An Approach for Identifying and Implementing Aspectual Features in Software Product Lines

Mohamed A. Zaatar
College of Computing & IT
Arab Academy for Science & Tech.
Cairo, Egypt
mzaatar@acm.org

Haitham S. Hamza
Dept. of IT
Cairo University
Giza, Egypt
hshamza@acm.org

Abd El Fatah Hegazy
College of Computing & IT
Arab Academy for Science & Tech.
Cairo, Egypt
abdheg@yahoo.com

Abstract

Software Product Lines (SPL) exploits reuse by identifying, modeling, and systemically reusing software features to develop different but related software systems. Successful reuse of a product line depends greatly on the modularity of the features that characterize the product line. Traditionally, features in SPL are grouped along the dimension of commonality and variability. However, this single-dimension grouping overlooks the crosscutting nature of some features in the system, which negatively impacts the reusability and modularity of the product line architecture. In this paper we address this particular problem by investigating the concept of Aspectual Feature (AF) as another grouping dimension that can be used in SPLs. To this end, the paper proposes the Aspectual Product Line Engineering (APPLE) approach for identifying, modeling, and implementing AFs to enhance the reuse of SPLs. A tool support for implementing the APPLE approach is also presented and demonstrated through a case study.

1 Introduction

Software product line (SPL) engineering has emerged as an effective and practical technology to exploit systematic reuse in developing software applications. An SPL can be defined as “a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a particular way” [1]. A core activity in engineering product lines is the Domain Engineering (DE) process used to engineer reusable assets necessary to develop a family of applications within a defined domain [2][3].

Domain Analysis (DA) [4] is a key activity in DE that focuses on identifying and modeling common and variable features for a set of software systems, and exploiting these features into reusable artifacts that can be used for engineering various related products and applications. This activity is known as Commonalities and Variabilities (C&V) analysis. Identifying, modeling, and encoding C&V features play a key role in facilitating the reuse of these features. Dependency, coupling, and cross-cuttings among features may limit their reuse, or at least make reuse more difficult and error prone. Conventional SPL development approaches group features along the dimension of commonality and variability without considering the interrelationships among these features.

In practice, features do not live in isolation. They typically cross-cut and interact with each other to implement the key functionalities of the system. Figure 1 shows some examples of features that may cross-cut each other within the commonality (A and B) or variability layers (X and Y), or even across the two layers (C and D).

Dealing with cross-cutting features in SPL is not new. In the literature, two research directions were investigated in this context. The first direction focuses on identifying crosscutting features within the variability features (e.g., Features X and Y in Figure 1. [6] [7] [8] [9]. However, this work ignores other types of crosscutting features within the commonality or across the commonality and variability layers. The second research direction focuses on developing...
and end-to-end development process to handle crosscutting features in the SPL life-cycle. Example of this direction is the approach reported in [12]. However, this approach is complex as it relies on data-mining and natural language processing to identify crosscutting features.

We argue that identifying cross-cut features is circuital in developing truly reusable software product lines, and there should be a practical approach for identifying various types of such features. Accordingly, in this paper, we investigate the concept of Aspectual Feature (AF) as another grouping dimension that can be used in conjunction with the traditional commonality and variability dimension used in most SPL techniques. Recent work reported in [6], [7], [8], and [9] focuses on To this end, we propose a new approach, namely, the Aspectual Product Line Engineering (APPLE) approach based on the theory of Formal Concept Analysis (FCA) for identifying, modeling, and implementing AFs to enhance the reuse of SPLs. In addition, we present a CASE tool for implementing the APPLE approach and demonstrate its use via a case study.

The rest of the paper is organized as follows. Section 2 presents the proposed APPLE approach and describes its main activities. In Section 3, a case study for applying the APPLE approach is presented. Related work is discussed in Section 4. Conclusions are presented in Section 5.

2 The Proposed Approach

In this section we present the proposed Aspectual Product Line Engineering (APPLE) approach, and provide an overview about its main activities and implementation. A brief overview about the Formal Concept Analysis (FCA) theory is first presented.

2.1 Formal Concept Analysis (FCA)

Formal concept analysis (FCA) provides means to identify meaningful groupings of objects that share common attributes as well as provides a theoretical model to analyze hierarchies of these groupings [18]. A formal context \( C = (A, O, R) \) in FCA consists of a set of objects \( O \), a set of attributes \( A \), and a binary relation \( R \subseteq O \times A \) between objects and attributes, indicating which attributes are possessed by each object. The formal context can be used to generate a set of concepts where a concept \( C \) is defined as a pair of sets \( (X, Y) \) such that:

\[
X = \{o \in O | \forall a \in Y : (o, a) \in R\} \tag{1}
\]

\[
Y = \{a \in A | \forall o \in X : (o, a) \in R\} \tag{2}
\]

Where \( X \) is said to be the extent of the concept \( C \), and \( Y \) is said to be its intent.

The set of all concepts of a formal context and the partial ordering can be represented graphically using a concept lattice. The concept lattice is the basis for further data analysis.

Table 1 shows an example of a simple context. The lattice of this context is illustrated in Figure 2.

2.2 The APPLE Approach

APPLE aims at providing a practical approach for detecting and propagating aspects from analysis to implementation without imposing heavy-weight analysis techniques those are not in the mainstream development process. As we discuss below, in the proposed approach, typical SPL activities such as Use Case analysis and Feature-Oriented Domain Analysis (FODA) are employed. The use of FCA helps to achieve this goal as it is a semi-automated technique that does not require much knowledge from typical analysts and developers.

In the following we present the various steps of the APPLE approach (Figure 3). A summary of the various activities, their input, and outputs are summarized in Table 2.

Step 1: Domain Analysis: As with typical SPL development approaches, the first step is to perform functional and domain analysis in order to identify the required functionalities and the common and variable features in the systems under consideration. In this step, the following two parallel activities are performed:

1.1 UML Domain Analysis: This activity aims at analyzing the functional and non-functional requirements of the systems within the specified domain. The output of this activity is a set of Use Cases (UCs) describes the requirements of the systems.
1.2 Feature Oriented Domain Analysis: In this activity, common and variable features among the systems are identified using the FODA method [5]. Identified features are modeled using conventional feature models (FM) [5].

Step 2: Aspectual Features Detection: This step focuses on identifying aspectual features (AFs) based on the UCs and FM developed in Step 1 above. To do so, the following two activities are performed:

2.1 Features and UCs Dependencies Detection: In this activity, the relationship between the identified UCs and features are carefully analyzed in order to identify their dependencies. We follow an approach similar to that reported in [8][9]. The result of this activity is a dependency matrix where the rows and columns represent the features and the UCs, respectively.

2.2 Aspectual Feature Extraction: The matrix obtained from the previous activity is processed using FCA in order to identify actual Aspectual Features in the systems. A lattice is constructed and its structure is analyzed (via a tool) in order to identify crosscutting and non-crosscutting features.

Step 3: Aspectual Feature Modeling: In this step, the conventional feature model constructed in Step 1 is now revised and updated based on the results of Step 2.2 above. The result is a new feature model with a set of features classified along two dimensions. The first dimension differentiates between common and variable features (from the original FM), whereas the second dimension differentiates between crosscutting and non-crosscutting features (based on activity 2.2). It is worth noting that, at this point crosscutting and non-crosscutting features include both common and variable features. This particular point set the APPLE approach from most existing research that focuses only on identifying crosscutting features within variabilities only. The result of this step is an updated FM named Aspectual Feature Model (AFM).

Step 4: Code Generation: In this step, the actual implementation of the SPL is performed. In our approach, we adopt the PLUM (Product Line Unified Modeler) as a tool for generating the code for the product line under development [17]. In PLUM, common features are encoded in what is called Flexible Components (FCs), whereas variable features are presented in the so-called Decision Model (DM). DM is used to set the values of the variable features according to the particular product in the SPL family. Accordingly, we handle the common and variable aspectual features differently. That is, common AFs are handled in the FCs, whereas variable AFs are handled in the DM. The following are the two activities used in APPLE to implement AFs:

4.1 Aspectual Features Filtration: In this activity, we isolate common and variable aspectual features in order to prepare these features for appropriate implementation.

4.2 PLUM Implementation: In this activity the actual code is developed and generated as appropriate. In particular, common AFs are implemented in the FCs using appropriate Aspect-Oriented Pro-
A CASE tool is developed in order to semi-automated most of the activities in the proposed APPLE approach. The tool supports the four steps shown in Figure 3. The GUI of the developed tool is illustrated in Figure 4. In the lattice generation it extracts and identifies the aspectual features by exploring the scattering and tangling dependencies. Each feature have to be scattered and at least one of the corresponding use cases are tangled to be an aspectual feature. A customization done on the FCA generation to show the aspectual features by analysis of the dependencies and draw hashing on the aspectual features. The tool includes add-in for UC modeling, and Feature Modeling.

The conventional feature model is extended to include the aspectual feature types. Figure 5 shows a sample aspectual feature model with two aspectual features, namely Save and Persistence.

Aspectual features are marked with an X in the left-hand side as shown in the figure. Each aspectual feature crosscuts one or more features. The modified aspectual model illustrates this crosscutting nature by dotted line. For example, as shown in Figure 5, the aspectual feature Save crosscuts three other features: Pause, Exit, and Persistence.

3 Case Study: The Arcade Game Maker

To demonstrate the APPLE approach, we applied it to the Arcade Game Maker Product Line [8][16]. Arcade Game Maker is a SPL that aims at generating three related, but different games. Each game is a Single-player Game and has some Rules. A Scoring point is obtained by hitting some hurdles. The installer component of the software can install and uninstall the game. The User should be able to interact with the game. Some interactions with the game is to play, save Game and exit. Saving Score is one of the features available in the game. Examples for some functional requirements in this system include: Play, Pause, install, and Uninstall. Examples for some non-functional requirements include: Performance and Persistence. Most of the func-
Table 2. Summary of Main Activities in the APPLE Approach.

<table>
<thead>
<tr>
<th>Step</th>
<th>Activities</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- Domain Analysis</td>
<td>UML Domain Analysis</td>
<td>Requirements</td>
<td>Use Cases (UCs)</td>
</tr>
<tr>
<td></td>
<td>Feature Analysis (FODA)</td>
<td>Domain Requirements</td>
<td>Feature Model (FM)</td>
</tr>
<tr>
<td>2- Aspectual Features Detection</td>
<td>Features and UCs Dependencies Detection</td>
<td>UCs and FM</td>
<td>FCA Lattice</td>
</tr>
<tr>
<td></td>
<td>Aspectual Feature Extraction</td>
<td>FCA Lattice</td>
<td>Aspectual Features (AFs)</td>
</tr>
<tr>
<td>3- Aspectual Feature Modeling</td>
<td>AF Modeling</td>
<td>AFs</td>
<td>Aspectual Feature Model</td>
</tr>
<tr>
<td></td>
<td>PLUM Implementation</td>
<td>Filtered Features</td>
<td>PLUM Code</td>
</tr>
<tr>
<td>4- Code Generation</td>
<td>Aspectual Features Filtration</td>
<td>AFM</td>
<td>Aspectual Variability and Commonality Features</td>
</tr>
</tbody>
</table>

Table 3 gives a comparison between the proposed APPLE approach and the matrix-based and NAPLES approaches.

5 Conclusions

In this paper, the Aspectual Product Line Engineering (APPLE) approach is proposed to exploit the concept of aspectual features (AFs) for developing modular software product line systems. The key objective of APPLE is to identify, model, and implement the crosscutting features by analyzing the relationships between the requirements and features of the system early in the analysis phase. Based on this analysis, AFs are identified and handled efficiently to facilitate the reuse of the SPL. The APPLE approach is...
Table 3. Comparison Between Various SPL Approaches.

<table>
<thead>
<tr>
<th>Property</th>
<th>Matrix-based Technique [8]</th>
<th>NAPLES [12]</th>
<th>APPLE (This Paper)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Development</td>
<td>Analysis</td>
<td>Analysis Code Generation</td>
<td>Analysis Design and Architecture Code Generation</td>
</tr>
<tr>
<td>Phase(s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aspectual Feature Modeling</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>C &amp; V</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Crosscutting Feature</td>
<td>Matrices</td>
<td>Data Mining</td>
<td>FCA</td>
</tr>
<tr>
<td>Detection Technique</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

presented and its activities are described. A CASE tool to semi-automate the various activities of the APPLE approach is also presented and its use is demonstrated through the Arcade Game Maker software case study.

References


