A Hybrid Approach to Feature-Oriented Programming in XVCL

Hongyu Zhang[1] and Stan Jarzabek[2]

[1] School of Software, Tsinghua University, Beijing 100084, China
hongyu@tsinghua.edu.cn
[2] School of Computing, National University of Singapore, Singapore 117543
stan@comp.nus.edu.sg

Abstract. Feature-Oriented Programming (FOP) is a programming paradigm for developing programs by composing features. It is especially useful for software product line development, as each product line member implements some combinations of features. FOP attempts to modularize features and to enable their flexible composition into programs. Recent studies have shown that it is not practical to modularize and then compose features that have fine-grained impact on base programs. In this paper, we present a hybrid approach to feature modularization/composition problem. We modularize only separable features that can be well contained in dedicated files. We handle inseparable features by annotating base programs using preprocessing-like directives. We show how the hybrid approach can be achieved in XVCL, a generative technique to manage variabilities in software product lines.

1 Introduction

Feature-Oriented Programming (FOP) is a programming paradigm for developing programs by composing features [2, 11]. FOP extends the principle of separation of concerns to features. It attempts to modularize feature implementation and provides a mechanism to compose features into a base program in required, legal combinations. The ability to compose features in flexible way is particularly useful for software product line development, where we need to handle a family of similar products characterized by common features, but with each product implementing some unique features.

Researchers have experimented with techniques that can realize the concept of FOP. AHEAD [2] is among the mature techniques. Some researchers also suggested Aspect-Oriented Programming (AOP) as a possible technique to realize FOP [1, 10]. Both AHEAD and AOP can handle the *separable features*, which can be well modularized in dedicated files and can be composed into base programs at a relatively small number of variation points. Recently Kastner and Apel found that AHEAD and AOP have limitations in handling features that have fine-granular impact on base programs at many variation points [7, 8]. These limitations could lead to the excessive use of inheritance and hook-methods, causing difficulties in program understanding and maintenance. Such *inseparable features* create a major challenge for FOP.

J. Bosch and J. Lee (Eds.): SPLC 2010, LNCS 6287, pp. 440–445, 2010.

In this paper, we present a hybrid approach to feature modularization/composition problem. We modularize only separable features and handle inseparable features by annotating programs using preprocessing-like directives. We show a realization of the proposed hybrid approach in XVCL (XML-based Variant Configuration Langue) [4, 5, 14], which is a generative technique to manage variabilities in software product lines. Although XVCL has been applied in many product line and software maintenance studies, its usage in FOP was not well explored before.

Unlike other approaches such as AHEAD or AOP, our approach does not force all the features to be separated and modularized. Our approach is both annotative and compositional. For features that can be well separated, we modularize their implementations by placing their code in separate modules. When clean feature modularization becomes problematic, we directly annotate the base programs with necessary feature code. By doing so, we avoid the limitations of current FOP techniques (such as the excessive use of inheritance and hook-methods), and improve the maintainability of the feature programs. We allow developers to choose the method for handling different features, thus achieving higher degree of flexibility in programming practices.

2 A Hybrid Approach to FOP

Feature-Oriented Programming (FOP) is based on the principle of advanced separation of concerns [13]. Features are often treated as a form of concerns that can characterize software products. In FOP, each module (or each layer of modules) localizes the code implementing a feature. If we could find a way to clearly separate all the features using FOP, no doubt our software engineering problems would be much less than they are today. However, some of the features are so tightly coupled with other features or with the base programs that their physical separation becomes difficult. We thus classify the features into two types:

Separable features: Features that can be easily separated from the base programs. The impact of these features can be well modularized in dedicated files and can be composed into base programs at a relatively small number of variation points.

Inseparable features: Features that are difficult to be separated from the base programs. Examples of such features include:

- Features that require fine-grained changes. The examples of fine-grained changes include changes in method signature, changes in the middle of a method, and extensions of expressions. To handle such fine-grained changes, the existing modularization and composition mechanisms often lead to complicated and unmaintainable implementations.
- Features that are inherent properties of base programs. These features are integral parts of the descriptions of concepts. They cannot be separated from the base programs without hampering the program integrity and understandability. For example, for a Buffer class, its data type (such as Int, Char, Byte, Double, etc) is an integral property of the class. Therefore it is better to implement this data type feature as generics, instead of separating it into independent modules.

- Features that have intensive interactions with other features. Some features heavily crosscut other features. One example of such feature is exception handling. Separating the exception handling code from the base program is shown to be difficult. Performance and security features are other examples. Performance/Security has pervasive impact on many design decisions. While we can conceive and express performance/security concerns conceptually (e.g., by documenting design decisions that have to do with performance/security), "physical" modularization of these features may not be feasible.

To handle separable features, we can modularize their impact into dedicated modules, and then compose these modules with the base programs. This process is the same as what current FOP approaches do. The modularization of feature-specific code helps programmers understand and maintain features. To handle inseparable features, we can directly annotate the base programs to incorporate the impact of these features. The basic concept is close to the concept of C/C++ preprocessing directives. We can thus understand the base programs and the inseparable features as a whole. In this way, we avoid the limitations of current FOP approaches (such as the excessive use of inheritance and hook-methods).

3 A Realization of the Hybrid FOP in XVCL

3.1 An Overview of XVCL

XVCL (XML-based Variant Configuration Language) is a variation mechanism for managing variability in software product lines based on generative techniques [3]. XVCL has been applied to support product lines in lab studies and industrial projects [4, 5, 14].

In XVCL, generic and adaptable programs are called x-frames. X-frame body is written in the base language, which could be a programming language such as Java or PHP. An x-frame is also marked-up by XVCL commands (in the form of XML tags), which enables the composition and adaptation of the x-frame. Typical XVCL commands include the <adapt> command that composes two x-frames, the <select> and <ifdef> commands that allow one to select pre-defined options based on certain conditions, the <break> command that marks breakpoints (slots) where additional code can be inserted, and the <insert> command that inserts additional code into a specified slot. To derive a customized program from x-frames, one needs to design a Specification x-frame (SPC), which records specific product configurations according to the user requirements. Given SPC, the tool *XVCL processor* performs composition and adaptation by traversing x-frames along <adapt> chains, executing XVCL commands embedded in visited x-frames, and constructing concrete programs.

XVCL is a software reuse technique based on the idea of "composition with adaptation". Like macros, x-frames contain code fragments along with adaptation commands. Unlike macros, XVCL has unique capabilities that enhance reuse, such as scoping and overriding rules for meta-variables, insertions at breakpoints and while loops that facilitate generating custom component instances from a generic x-frame. Unlike OO inheritance, C++ template and Aspect-Oriented Programming (AOP)

techniques, XVCL can handle variants at any granularity level. One can explicitly mark the variation points in a program and specify required adaptations.

More details about XVCL can be found at the XVCL homepage[1]. Although XVCL has been applied to many product line and software maintenance studies, its usage in FOP was not well explored before.

3.2 Implementing FOP Using XVCL

We use XVCL to realize the proposed hybrid approach to FOP. To implement inseparable features, we use XVCL commands (such as <select> and <ifdef>) to directly mark the variation points in base programs at which customization will occur. These commands will select necessary code to be included into the generated program based on the values of meta-variables that are set in the specification x-frame (SPC).

We place the implementation of separable features in dedicated x-frames. To support feature modularization, we use the XVCL <select> commands to encapsulate all necessary code fragments for implementing a feature. We also use the XVCL <break> commands to specify the slots in the base programs at which possible extensions could be made. During program composition, we select suitable code fragments in the feature x-frame and insert them into the slots defined in the base programs.

In our XVCL-based hybrid approach to FOP, we develop two types of x-frames:

- x-frames that implement the base programs. The base programs can be developed in the conventional object-oriented manner and can be annotated with XVCL commands to accommodate inseparable features.
- x-frames that implement separable features.

These two types of x-frames can be composed together to form a complete, compilable program. A valid combination of features for a specific system is described in an SPC. The XVCL processor performs the composition of base classes and features according to the instructions given in the SPC, and generates a specific system that implements the required features.

3.3 Experiments

We have evaluated our approach through the development of the Graph Product Line (GPL). GPL has been proposed by Lopez-Herrejon and Batory [9] as a standard problem for evaluating software product line technologies. It is a family of classic graph applications. In GPL, we identify *Directed/Undirected* and *Weighted/Unweighted* features as inseparable features. These features are inherent properties of the base programs. Their code is tightly coupled with the base program at the fine-granular level and cannot be easily isolated. Therefore we use the <ifdef> commands to annotate the base program with the optional feature code. We modularize other features (such as the *Cycle* feature) in dedicate x-frames. By separating these features from the base

[1] `http://xvcl.comp.nus.edu.sg`

programs, we enhance the readability of the base programs. Given a set of features, a specific GPL product can be generated from the x-frames.

To further evaluate our method, we re-engineered the Berkeley DB system into hybrid FOP representations. The Berkeley DB case study was also used by Kastner et al. [7], who refactored the system into features using AspectJ. In our study, we reengineered 23 Berkeley DB features into x-frames. Features such as *EvictorDaemon* are implemented as inseparable features, features such as *FSync* are implemented as separable features.

3.4 Tool Support

To facilitate XVCL-based development, especially for a large and complex product line, we have also developed a set of tools including a development workbench (an IDE for editing x-frames), a feature configuration checker (for checking the validity of a selected feature combination), and a feature query tool (for analyzing the impact of a feature on x-frames). These tools support the development of x-frames, their reuse and evolution, mitigating potential problems of understandability and scalability of complex x-frames. More details about these tools can be found at [6, 12].

4 Conclusions

Feature-Oriented Programming (FOP) is a programming paradigm for developing programs by composing features. We classify features into two types: 1) inseparable features that affect base programs at fine-granular level and cannot be easily separated from base programs, and 2) separable features that can be easily separated from base programs and contained in dedicated modules. Recent studies show that current FOP techniques based on advanced separation of concerns principle, such as AHEAD and AOP, have limitations in implementing inseparable features.

In this paper, we have presented a hybrid approach to feature modularization/ composition problem. We modularize only separable features and handle inseparable features by annotating programs using preprocessing-like directives. We have presented a realization of the above approach in XVCL, a generative technique to manage features in software product lines.

In future, we plan to further investigate new tools that can facilitate XVCL-based FOP development. One trade-off in our approach is that we cannot guarantee correctness of programs generated from meta-programs (x-frames). We will address this issue in future work. We also plan to carry out a larger-scale evaluation of the proposed hybrid approach.

Acknowledgments. We would like to thank Roberto E. Lopez-Herrejon for providing us the source code of their implementation of GPL in AHEAD. This research is supported by the Chinese NSF grant 60703060, the NUS research grant RP-252-000-336-112, and the MOE Key Laboratory of High Confidence Software Technologies at Peking University.

References

1. Apel, S., Leich, T., Saake, G.: Aspectual Feature Modules. IEEE Trans. Software Eng. 34(2), 162–180 (2008)
2. Batory, D., Sarvela, J., Rauschmayer, A.: Scaling Step-Wise Refinement. IEEE Trans. Software Eng. 30(6), 355–371 (2004)
3. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley, MA (2000)
4. Jarzabek, S.: Effective Software Maintenance and Evolution: Reuse-based Approach. CRC Press, Taylor & Francis (2007)
5. Jarzabek, S., Zhang, H.: XML-based method and tool for handling variant requirements in domain models. In: Proc. Int'l. Symp. on Requirements Engineering (RE 2001), Toronto, Canada (2001)
6. Jarzabek, S., Zhang, H., Lee, Y., Xue, Y., Shaikh, N.: Increasing Usability of Preprocessing for Feature Management in Product Lines with Queries. In: Proc. ICSE 2009, Vancouver, Canada, May 2009, pp. 215–218 (2009)
7. Kästner, C., Apel, S., Batory, D.: A Case Study Implementing Features Using AspectJ. In: Proc. Int. Software Product Line Conference (SPLC 2007), Kyoto, Japan, September 2007, pp. 223–232 (2007)
8. Kästner, C., Apel, S., Kuhlemann, M.: Granularity in Software Product Lines. In: Proc. ICSE 2008, Leipzig, Germany, May 2008, pp. 311–320 (2008)
9. Lopez-Herrejon, R.E., Batory, D.: A standard problem for evaluating product-line methodologies. In: Bosch, J. (ed.) GCSE 2001. LNCS, vol. 2186, pp. 10–24. Springer, Heidelberg (2001)
10. Mezini, M., Ostermann, K.: Variability Management with Feature-Oriented Programming and Aspects. In: Proc. SIGSOFT FSE 2004, Newport Beach, CA, pp. 127–136 (2004)
11. Prehofer, C.: Feature-oriented programming: A new way of object composition. Concurrency and Computation: Practice and Experience 13(6), 465–501 (2001)
12. Sun, J., Zhang, H., Li, Y., Wang, H.: Formal Semantics and Verification for Feature Modeling. In: Proc. 10th Int. Conf. on Engineering of Complex Computer Systems (ICECCS 2005), Shanghai, June 2005, pp. 303–312 (2005)
13. Tarr, P., Ossher, H., Harrison, W., Sutton Jr., S.M.: N Degrees of Separation: Multi-Dimensional Separation of Concerns. In: Proc. ICSE 1999, Los Angeles, CA, USA (May 1999)
14. Zhang, H., Jarzabek, S.: XVCL: A Mechanism for Handling Variants in Software Product Lines. Science of Computer Programming 53(3), 381–407 (2004)