Deriving design aspects from conceptual models

Bedir Tekinerdogan & Mehmet Aksit

TRESE project, Department of Computer Science,
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands.
email: {bedir | aksit}@cs.utwente.nl, www server: http://wwwtrese.cs.utwente.nl

Abstract. Two fundamental issues in aspect orientation are the identification and the composition of aspects. We argue that aspects must be identified at the requirement and the domain analysis phases. We also propose a mechanism for gradually composing aspects throughout the software development process. We illustrate our ideas for the design of a transaction framework.

1. Introduction

Software components can be defined as programming language abstractions. Examples of software components are procedures, data structures and objects. Programming languages are vehicles to express abstract executable mechanisms. Components can be identified by the use of heuristic rules in conventional methods. For example in OMT [Rumbaugh 91] tentative classes are identified by looking for nouns in a problem statement. The identified components are composed by means of the object-oriented composition mechanisms, such as inheritance, aggregation and association. Object-oriented methods define a number of heuristic rules to identify relations among components.

Similar to object-oriented design, in aspect-oriented design, two issues appear to be important: identification of the abstraction models, that is, aspects and the composition of these aspects, or aspect weaving [Kiczales et. al 97]. We will focus on these two issues in this paper. Regarding the aspect identification we argue that like objects, aspects should be identified during the requirement and the domain analysis phases. We will also discuss an approach in which aspects are gradually composed along the software development process.

The outline of this paper is as follows: Section 2 will elaborate on our approach for aspect identification and aspect composition. Section 3 will give our conclusions.

2. Aspect identification

2.1 Where to look for?

Software development can be seen as a problem solving process in which the requirements represent the problem for which a programming solution is required. A software development process involves a number of steps, which produce various kinds of software artifacts. These steps can be considered as transitions between artifacts. No doubt, the early phases of the software development process includes concerns, which have a major impact on the final structure and quality of the software [Aksit 97]. We therefore, believe that aspects appear beyond the programming level and as such the identification of aspects should start at the levels of requirements and domain analysis phases.

2.2 How to identify?

Here the fundamental question is, given a problem domain like for example the transaction domain, how should we abstract and identify the aspects? To address this issue, we propose the method described in Figure 1, which defines the basic steps needed to identify aspects in the earlier phases of the software development process. This method will be described in more detail in subsequent sections.

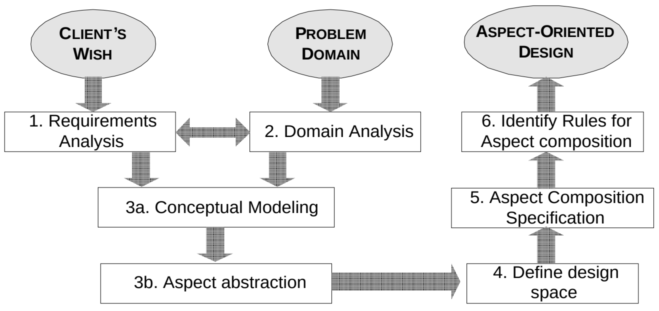

Fig. 1. The aspect modeling method

2.2.1 Requirements Analysis

The first step in software development is the requirements analysis phase. The goal of the requirements analysis phase is to understand and capture the exact needs of the clients of a software system [Wieringa 96]. Requirements analysis deals with eliciting, analyzing and capturing the requirements of the client for which the software system is developed.

2.2.2 Domain Analysis

Domain analysis aims at systematically identifying, formalizing and classifying the knowledge in problem domain in a reusable way [Arrango 94]. The basic steps of domain analysis are the identification of the knowledge sources, the data collection from the knowledge sources, data analysis of the extracted knowledge and knowledge modeling. Domain models are mainly derived by considering commonalities and variations between the retrieved data. The basic difference between requirements analysis and domain analysis is that requirements analysis focuses on the requirements of one application whereas domain analysis attempts to model the knowledge of a wide range of related applications. The deliverables of domain analysis are a set of domain models, relations and rules that are common in a problem domain for the corresponding applications. Domain models can be represented in many ways, e.g. ER diagrams, object-oriented class diagrams, or just ordered text. In [Aksit et. al. 98] we applied domain analysis techniques to support the development of stable frameworks.

2.2.3 Conceptual modeling

Requirements analysis extracts the potential aspects. Domain analysis collects knowledge about these aspects. The conceptual modeling process elaborates on the domain model to define canonical models. Canonical models are similar to concepts of the classical view [Smith & Medin 81]. Concepts are not chosen arbitrarily but are formed by abstracting the knowledge about instances. An identified concept is useful if it has meaningful differences with the existing concepts. The meaningfulness on its turn is defined by the context.

An example: Aspect modeling for Adaptable Transaction Systems

We applied our above ideas for aspect identification and aspect modeling in a pilot project which aims at designing an object-oriented atomic transaction framework to be used in a distributed car dealer management system [Tekinerdogan 96]. After the requirements analysis and domain analysis phases we could extract four basic groups of aspects. Aspects related to the transaction models [Elmagarmid 92], aspects related to quality factors such as adaptability [Adaptability 96] and performance and aspects related to the object model. We developed conceptual models for all these aspects.

2.2.4 Define design space

Even though we may not know about individual designs it is convenient to talk about design spaces. We define a design space as a set of descriptions of possible designs. The identified concept models represent the dimensions of such a design space. Our concept of design space is similar to the concept of information space described in [Jacobson 92]. In [Aksit & Tekinerdogan 98] we elaborate on the concept of design space and more specific on the concept of design algebra. For the adaptable transaction domain the design space is as follows:

Design Space = (Transaction x ObjectModel x Object Coupling x Adaptability x Performance)

Each element in this design space represents a design solution for the given problem domain. Since each basic concept is composed of sub-concepts the design space is very large.

2.2.5 Aspect composition specification

Basically there are two ways for aspect composition. Composition at-once or gradual composition. In the composition at-once approach, one aspect composer composes all the identified aspects into to the final realization model. The problem with this approach is that the aspect composer needs to deal with all the aspects at once which may be a difficult, error-prone and time-consuming process. In addition, not all the combinations of the design space may be possible or useful. It is therefore, not necessary to elaborate on all the elements of the design space. Accordingly, we need some mechanisms to restrict this large design space and exploit only the useful combinations. In order to meet this requirement dedicated aspect composers will be used in our approach. These aspect composers are used to gradually explore the useful combinations in the design space. For example in the transaction system application we adopt a *AdaptabilityComposer* which composes a domain model aspect with the adaptability aspect. The basic issue of the use of multiple dedicated aspect composer is the ordering of the composition of aspects. If we have 6 aspects we can apply the gradual aspect composition in 6!= 720 ways. This is a difficult task. We therefore apply some general rules to manage this situation. The most intuitive ordering is to start with the domain models and end with the component models. From the resulted space a new model is selected which includes the useful and desired combinations. This process is iterated until we have included all the aspects and domain models. The final result of this process is a realization model. The realization model includes all the elements, which defines the final implementation for the design problem.

2.2.6 Define aspect composition rules

After we have determined the ordering of the aspect composers we must define the rules which will be applied in each specific aspect composer. In [Aksit & Tekinerdogan 98] we have described this process in more detail.

Conclusion

In this paper we have proposed an approach for identification of proper aspects from the domain analysis and requirements analysis phases. The aspect composition can be done centrally by one composer or gradually by multiple composers along the software development process. We illustrated the practical applicability of gradually composing aspects.

References

[Adaptability 96] M. Aksit, B. Tekinerdogan, L. Bergmans, K. Lieberherr, P. Steyaert, C. Lucas, & K. Mens, ECOOP '96 Adaptability in Object-Oriented Software Development Workshop, url: http://wwwtrese.cs.utwente.nl/ecoop96adws/, 1996.

[Aksit 97] M. Aksit. *Issues in Aspect-Oriented Programming*, Position paper, AOP workshop, ECOOP '97.

[Aksit & Tekinerdogan 98] M. Aksit & B. Tekinerdogan, *Models for Composing Design Aspects*, University of Twente, Department of Computer Science, 1998.

[Aksit et. al. 98] M.Aksit, B. Tekinerdogan, F. Marcelloni., & L. Bergmans. Deriving Object-Oriented Frameworks from Domain Knowledge. To be published as chapter in M. Fayad, D.Schmidt, R. Johnson (eds.), Object-Oriented Application Frameworks, Wiley, 1998.

[Arrango 94] G. Arrango. *Domain Analysis Methods*. In *Software Reusability*, Schäfer, R. Prieto-Díaz, and M. Matsumoto (Eds.), Ellis Horwood, New York, New York, 1994, pp. 17-49.

[Elmagarmid 92] A. Elmagarmid (ed), *Database Transaction Models for advanced applications*, San Mateo, CA, Morgen Kaufmann, 1992.

[Jacobson 92] I. Jacobson, M. Christerson, P. Jonsson & G. Overgaard, *Object-Oriented Software Engineering - A Use Case Driven Approach*, Addison-Wesley/ACM Press, 1992.

[Kiczales et al. 97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J-M. Loingtier and J. Irwin, *Aspect-Oriented Programming*, ECOOP'97 Conference proceedings, LNCS 1241, June 1997, pp. 220 – 242.

[Smith & Medin 81] E.E. Smith & D.L. Medin. *Categories and Concepts*, Harvard University Press, 1981.

[Tekinerdogan 96] B. Tekinerdogan. Requirements analysis of transaction processing in a distributed car dealer system. Technical report. University of Twente, 1996.

[Wieringa 96] R.J. Wieringa. Requirements Engineering: Frameworks for understanding, Wiley, 1996.