Exploring the Synergies between Join Point Interfaces and Feature-Oriented Programming

Cristian Vidal Silva[1], David Benavides[2], José Angel Galindo[3], and Paul Leger[4]

[1] Departamento de Computación e Informática
Facultad de Ingeniería, Universidad de Playa Ancha
Av. Leopoldo Bertossi 270, Playa Ancha, Valparaíso, Chile
cristian.vidal@upla.cl

[2] University of Seville
Av. de la Reina Mercedes S/N, 41012 Seville, Spain
benavides@us.es

[3] INRIA
Renes, France
jagalindo@inria.fr

[4] Escuela de Ciencias Empresariales
Universidad Católica del Norte
Coquimbo, Chile
pleger@ucn.cl

Abstract. Feature-oriented programming FOP, and aspect-oriented programming AOP have been used to develop modular software product lines SPL. Both approaches focus on modularizing classes behavior and crosscutting concerns CC. Therefore, the symbiosis of FOP and AOP would permit reaching pros and cons of both software development approaches. Concretely, FOP permits a modular refinement of classes collaboration for software product lines SPL -an adequate structural representation of heterogeneous CC, but FOP does not well represent homogeneous CC. On the other hand, traditional AOP structurally well modularizes homogeneous CC, but aspects are not adequate to represent collaboration of classes for software evolution. In addition, AOP solutions present implicit dependencies and strong coupling between classes and aspects. Since Join Point Interface JPI solves mentioned AOP issues, this paper present JPI Feature Modules to represent and modularize the structure of FOP and JPI SPL instances, i.e., classes and join point interfaces for a transparent implementation in a FOP and JPI context. This paper, highlights benefits of a FOP and JPI symbiosis for the modular software conception using a case study to exemplify its use.

Keywords: FOP, classic AOP, JPI, modular software, JPI-FM

1 Introduction

The separation of concerns principle [6] mentions that generating modular programs is one of the main goals of programming methodologies. In this context, for software modularity, Object-Oriented Programming OOP represents an evolution concerning structured programming -OOP encapsulates attributes and functions as members of classes, and defines information hiding rules to access class members. Nevertheless, OOP neither directly encapsulates features and SPL architecture [1] [9], nor CC as independent modules [7].

Just, to improve OOP issues, the software development paradigms Feature-Oriented Programming FOP and Aspect-Oriented Programming AOP appeared:

- FOP well modularizes collaboration of classes, named heterogeneous CC, as features, and permits step-wise development of SPL [1]. Nevertheless, [7] [2] remark, FOP lacks adequate crosscutting modularity for software evolution since software has to change and adapt to fit non-predictable modifications. Particularly, for code repetition, “FOP does not modularize elegantly homogeneous CC ”[1].
- AOP, for a modular behavior of classes, defines oblivious advised classes and modularizes CC as aspects [7]; so AOP solutions are able to respect the “single responsibility”OOP design principle [10]. As [1] indicate, AOP well modularizes homogeneous CC as aspects, but it is not adequate to modularize classes collaborations. Moreover, classic AOP introduce implicit dependencies between aspects and classes, and aspects need to know about advised classes before advising [5]. Last thing is a great issue for independent development.

Since, FOP and AOP well modularize different CC, a symbiosis of FOP and AOP should profit of each other to obtain modular software [8] [1]. Thus, looking for a FOP and AOP symbiosis, for a collaboration-based design, Aspectual Feature Modules AFM [3] [1] are useful to represent modular classes and aspects, their association and evolution. Nonetheless, AFM preserves classic AOP issues. Thus, given the FOP benefits to produce modular software, and the JPI capability to respect OO principle and modularize dynamic homogeneous CC [5], the main goals of this work are to present JPI Feature Modules JPI-FM to structurally model JPI and FOP solutions as well as analyze its benefits for the massive customized software production.

Next, we defines a JPI-FM approach and highlights its benefits using an application example. After it, conclusions and future research work appear.

2 JPI Feature Modules: JPI-FM

Given the FOP and JPI benefits, we propose JPI-FM looking for the symbiosis of both paradigms: 1st, FOP, for software evolution, for collaboration of classes and new system elements, heterogeneous and static CC; 2nd, JPI, to avoid code replication and represent dynamic homogeneous CC, to respect OOP design principles. JPI and aspects can be refined.

Figure 1 illustrates a JPI-FM structure of 3 layers.

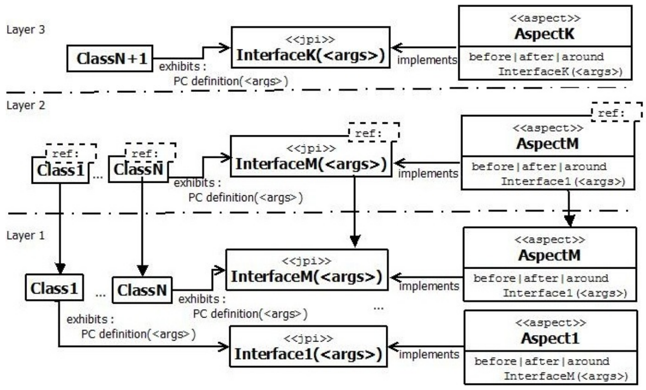

Fig. 1. General application of JPI-FM

- Base layer presents a set of N classes, M JPIs, and M aspects where classes exhibit JPIs which are implemented by aspects. JPI-FM stereotype classes for JPI components. In addition, classes that exhibit JPI instances define pointcut (PC) rules to define join point occurrences, and aspects implement those JPIs.
- Layer 2 presents the refinement of some previous layer elements. Layer 2 uses template on components to indicate refinement operation. A JPI refinement possibly requires refinement of associated advised class and aspect.
- Layer 3 preserves elements of Layer 2, and add a new class, a new JPI, and a new aspect; all associated.

Next, Figure 2 illustrates an application of JPI-FM on the graph example of [1] for 3 layers that presen an aspect and a JPI without refinement exhibited by class Edge of layers Basic Graph and Weighted.

3 Conclusions

JPI-FM presents modularization advantages respecting AFM since JPI exhibits notable modularization improvements over AOP, and uses FOP main properties to produce massive customized software applications. Thus, JPI-FM models a high-level massive modular software in a FOP and JPI symbiosis context. Exactly, to produce a complete FOP and JPI symbiosis is our MAIN goal.

As a future work, JPI-FM has can be extended to support a closure join points [4]. In addition, we want to define a case study to apply our modeling and future programming proposal, to evaluate its modularization effectiveness.

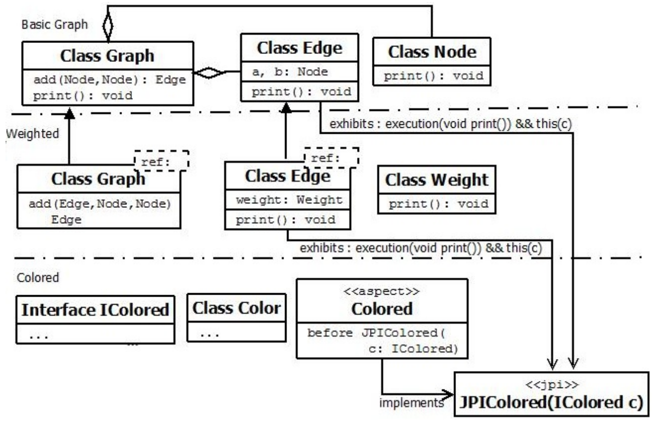

Fig. 2. JPI-FM of the graph example

References

1. Sven Apel, Don Batory, Christian Kstner, and Gunter Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, Incorporated, 2013.
2. Sven Apel, Thomas Leich, Marko Rosenmller, and Gunter Saake. Combining feature-oriented and aspect-oriented programming to support software evolution. In Walter Cazzola, Shigeru Chiba, Gunter Saake, and Tom Tourw, editors, *RAM-SE*, pages 3–16. Fakultt fr Informatik, Universitt Magdeburg, 2005.
3. Sven Apel, Thomas Leich, and Gunter Saake. Aspectual Feature Modules. *IEEE Transactions on Software Engineering*, 34(2):162–180, 2008.
4. Eric Bodden. Closure joinpoints: Block joinpoints without surprises. In *Proceedings of the Tenth International Conference on Aspect-oriented Software Development*, AOSD '11, pages 117–128, New York, NY, USA, 2011. ACM.
5. Eric Bodden, Éric Tanter, and Milton Inostroza. Join point interfaces for safe and flexible decoupling of aspects. *ACM Transactions on Software Engineering and Methodology*, 23(1):7:1–7:41, February 2014.
6. Edsger W. Dijkstra. The structure of the multiprogramming system. In *Communications of the ACM*, page 11(5):341346. Springer-Verlag, May 1968.
7. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean marc Loingtier, and John Irwin. Aspect-oriented programming. In *ECOOP*. SpringerVerlag, 1997.
8. Mira Mezini and Klaus Ostermann. Variability management with feature-oriented programming and aspects. *SIGSOFT Softw. Eng. Notes*, 29(6):127–136, October 2004.
9. Christian P—rehofer. Feature-oriented programming: A fresh look at objects. pages 419–443. Springer, 1997.
10. Dean Wampler. Aspect-oriented design principles: Lessons from object-oriented design. In *Proceedings of the Sixth International Conference on Aspect-Oriented Software Development (AOSD'07)*, pages 615–636, Vancouver, British Columbia, March 2007.