The Role of Aspects in Domain Engineering

Article · September 2005

CITATIONS
2

READS
20

3 authors:

Harvey Siy
University of Nebraska at Omaha
75 PUBLICATIONS **1,604** CITATIONS
SEE PROFILE

Mansour Zand
University of Nebraska at Omaha
47 PUBLICATIONS **128** CITATIONS
SEE PROFILE

Victor L. Winter
University of Nebraska at Omaha
85 PUBLICATIONS **387** CITATIONS
SEE PROFILE

The Role of Aspects in Domain Engineering

Harvey Siy, Mansour Zand, Victor Winter
Department of Computer Science*
University of Nebraska at Omaha
{hsiy,mzand,vwinter}@mail.unomaha.edu

August 19, 2005

Abstract

Aspect-oriented domain engineering is a promising extension to the domain engineering process. We present several lines of inquiry that demonstrate how domain engineering benefits from adopting aspect-oriented software development concepts.

1 Introduction

The product line approach[9] to software development is gaining popularity as organizations recognize the fact that they are oftentimes repeatedly creating similar applications or implementing similar features. For example, in telecommunication switches used to route telephone calls from one party to another, different telecommunications service providers often have different, yet similar needs. They need to get calls from A to B, and, sometimes, to C. They need functionalities to add, remove and bill customers. On the other hand, they differ due to country-specific regulations, required feature sets, as well as in the existing environments that need to interact with the switches. The switching equipment provider can benefit from producing the same switching hardware with multiple customizations of the software. Through an analysis of the economies of scale, it can be shown that a product line approach to producing and maintaining switching software is a cost-effective way of managing the needs of multiple customers.

A major problem of software design and development is effecting separation of concerns, especially when concerns crosscut each other. Taking the same example from the telecommunications domain, subscribers are offered a variety of features such as caller ID, call waiting, voice mail, busy line transfer, 3-way calling, emergency services, and so on. These features are required to work on traditional phones, newer packet-based phones, and mobile phones, which use different signalling protocols to set up the call. There are also more basic features such as digit-analysis which determine when certain features are activated and when, depending on the state of the call when certain digits are pressed. These features are all somehow expected to work independently and transparently of the actual type of phone. Billing adds another dimension of concerns that includes, but is not limited to, knowledge of the call type, call medium, call duration, as well as feature usage. On top of these, are so-called non-functional requirements such as capacity, performance, synchronization, reliability, high availability, and flexibility. These concerns are usually spelled out as different policies in high level documents, but end up being interspersed into call processing code during implementation. We can imagine that the problems are compounded when different members of the switching software product line share these cross-cutting concerns in varying degrees. Clearly, mechanisms that can effect separation of concerns are of great value in this broad domain.

In this paper, we examine how domain engineering technology can be extended with aspect-oriented software development technology to address the issues of software product line development and separation of concerns. We also discuss the creation of a concern classification scheme.

*This work is supported by the UNO Center for Aspect-Oriented Design and Development

2 Domain Engineering

One of the earliest steps in software product line development is domain engineering. Domain engineering is a methodology for building a domain-specific application development environment for an entire software product line. A *domain* is defined to be the space of all possible applications for that product line, as specified by their commonalities and all the ways in which they could vary. It provides formalisms for strategic reuse of designs and implementations by providing a development environment based on the specified commonalities and variabilities of the domain. This leads to fast and efficient creation of product line member applications.

3 Aspect-Oriented Software Development

Aspect-oriented programming[4] addresses the problem of separation of concerns using the mechanisms of obliviousness and quantification[6]. Quantification is the ability to specify properties of the modules on which to apply a particular aspect. Obliviousness refers to the notion that the modules themselves are not aware of the aspects being applied to them. This is also known as implicit invocation. In an aspect-oriented language such as AspectJ[7], quantification and obliviousness are realized through the constructs of *join points*, *advice* and *pointcuts*. *Advice* are methods that define behaviors that need to be inserted in order to realize a concern. *Join points* specify where advice can be inserted, and *pointcuts* are used to specify join point sets to which advice should be applied.

4 Aspects and Domain Engineering

Much research has gone into aspect-oriented programming and programming languages[1][5]. Recently it has been realized that these concepts can also be applied earlier in the development life cycle[3][2]. These are called Early Aspects.

Synergies have been recognized in the domain engineering phase of product line development. Aspect-oriented concepts extend the power of domain engineering in many ways. For instance:

1. **As an enabling technology for realizing the application engineering environment.** Domain engineering does not deal specifically with separation of concerns in the implementation of the application environment or the creation of instances of applications. This is left as an implementation detail. In this context, aspect-oriented software development fits in nicely as a technology enabling separation of concerns in the implementation of the application engineering environment. In particular, a more precise implementation of aspect quantification can be realized with domain-specific constructs for specifying the analogues of advice, join points and pointcuts.
2. **As a contributor to maintainability of product line members.** A key advantage of aspect-oriented programming is the ability to localize changes, thus preserving modularity of the design. As is the case with all successful software products, software product line members need to evolve as well. One option commonly employed is to simply regenerate the application member with the new requirements. However there are cases where this is not feasible especially if the application generator is not equipped to handle such changes, but the changes can be implemented directly on the source code. Thus, the intrinsic advantage of aspect-oriented development in localizing changes makes these changes easier.
3. **As a contributor to maintainability of the product line definition.** As new applications are created, it is almost inevitable that an application will come along whose requirements put it just outside the currently defined domain space. This happens due to an incomplete understanding of the real domain. In any case, the commonality and variability analysis needs to be refined to cover this new understanding of the domain. The change could cut across several concerns. Without explicit separation of concerns, the necessary modifications could turn out to be messy, with many scattered changes. By explicitly forcing the separation of concerns during the original domain analysis, the chances are improved that future changes are localized to one concern.
4. **As an integration mechanism for domain analysis.** Domain analysis is useful for decomposing requirements into separate concerns. However it provides little support to specifying how the environment or application is to be put together while satisfying all these concerns. Through domain analysis, concerns can be identified as independent policies, and commonalities and variabilities can be defined within each policy. An integration strategy can be defined using the aspect concepts of join points and point cuts.

Specificity	Usage	Functional Stability	Potential Classification	Example
Domain	Multiple	Stable	domain-specific, cross-cutting, stable concern	Specific security checking of on-line transaction
Domain	Multiple	Variable	domain-specific, cross-cutting, varying concern	Security checking of on-line transaction using different fields
Domain	Single	Stable	domain-specific, localized, stable concern	
Domain	Single	Variable	domain-specific, localized, varying concern	
Generic	Multiple	Stable	generic, cross-cutting, stable concern	Log in
Generic	Multiple	Variable	generic, cross-cutting, varying concern	Log in using different identifiers
Generic	Single	Stable	generic, localized, stable concern	Encryption algorithm
Generic	Single	Variable	generic, localized, varying concern	Authentication library

Table 1: Concern Classification Scheme

5. **As a notation for reasoning about variabilities in the integration process.** An additional dimension of complexity with product line development occurs when cross-cutting concerns have varying requirements from member to member. By having an integration mechanism, we than have a notation for describing the commonalities and variabilities of the integration strategy itself.
6. **As a means of understanding cross-cutting "non-functional" requirements in a product line.** Non-functional requirements, such as performance, are often left to be verified after the implementation is finished[10]. This is partly because these requirements are defined very vaguely in the domain analysis and partly because we don't know how to work with them except to say, "keep an eye on the performance issue." Further work is needed to examine these non-functional requirements individually and discussing what it means to define it as an aspect. For example, in building a product line of routing software for core network routers, what does it mean to define quality of service as a concern?

5 Discussion

The last question highlights a need to better understand what constitutes a concern. Identification and separation of concerns lies at the core of domain engineering for successful product line development. In its most general form, it refers to the ability to identify and encapsulate those parts of software that are pertinent to a particular concept, goal, task, or purpose[8].

One of the main benefits of identification of concerns in these phases is to help tag those concerns that have the potential of becoming aspects of a product line. As a result, we are developing a framework to identify and classify early aspects in the domain analysis phase.

A simplified version of this schema is presented in Table 1. This schema examines the concerns against three criteria. They are *specificity*, *scope of usage*, and *stability*.

Specificity refers to whether the concern is a generic one or is domain-specific. This dictates whether

the solution warrants the creation of a domain-specific language. Take, for example, the ability to login. In this case, it is a generic concern shared by many different types of software applications in different industries. However, some industries require a higher level of security, like e-commerce. Hence, security checking of online transactions is a domain-specific concern.

Scope of usage refers to the ease of localizing a concern. Cross-cutting concerns affect multiple concepts, tasks or functional areas. This determines the need for an aspect-oriented solution. Taking the same example, the ability to login can crosscut many functionalities, as in the case when repeated authentications are required due to session timeouts.

Stability refers to the concern's potential for variability. If the concern has the same characteristics across multiple products or applications, it can be developed as a single application or library. Otherwise, additional investment in effort is needed to go to the product line approach.

Using the outcome of this schema the concerns are temporarily documented as potentially being one of the given classifications and further decisions can be made regarding the need for product line development and/or aspect-oriented technology.

This table only provides a few examples and we are working to add additional criteria to the table to smooth the progress of identifying and classifying concerns. Among other criteria that we are considering to include to this table are functional vs. non-functional, interacting vs. orthogonal, etc.

6 Summary

Concerns can arise at any point during the software life-cycle and are not only limited to implementation and programming. The early phases in the software life-cycle typically include domain analysis, requirements analysis, and architecture design. Concerns that are identified in these phases are called *Early Aspects*. One of the main benefits of identification of concerns in these phases is to help tag those entities that may become aspects of a product line. We are developing a framework to identify and classify early aspects in the domain analysis phase.

References

[1] AOSD.net. `http://www.aosd.net`.

[2] J. Araújo, E. Baniassad, P. Clements, A. Moreira, A. Rashid, and B. Tekinerdoğan. Early aspects: The current landscape. Technical Report CMU-SEI-2005-TN-xxx, Carnegie Mellon University, Pittsburg, PA, 2005. Also available as `http://trese.cs.utwente.nl/workshops/early-aspects-AOSD2005/Papers/EarlyAspects-LandscapePaper-FirstDraft-2005.pdf`.

[3] Early Aspects.net. `http://www.early-aspects.net`.

[4] T. Elrad, R. E. Filman, and A. Bader. Aspect-oriented programming. *Communications of the ACM*, 44(10):29–32, October 2001.

[5] R. E. Filman, T. Elrad, S. Clarke, and M. Akşit, editors. *Aspect-Oriented Software Development*. Addison-Wesley, 2005.

[6] R. E. Filman and D. P. Friedman. Aspect-oriented programming is quantification and obliviousness. In R. E. Filman, T. Elrad, S. Clarke, and M. Akşit, editors, *Aspect-Oriented Software Development*, chapter 2. Addison-Wesley, 2005.

[7] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Budapest, Hungary, 2001.

[8] P. Tarr and H. Ossher, editors. *Workshop on Advanced Separation of Concerns in Software Engineering at ICSE 2001*, Toronto, Ontario, May 2001.

[9] D. M. Weiss and C. T. R. Lai. *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison Wesley, 1999.

[10] L. Xu, H. Ziv, D. Richardson, and Z. Liu. Towards modeling non-functional requirements in software architecture. In *Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop*, Chicago, IL, March 2005.