Using Aspect-Oriented Approach for Software Product Line Development

Lei Tan and Yuqing Lin
School of Electrical Engineering and Computer Science, University of Newcastle, Callaghan, NSW, Australia
lei.tan@uon.edu.au, yuqing.lin@newcastle.edu.au

Keywords: Software Quality, Software Product Line, Aspect-Oriented, Crosscutting Concern.

Abstract: Software Product Line Engineering (SPLE) is a software development paradigm to improve systematic software reuse. SPLE is intended to develop a set of similar software systems which share great commonalities within a particular application domain. There are two key assets underpin Software Product Line (SPL) development: feature model and reference architecture. To deal with complex crosscutting behaviors in SPL and also manage the impact of Non-Functional Requirements (NFRs), we propose an aspect-oriented framework in this paper. The proposed framework is able to improve the modeling of interrelationships between design factors and representation of the variabilities in product families. We introduce a small case study to illustrate our approach at the end.

1 INTRODUCTION

In software engineering, reusing software and its components becomes a practical solution towards cost-effective and quality-oriented software development. Software Product Line Engineering (SPLE) (Bosch, 2000) is one of software engineering approaches that focuses on providing systematic software reuse on software families, where a product family is a collection of software systems that share great commonalities in a particular application domain.

Different from other software reuse techniques, SPLE keeps design with reuse in mind. All assets developed will be reused to produce member products based on different requirements. So SPLE contains various reusable assets and artifacts, such as system requirements, source code, architecture and documentation. When developing a member product of the family, these reusable artifacts will be refined, modified and configured. SPLE saves the efforts of developing every single software system from scratch. SPLE also provides systematic mapping and traceability to ensure that requirements will be realized by final products. Once the reusable artifacts are defined and successfully implemented, the individual member products can be produced through a very organised customization.

The remainder of the paper is organised as follows. In section 2, we will introduce background of SPLE and some open problems. In section 3, we propose a comprehensive framework that dealing with these problems by using aspect-oriented techniques. In sections 4, we present a case study to demonstrate how we develop the framework and achieve aspect identifications and mappings. Section 5 concludes the paper and discusses future works.

2 BACKGROUND

2.1 Software Product Line Engineering (SPLE)

SPLE consists of two processes, domain engineering and application engineering. The tasks of establishing a Software Product Line (SPL) belong to domain engineering and the processes developing a particular member product from an SPL are included in application engineering. For SPLE, it is very important to understand where the variations are among the member products, and also potential changes to a product family in the future. This information should be well represented and managed to provide the specific scope of a domain-based product family.

In domain engineering, there are two key reusable artifacts: feature model and reference architecture. These two artifacts are significant in SPL as they are the basis for SPL establishment and products derivation.

2.1.1 Feature Model

A feature model represents all the possible member products of an SPL in terms of features and the relationships among features. In a feature model, features are prominent and distinctive system requirements or characteristics in an SPL (Lee et al., 2002). Features are not existing independently, there are different relationships between features. For example, "requires" and "excludes" are two kinds of common feature relationships. Feature model is used to configure member products by selecting desired features based on customers' requirements and feature relationships.

2.1.2 Reference Architecture

Another important artifact derived based on feature modeling is called reference architecture. A reference architecture (Eixelsberger et al., 1998) is a software architecture that provides the common structures, components and their relationships to the existing systems in a particular domain or an SPL. In SPL, a reference architecture describes the structures and respective elements and relations for the whole product family, i.e. for multiple products in the product line. So it stresses the commonalities of the architecture and also describes the planned variabilities. An important character of the reference architecture is that it is configurable. Based on the requirements on the product, the reference architecture provides templates for developing concrete architectures for the members in the product line.

2.1.3 Open Problems

Some open problems of current SPLE have been identified in both feature modeling and reference architecture development.

- **Some Complex Feature Relationships Are Hard to Manage.** One-to-one relationships are modeled effectively in feature model. The more complex feature relationships are hard to handle by current approaches. For example, various feature groups, which represent different levels of product quality, are hardly modeled by "requires" and "excludes" relationships. As a result, feature selections in configuration become error-prone and the quality of configured product is barely preserved.
- **Quality Attributes Are Not Modeled Systematically in Feature Model.** Feature model is produced from requirement engineering based on requirements documents from the software family. However, the relationships between quality attributes and features are not modeled in a systematic way. The impact of quality attributes on features should be identified to demonstrate the transformation from NFRs to quality attributes and how NFRs are achieved by final products.
- **Transformation from Feature Model to Reference Architecture Is Not Straightforward.** Feature model is a foundation for reference architecture development. However, in most of current approaches, the mechanisms of systematic feature mappings to components are not clearly established. This information should be represented explicitly to enhance SPLE in terms of efficient product derivation.
- **Variability Representation Is Not Adequate.** A well-defined variability representation is needed in both feature model and reference architecture, as it is critical to product development. All the desired features for products and the possible variations to reference architecture should be described in an easy understandable way. Furthermore, the impact of NFRs on feature selections and architecture should also be explicitly represented to facilitate product derivation.

Aspect-oriented approaches have the ability to model the crosscutting relationships among multiple features and components. This is an ideal solution to address mentioned problems in SPLE.

2.2 Aspect-Oriented Software Development (AOSD)

Aspect-Oriented Software Development (AOSD) (Brichau et al., 2008) is to modularize crosscutting concerns in order to provide advanced implementation structures. In software development, crosscutting concerns are those broadly-scoped features or properties that often crosscut several other modules in software systems. By representing crosscutting concerns (or aspects) as first-class abstractions, Aspect-Oriented (AO) approaches are able to address modularity problems that are not well handled by other approaches.

For example, products in an SPL might have different levels of security and performance, i.e. different NFRs, which normally have impact on multiple features and the way the features are implemented. So, the impact of NFRs on features should be managed to improve SPLE. As the nature of AOSD is to modularize aspects and provide better representations of complex relationships, it is appropriate to adapt this idea to deal with existing messy relationships between features and NFRs. Aspect-oriented modeling ap-

proach will further facilitate mappings from requirement level to architecture level.

Currently, there are some AO methods focusing on feature modeling and architecture development in SPL. For example, the framework (Conejero and Hernandez, 2008) proposed is to identify crosscutting features at early stages to reduce the difficulty of products evolution and derivation in the latter stages. This framework is based on a crosscutting pattern and uses traceability matrices to perform the analysis of crosscutting. To identify crosscutting features, the crosscutting features analysis need to be conducted and both common and variable assets of product lines need to be acquired. In this approach, NFRs are able to be inserted into a crosscutting matrix. However, the focus of this approach remains on early aspect of requirement level and it does not specify how to use these information for implementation.

Aspect-Oriented Generative Approach (AOGA) (Kulesza et al., 2004) is an architecture-centric approach that has been implemented to support concern modularization. The main consideration of the approach is to improve the domain modeling and architecture specification of crosscutting features from early development phases. In the problem space, feature model is extended and described by a new domain specific language to collect both non-crosscutting and crosscutting features. In the solution space, software architecture is designed by using aspect-oriented abstractions to model features and using programming languages to implement components of the architecture. However, this approach is specifically dealing with multi-agent systems and the capacity of the approach is limited.

3 PROPOSED FRAMEWORK

We propose an AO-based framework to better model crosscutting features and NFRs in domain engineering. Compare to other AO approaches, our framework contains a particular significance. Our approach starts from feature model at requirement level and provides traceability to NFRs and crosscutting concerns. It specifies the impact of NFRs onto systems and member products, so quality of member products is better modeled. We also expect to have appropriate mechanisms to achieve systematic mapping and transformation from requirements to architectures.

The proposed framework is composed into two parts: AO feature model and AO reference architecture development. The main expectation of AO feature modeling is to convert conventional feature model into four elements, i.e. NFRs, concrete features, aspectual features and aspectual concerns. We also expect to define impact of aspectual concerns on both concrete and aspectual features. The goal of AO reference architecture is to specify the mappings from AO feature model onto reference architecture. The crosscutting concerns at architecture level will be identified and the impact on the reference architecture will be presented.

3.1 Aspect-Oriented Feature Model

To develop AO feature modeling, we first model NFRs and quality attributes, and specify the relationships between features and related quality attributes by using NFR Framework (Chung et al., 1999). Then we moves to functional requirement modeling by using use case diagrams and activity diagrams. In our approach, we extent conventional use case and decompose it into business flows and sub-processes. We are interested in systems' internal processes and responsibilities. To apply use case models in product line, the branch lines that under different conditions should be considered and investigated. If a particular system behavior existing in multi-business processes interacts with many other user visible functionalities of the system, this would be a candidate of aspectual feature.

To better understand AO feature model, we divide features into several categories as following:

- **Concrete Feature.** Concrete feature represents concrete functionalities of a product family. These features represent the fundamental services the systems provide. In AO context, these features are crosscut by concerns.
- **Aspectual Feature.** Aspectual features also represent the functionality of a product line, and these features have various relationships and crosscut other features.
- **Aspectual Concerns.** Aspectual concerns are the key requirements and system considerations crosscutting the concrete features and aspectual features. The aspectual concerns map the NFRs and requirements on qualities to the features in the feature model. The concerns describe the impact of NFRs and quality requirements on the system composition and the way features interact to each other.

According to these feature categories, the tasks of AO feature modeling include developing related models to treat crosscutting features (functional) and concerns (non-functional) in a systematic way. The steps are roughly described as follows:

- The first step is to model concrete features, which is relatively easy as concrete features correspond to the user visible functionalities of the system. Current use case modeling and scenarios based approaches are able to address the functional requirements of the software systems. The output of this step is a model which is able to clearly represent the concrete features and relationships between these concrete features.
- The second step is to identify and modularize aspectual concerns, which is one of the major contributions of our framework. Aspectual concerns could be identified in NFR modeling process in order to describe the impact of NFRs onto the functional features. In the framework, the contribution of each concrete feature to satisfy the NFRs will be examined so that the related quality levels can be specified by these options.
- The last step of AO feature modeling is to identify aspectual features and map aspectual concerns to concrete features and aspectual features. An aspectual feature is a feature that crosscutting other concrete features, i.e. impact the behaviors of other features. They are additional responsibilities that do not affect the main business flow.

By the end, we will convert feature model into quality attributes and other three components, i.e. concrete features, aspectual features and aspectual concerns. The impact of aspectual concerns on both types of features will be identified as well. Moreover, the complex feature relationships will be decomposed into relationships among these three components and NFRs.

3.2 Aspect-Oriented Reference Architecture

Aspect-oriented reference architecture framework is to specify how features and aspectual concerns from AO feature modeling are mapped onto reference architecture. The key point is to identify crosscutting concerns caused at architecture level and how reference architecture is affected and composed.

- The first task is to model functional requirements by developing components and subsystems to implement concrete features. Conventional SPL approaches can be used to transform concrete features into components or subsystems to meet functional requirements. The output of this task will be components and connectors to link these components.
- The next step is to develop a set of architecture scenarios (Rommes and America, 2006) to describe systems. From use case and business processes identified, we are able to work out a set of scenarios describing the internal behaviors of systems. Since the requirements are from multiple products, it is common that we have multiple alternative flows, which suggest alternative internal workflow of the system and will be mapped to parallel scenarios.
- The following step is to identify variabilities of components and possibly pointcut on the architecture components. Components of architectural model from the first step are involved, so it is possible to specify the responsibilities of these components and their interactions. Here we need to examine how the aspectual concerns are addressed in the architecture. These aspectual concerns suggest how the components interact and possibly ways the component crosscutting each other for satisfying functional and non-functional requirements at different levels.
- The final step is to assess scenario interactions to refine possible bigger scope crosscutting concerns. Scenarios could be combined together to implement more complex functionalities, so examining the interactions between scenarios could address some complex aspectual concerns. After this step, all the identified concerns should be properly address in the reference architecture in terms of components crosscutting relationships.

4 CASE STUDY

In this section, we will present a small case study to illustrate how to identify aspects from requirement level and map them to architecture level in our framework. The case study used is a crisis management system product line (Kienzle et al., 2010). To better demonstrate our approach, we also did some adaptations to fit this case into our framework, e.g. adding some features, expanding use case mapping diagrams, developing alternative flows of some use cases, etc.

First, we will identify all the features of the system, which includes concert feature and aspect features. Table 1 below shows a list of features with their responsibilities. By relating these features to use case mappings, some features exist in multiple business flows. These features should be considered as candidates of aspectual features in this step. Fig. 1 shows a trivial part of use case mapping. As mentioned, we decompose use case into business flows and observe systems' internal processes and behaviors. Fig. 1 shows a business flow of "Capture a Crisis Report".

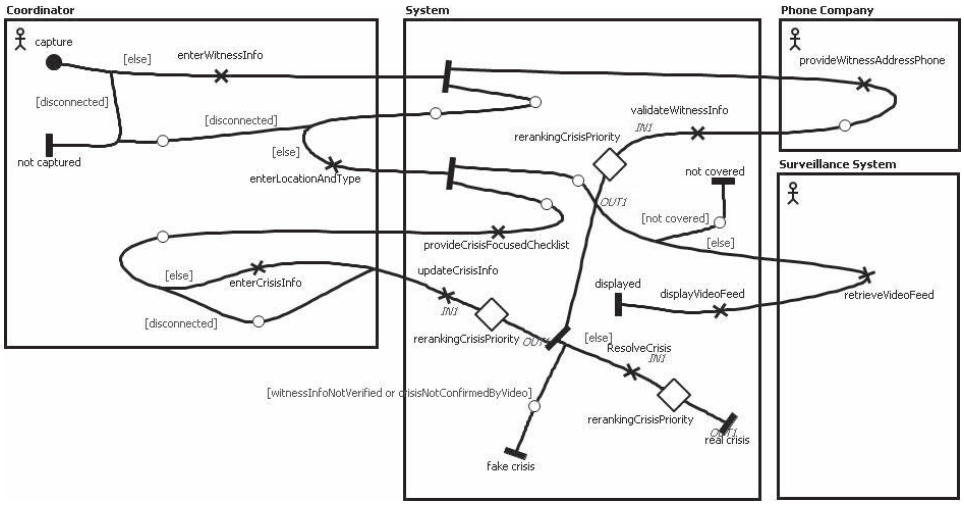

Figure 1: "rerankingCrisisPriority" in use case mapping.

From the figure, we can see that "rerankingCrisisPriority" (marked as $\diamondsuit$), which is a functional behavior of the system, exists in multiple parts of use case processes. This functional behavior has interactions with multiple system functions, so the corresponding feature could be looked as a potential candidate of aspectual feature.

Table 1: Features and responsibilities.

Features	Responsibility
Crisis receive	when system receives a new crisis report, this information will be sent to center management
Position detection	to confirm a crisis report, the location has to be identified by various approaches
Communication	which provides support for communication and information transmission among all resources
Task assignment	system assigns rescue tasks to mobile units according to current resources and crisis priorities
rerankingCrisisPriority	once a crisis been received or a rescue task been completed, system will automatically re-rank the rest crisis based on their critical levels

The next step is to identify aspectual concerns. To identify aspectual concerns in AO feature model, we need to find out the related quality attributes and how they are decomposed into concerns. In this case, we know that several quality attributes, such as "Performance", "Security" and "Mobility", are affected by multiple features (concrete and aspectual). These features are crosscut by quality attributes in terms of various aspectual concerns. Table 2 lists the crosscutting relationships between aspectual concerns and related quality attributes by "**X**".

Note that these aspectual concerns are not functional requirements, they represent system's non-functional properties. For example, "Constant cache management" is an identified aspectual concern that has impact on systems' "Performance" and "Security". This concern could contain various representations to meet particular levels of related quality attributes. For a high level of system security, "Constant cache management" could be achieved by encrypted data storage in cache that only can be accessed through appropriate authorization. For a system with low security requirement, this concern could be achieved by a direct-mapped cache. However it is able to enhance systems' performance in term of fast memory transmission.

Table 2: Aspectual concerns and related quality attributes.

Aspectual Concerns	Performance	Security	Mobility
Constant cache management	**X**	**X**	
Instant message exchange	**X**		**X**
Fast logging verification	**X**	**X**	

To identify aspectual features, only candidates that contribute to achievement of aspectual concerns should be considered as aspectual features. For example, for features in Table 1, "rerankingCrisisPriority" is considered as an aspectual feature as it contributes to aspectual concerns such as "Constant cache management" and "Instant message exchange". "Communication" does not contribute to any of identified aspectual concern, so it is considered as a concrete feature although it exists in multiple business flows and crosscut other features in different system processes.

To dig further, the crosscutting relationships between aspectual concerns and features also need to be addressed. Table 3 below represents these relationships. For example, for the feature of "Task assignment", it is affected by "Instant message exchange" for system to explore and examine current crisis situations and available resources to arrange rescue tasks. Similarly, related users need log into system to check

and receive rescue orders, so system has to provide fast and secure logging through "Fast logging verification".

Table 3: Relationships between concerns and features.

Features	Constant cache management	Instant message exchange	Fast logging verification
Crisis receive	X	X	
Position detection		X	X
Task assignment		X	X
rerankingCrisisPriority	X	X	

Down to architectural level, we develop a set of architecture scenarios to describe system architecture, which includes branching lines and alternative flows to address architectural variabilities. Fig. 2 shows a partial example of architecture. Here we have adapted the representation of AOGA (Kulesza et al., 2004) to represent aspectual components and aspectual interfaces. In the figure, identified aspectual feature "rerankingCrisisPriority" is mapped as a component called "reranking" on architecture and variable design options are achieved by aspect crosscutting in different ways. For example, "reranking" crosscuts components "Report" and "Task management" through interface "data transferring" to transmit data for systems' report generation and live task management. It also crosscuts "Communication" and "Execute mission" through "information collecting" to provide secure/instant communication and information update among available resources.

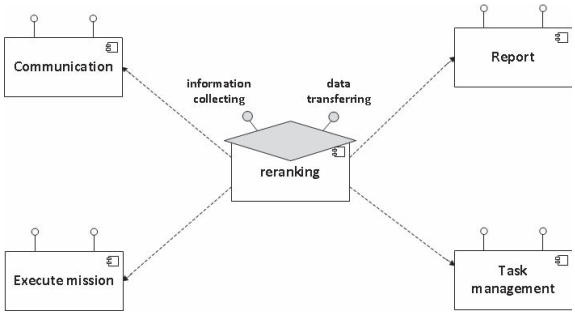

Figure 2: "reranking" crosscuts other components.

5 CONCLUSION AND FUTURE WORKS

In this paper, we introduce Software Product Line Engineering (SPLE) and discuss existing problems. To deal with these issues, we propose a new comprehensive framework by using aspect-oriented techniques and approaches to simplify complex features relationships and NFRs impacts. Our approach starts from feature modeling and provides mappings onto architecture development. The advantage of the framework is that it enhances the modularity of system as an AO approach. The transformation between requirement and architecture is more efficient. The software architecture development will be more accurate because NFRs and qualities are treated as part of architectural elements.

For the future work, we need further investigate how to enhance the systematic mappings from requirement to architecture. We have developed a City Evolution product line and we would like to apply the framework on this real-case to evaluate our approach.

REFERENCES

Bosch, J. (2000). *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley.

Brichau, J., Chitchyan, R., Rashid, A., and D'Hondt, T. (2008). Aspect-oriented software development: An introduction. *Wiley Encyclopedia of Computer Science and Engineering*.

Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J. (1999). *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, Boston.

Conejero, J. M. and Hernandez, J. (2008). Analysis of crosscutting features in software product lines. In *Proceedings of the 13th International Workshop on Early Aspects*, pages 3–10.

Eixelsberger, W., Ogris, M., Gall, H., and Bellay, B. (1998). Software architecture recovery of a program family. In *Proceedings of the 20th International Conference on Software Engineering*, pages 508–511.

Kienzle, J., Guelfi, N., and Mustafiz, S. (2010). Crisis management systems: A case study for aspect-oriented modeling. *Transactions on Aspect-Oriented Software Development*, pages 1–22.

Kulesza, U., Garcia, A., and Lucena, C. (2004). Generating aspect-oriented agent architectures. In *Proceedings of Early Aspects AspectOriented Requirements Engineering and Architecture Design*.

Lee, K., Kang, K., and Lee, J. (2002). Concepts and guidelines of feature modeling for product line software engineering. In *Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools*, pages 62–77.

Rommes, E. and America, P. (2006). A scenario-based method for software product line architecting. In *Software Product Lines*, pages 3–52. Springer Berlin Heidelberg.