Gary Allen
University of Huddersfield, Huddersfield, HD1 3DH, UK

ABSTRACT

Designing crosscutting concerns (aspects) is a challenging task. Since crosscutting concerns were not addressed while developing contemporary software design techniques, so they lack support for accommodating representation of such concerns along with base program. Some design languages like UML have been extended to express aspects and their elements but they do not fully represent aspects. Some lack adequate representation of aspect elements and some lack an efficient and reusable composition technique. In this paper, some of the aspect-oriented design techniques have been critically discussed. A proposed aspect model has been discussed which helps in overcoming the deficiencies in the contemporary aspect-oriented design techniques. This model represents aspects and their elements throughout the software development life cycle.

Keywords aspect-oriented programming, aspect-oriented design

1. INTRODUCTION

"Separation of concerns" (Parnas (1972), Dijikstra (1976)) is not a new principle in software development. It suggests that complex systems must be divided into sub-systems, usually called modules, to make them easier to implement and understand. Modules can be developed separately and then be joined together to form the whole system. This property enhances system's evolvability, extensibility and traceability.

The Object-oriented software development paradigm is based upon this principle. Each concern or functionality of system is encapsulated in objects and they only perform their own specified functions without accessing other object's private elements. This is an example of "separation of concerns". Though this property is achieved in object-oriented programming (OOP), over time developers started to feel that OOP does not separate all the concerns in from of classes (Kiczales et al (1997)). There are some concerns such as tracing, logging, fault-tolerance, synchronization, to name a few, which cannot be implemented in distinct classes, rather their code is present in more than one class, usually referred to as the scattered code problem, and such concerns are called cross-cutting concerns. To separate such crosscutting concerns from being scattered, new separation or modularization techniques are required. Two different methods have been used by software developers to handle crosscutting concerns: Interception based approaches, in which event or subroutines to implement crosscutting concerns are intercepted in the execution, and Code Weaving approaches, in which code is woven into the base code with the help of a tool. Based on the later solution, many programming paradigms have been suggested, such as aspect-oriented programming (Kiczales et al (1997)), adaptive programming (Lieberherr (1996)), role-modelling (Reenskaug, World and Lehne (1995)), composition filters (Aksit, Bergman and Vural (1992)) and subject-oriented programming (Aspect team (2001)). These approaches decompose each concern in an individual way.

In this ongoing research, aspect-oriented techniques are being explored. In Aspect-oriented programming, each concern, which has scattered code, is implemented in an individual module, called an aspect. Aspect-Oriented technology provides efficient modularity and serves its purpose as an approach to implement separation of concerns. But because of its original purpose of inventions, all the efforts have been put in the field of its implementation. That is why it lacks support in the earlier phases of AO system development. Especially, there are no formal design frameworks available for the design of AO systems. This work focuses on formulating a framework which could help in designing both base program and aspects.

A popular implementation of aspect-oriented techniques is an extension to Java language, called AspectJ, which provides support to represent aspects and their constructs in the program and also provides interfaces to join aspects with the base program. Aspects comprise of an advice, a piece of code to implement the aspect's logic, and pointcuts, which are packages of related join point. Join points are the points in the base program where an advice has to be woven into the program. We are working on developing design support for aspects and their constructs. Although we are looking at other options too, the most obvious option in case of designing aspects seems to be the Unified Modeling Language (UML). UML is a general-purpose language which gives diagrammatic support for analyzing and designing a system. Both structural and behavioural representation of the system can be captured through UML. Many extensions to UML have

already been suggested to accommodate the design of aspects and their artefacts along with that of base program.

This paper gives a brief literature review of aspect-oriented design and modelling techniques and their possible representation in UML. Some of the important factors to consider while proposing new AO design techniques have been outlined. An aspect-oriented model has also been suggested which helps in identifying and representing aspects in the software analysis and design phases. Section 2 is about aspect-oriented design and proposed techniques by its research community. Proposed techniques of modelling and design have been described in section 3 and section 4 is the conclusion and future work.

2. ASPECT-ORIENTED DESIGN

Design of software is the structural and behavioral implementation of requirement specification. Requirements are shaped into implementable elements, entities or functions. Due to the complexity of contemporary systems, software design must provide support for the abstraction of system elements and separation of concerns. Object-oriented programs (OOP) are usually designed in the unified modelling language (UML). UML provides both behavioural and structural representation of the system. For example, for behavioural representation interaction diagrams and state diagrams are developed and for structural representation object and class diagrams. It also allows the designer to show the abstraction level of the classes of the system. If we evaluate the ability of UML for separation of concerns, we will have to evaluate object-oriented programming first since UML is an object-oriented modelling language. Object-orientation encapsulates the business logic of concerns within objects. Objects are the unit of development in OOP. There are some concerns of the system which are not fully handled in OOP, such as performance, persistence, fault-tolerance, etc. Such concerns affect or have connection with more than one object, thus, their representation and code is scattered over the system. Such scattered nature of code causes tangling problem of code.

Aspect-oriented design (AOD) must allow the designer to design aspects and their elements. It must also be able to develop mechanism to connect join points of aspects to their corresponding points in the base program. In other words, we need an aspect model and a composition technique to weave aspects to the base program. There have been many proposed techniques for AO design. They could be divided into aspect-oriented design techniques and non-aspect-oriented techniques:

2.1. NON ASPECT-ORIENTED TECHNIQUES FOR AO DESIGN

2.1.1. UML

A lot of work has been carried out for designing aspects in UML. As UML is for designing object-oriented systems, so it does not provide full support for aspect-oriented design. Therefore, a number of extensions to UML have been proposed. These extensions help in representing aspect elements like pointcuts, join points, advices and introductions. Besides aspect elements, composition techniques of aspects with base elements are equally important. Composition techniques of aspect-oriented system are not that easy to represent in UML since there is no support available in UML for composing systems through specialized interfaces. In aspect-oriented case, interface of every aspect is different and has to be developed differently. This kind of hindrance in developing interfaces is not easy to accommodate in UML.

Some examples of the use of UML for AOD are Theme/UML (Baniassad and Clarke (2004)), CoCompose, AODM (Stein, Hanenberg and Unland (2002)), AAM (Clarke and Walker (2002)) and AML (Groher and Baumgarth (2004)). The problem with all these technique is that they do not fully accommodate all elements and composition of aspects.

2.1.2. OCL

Object constraint language (OCL) is a subset of UML which allows users to write queries and constraints. OCL has been used in AOD as a mean to select join points depending on the specified constraints in the program's specification. OCL (Stein, Hanenberg and Unland (2004)) is a non-graphical language and it has been used as a way of evaluating a graphical design on the basis of specified constraints. It has been used to define constraints on aspect model as well as constraints on their compositions.

2.2. ASPECT-ORIENTED TECHNIQUES FOR AO DESIGN

2.2.1. AODM

AODM is a technique that helps in the design of aspect and their elements. It extends UML for accommodating aspect elements and their composition. In the beginning, it was intended for expressing aspects being implemented with AspectJ, but over the time, it has evolved to be a generic language, giving support to asymmetric approaches (such as composition filters and adaptive programming) as well. For example, advice is represented in AODM like in figure 1.

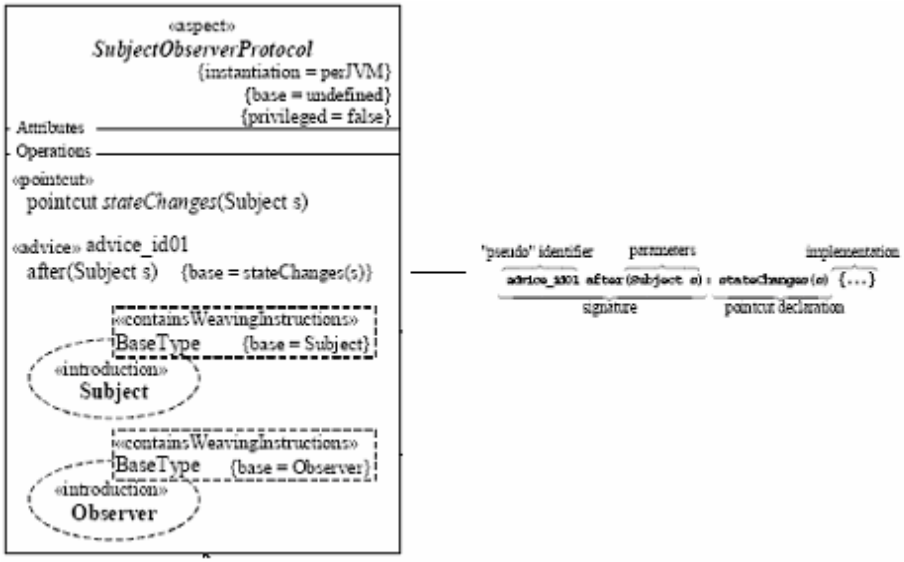

Figure 1: aspect's advice representation in AODM (Stein, Hanenberg and Unland (2002))

2.2.2. THE SUP METHOD

The State charts and UML profile (SUP) (Aldawud, Bader and Elrad (2002)) method supports a process for analysis and design of aspect-oriented systems, using state charts complemented with UML profiles. This method extracts aspect concerns from an Object-Oriented (OO) concern specification by identifying crosscutting state transitions from within an OO model.

3. ASPECT-ORIENTED MODEL

We have discussed some of the AOD techniques in the earlier sections. Similarly, there are many techniques for identifying aspects from requirements specification documents or analysis of system. All these technique serve the purpose to a greater or lesser extent. There is no unanimous approach for identifying and representing aspects which makes the system hard to trace back or evolve.

We have proposed a technique to fill this gap. Our suggested aspect model (Saqib and Gary (2009)) can be an effective way to systematically identify and represent aspects. This does not only help in identifying aspects at the right time but also makes it easy to trace them throughout the development life cycle. Aspect model (figure 2) is a process-oriented model to represent aspects in the development cycle.

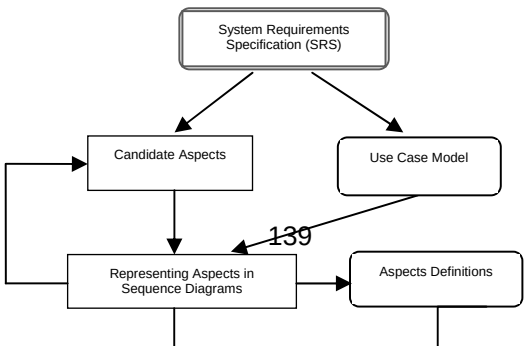

Figure 2: Aspect Model

Identification of aspects is the most challenging task. There are many approaches to identify aspects, such as Viewpoint-based approaches with Arcade model (Lieberherr (1996), Kiczales (1997)), Goal-oriented approaches (Reenskaugh, Word and Lehne (1995)), Use case- and scenario-based Approaches (Aksit, Bergmans and Vural (1992)) and Multi-dimensional separation of concerns (Dijkstra (1976)). Most of these approaches try to identify aspects from either the requirement specification or Use Case model. But we argue that the requirement specification and analysis phase is not the best level of system development to identify aspects. Aspects are crosscutting concerns. Their crosscutting behaviour cannot be observed without looking at the object interactions with each other. In other words, the scattered nature of crosscutting concerns can only be seen in an object model not requirement model. Hence we have proposed to identify aspects from sequence diagrams since they show the objects and their communication with each other (figure 3).

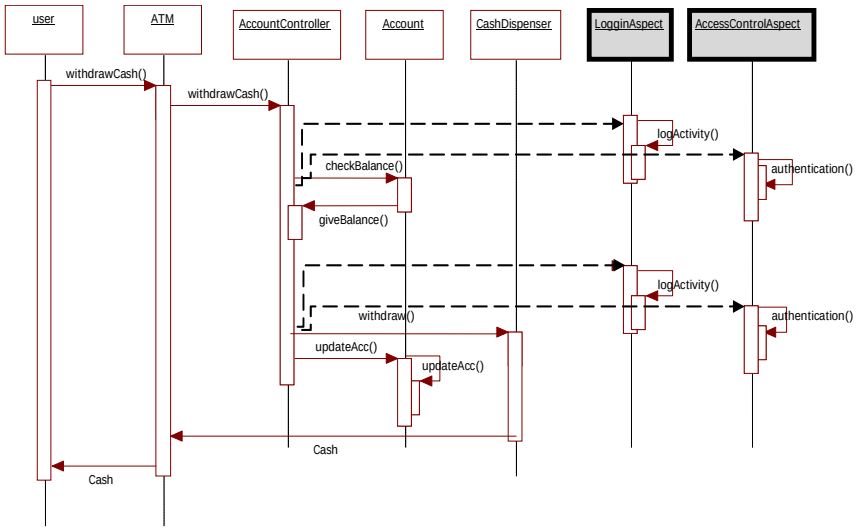

Figure 3: identification of aspects in a sequence diagram

Such a way of identifying and representing aspects can also be shown in the class diagram. In such a way, we can trace back an aspect from the design to requirement specification phase to maintain it and evolve it better.

4. CONCLUSION AND FUTURE WORK

We have discussed some of the most common approaches for aspect-oriented design. We have evaluated them and have presented their pros and cons. We have also given criteria to develop AO design techniques. Finally, we have discussed our approach of identifying and representing aspects throughout the development life cycle. We have argued that our approach cannot only identify all possible aspects from the analysis and design phases but can also provide a means to represent aspects at the key points of the design phase. This technique enhances the efficiency of design and makes it easily traceable and evolvable.

In the future, identification and representation techniques of aspects will be discussed in detail. Composition techniques of aspect-oriented model will also be formulated for aspects to be woven efficiently into the base program.

REFERENCES

Aksit, M., Bergmans, L., Vural, S.: An Object-Oriented Language-Database Integration Model: The Composition-Filters Approach, in Proceedings of the ECOOP 1992, Utrecht, The Netherlands, June 29 - July 3, 1992, pp. 372-395, in: Lecture Notes in Computer Science,vol. 615, Springer, 1992

Aldawud O., A. Bader, and T. Elrad, "Aspect-Oriented Modeling: Bridging the Gap between Implementation and Design," presented at Generative Programming and Component Engineering Conference (GPCE), Pittsburgh, PA, USA, 2002.

AspectJ Team, The AspectJ Programming Guide, http://aspectj.org/doc/dist/progguide/index.html, September 2001

Baniassad E. and S. Clarke, "Theme: An Approach for Aspect-Oriented Analysis and Design," presented at International Conference on Software Engineering, 2004

Dijkstra, E., A Discipline of Programming, Prentice Hall, Englewood Cliffs, New Jersey, 1976

Groher I. and T. Baumgarth, "Aspect-Orientation from Design to Code," presented at Workshop on Aspect-Oriented Requirements Engineering and Architecture Design (held with AOSD 2004), Lancaster, UK, 2004.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, Ch., Lopes, Ch.V., Loingtier, J.-M., Irwin, J., Aspect-Oriented Programming, in: Proceedings of ECOOP 1997, Jyväskylä, Finland, June 9-13, 1997, pp. 220-242, in: Lecture Notes in Computer Science, vol. 1241, Springer, 1997

Lieberherr, K.J., Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns, PWS Publishing Company, 1996

Parnas, D.L., On the criteria to be used in decomposing systems into modules, Communications of the ACM, vol. 15(12), December, 1972, pp. 1053 – 1058

Reenskaug, T., Wold, P., Lehne, O.A., Working with Objects: The OORam Software Engineering Method, Manning/Prentice Hall, Upper Saddle River, New Jersey, 1995

S. Clarke and R. J. Walker, "Towards a standard design language for AOSD," presented at 1st International Conference on Aspect-Oriented Software Development (AOSD 2002), 2002.

Saqib Iqbal, Gary Allen, "Representing Aspects in Design," tase, pp.313-314, 2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering, 2009

Stein D., S. Hanenberg, and R. Unland, "A UML-based Aspect-Oriented Design Notation For AspectJ," presented at Aspect-Oriented Software Development (AOSD 2002), Enschede, The Netherlands, 2002.

Stein D., S. Hanenberg, and R. Unland, "Query Models," presented at Unified Modeling Language (UML) - Modeling Languages and Applications, Lisbon, Portugal, 2004.