JPI Feature Models

Exploring a JPI and FOP Symbiosis for Software Modeling

Cristian Vidal Silva
Departamento de Computación e Informática
Facultad de Ingeniería, Universidad de Playa Ancha
Valparaíso, Chile
cristian.vidal@upla.cl

David Benavides
Universidad de Sevilla
Av. de la Reina Mercedes S/N, 41012
Sevilla, España
benavides@us.es

José Ángel Galindo
INRIA
Rennes, France
jagalindo@inria.fr

Paul Leger
Escuela de Ciencias Empresariales
Universidad Católica del Norte
Coquimbo, Chile
pleger@ucn.cl

Rodolfo Villarroel
Escuela de Ingeniería Informática
Facultad de Ingeniería,
Pontifica Universidad Católica de Valparaíso,
Valparaíso, Chile
rodolfo.villarroel@ucv.cl

Sebastián Valenzuela
Departamento de Computación e Informática
Facultad de Ingeniería, Universidad de Playa Ancha
Valparaíso, Chile
sebastian.valenzuela.f@alumnos.upla.cl

***Abstract*—Looking for a complete modular software development paradigm, this article presents Join Point Interface JPI Feature Models, in the context of a JPI and Feature-Oriented Programming FOP symbiosis paradigm. Therefore, this article describes pros and cons of JPI and FOP approaches for the modular software and software product line production, respective; and highlights the benefits of this mixing proposal; in particular, the JPI Feature Model benefits for a high-level software product line modeling. As an application example, this article applies JPI Features Models on a classic FOP example already modeled using a previous aspect-oriented feature model proposal. Main goals of this application are to visualize traditional feature models preserved components such alternative and optional feature sets and optional and mandatory features as well as special features associations (cross-tree constraints), and differences and advantages with respect to previous research works about extending feature model to support aspect-oriented modeling principles.**

Keywords—Feature Model, Features, Aspects, JPI, FOP

I. Introduction

Feature models represent features and their relationships of a family of software products or Software Product Line SPL [3] [4] [13] [14]. In addition, Feature Models FMs illustrate commonality and variability of software products of a SPL. According to [4] [13], since a SPL is a set of features along with their associations, a product represents a set of chosen features, and a feature corresponds to any incremental functionality.

Feature models capture the problem space, i.e., a particular selection of features to define a product of the family. The process of selecting desired features is called configuration and the set of selected features a product configuration [4] [10] [13]. A feature model is a tree of hierarchical features, and all configurations always include the root feature that represent the concept of the SPL. Figure 1 [3] [14] illustrates a known example of the feature model for a graph.

Feature models [4] [13] [14] support the following parent-child feature relationships:

- M*andatory*: a black circle in the head of the child feature connection, i.e., child feature is part of the configuration when its father is part of, and vice-versa.

- *Optional*: a white circle in the head of the child feature connection, i.e., child feature is not necessarily part of the configuration when its father is part of.

- *OR* set of features: a black arc below the father feature over the set of child features of which at least one of them must be in the configuration when the father is part of.

- *XOR* set of features: a white arc below the father feature over the set of child features of which only one of them must be in the configuration when its father is part of.

In addition, feature models support named Cross-Tree Constraints CTC *excludes* and *requires* between features; a double directed edge between associated features and a

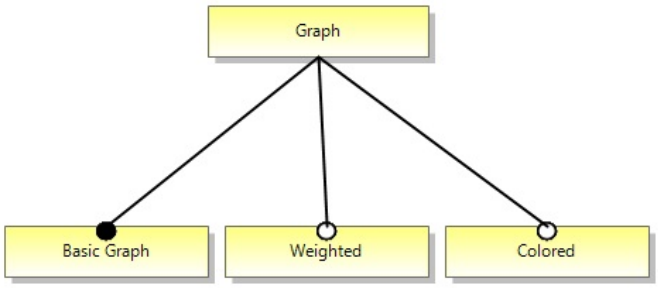


Fig. 1. Feature model of Graph.

directed edge from the source feature to the target feature, respectively.

FOP, comparing to other modularization approaches, and feature models as well, present a simpler and more direct representation of heterogeneous crosscutting concerns modularization [13] [15], i.e., by refining, FOP works on changes of different functionality pieces for which to define join points is not a simple task. Nevertheless, as [8] [10] [11] [13] [14] [15] argue, feature models do not support a complete modularization of crosscutting concerns; FOP does not efficiently modularize named homogeneous crosscutting concerns, i.e., for applying the same changes on different join points, FOP presents redundancies and repetition programming efforts. Thus, already mentioned sources describe solutions to mix traditional Aspect-Oriented Programming AOP approaches and feature oriented programming to support crosscut features modularization. Since AspectJ [1] like AOP approaches present implicit dependencies between advised classes and aspects, proposals to mix AOP like AspectJ approaches and FOP preserve mentioned issues.

For the homogeneous crosscutting concern modularization, Join Point Interfaces JPI [3] [5] [6] present a modularization approach to delete traditional issues of classic like AspectJ [1] AOP approaches, implicit dependencies between advised classes and aspects. JPI permit defining join point interfaces between advisable classes and aspects; hence, classes can exhibit a JPI instance and aspects implement these interfaces. Thus, JPI approach permits no more implicit dependencies between aspects and advised classes, no more oblivious classes and aspects dependencies of advised classes.

This article looks for a simple massive modular software production by taking advantages of the FOP to modularize heterogeneous crosscutting concerns [13] [15], and exploiting JPI to modularize homogeneous crosscutting concerns. Thus, this paper proposes and applies JPI feature model to draw features and crosscut-features along with their associations in a JPI + FOP context that represents a 1st step for a complete JPI + FOP software development process [3].

Clafer [9] is a work that presents, likes a meta-model, a mix of a class diagram and feature model for a system. Thus, Clafer represents crossed levels in the software development to add dominium knowledge in a feature model context. In addition, as one of its main advantages, Clafer offers tools to create model and for models validation. In comparison to Clafer, this article preserves conceptual modeling of feature models and extend them to support concepts of JPI for a complete JPI + FOP symbiosis. Developing tools for mode validity is part of future applied research works.

The rest of the article is organized as follows: next section describes main ideas of JPI and FOP along with main goals of the JPI + FOP symbiosis proposal. Section III presents JPI feature models and describes main components of this proposal. Section IV gives additional JPI feature models details. Section V applies JPI feature models on a case study previously modeled using a previous research about a mix of FOP and a traditional AOP approach. Finally, a section to conclude and mention future work and research.

II. JPI AND FOP

Just to improve and solve the known main Object-Oriented Programming OOP issues [2] [3] [12] [13] [15] , Feature-Oriented Programming FOP, and Aspect-Oriented Programming AOP appeared:

A. FOP

FOP modularizes collaboration of classes, named heterogeneous crosscutting concerns, as features, and permits step-wise development of software product lines [3] [12] [13]. As [12] [13] [15] indicate, FOP well modularizes static crosscutting concerns -new fields, methods, classes, and interfaces definition. Nevertheless, [8] [13] remark that FOP lacks adequate crosscutting modularity for software evolution since software has to change and adapt to fit non-predictable modifications.

Particularly, FOP does not modularize correctly code repetition, a recognized homogeneous crosscutting concern, [3].

B. AOP and JPI

Aspect-oriented programming, AOP, proposed by [7], permits modularizing crosscutting concerns as aspects in OOP, so aspects advise classes, i.e., like events, aspects introduce behavior on classes. Nevertheless, as [5] [6] indicate, AOP presents implicit dependencies between advised classes and aspects. First, aspects define pointcut into advised classes' behavior; so, instances of classes are completely oblivious about experimenting possible behavior and properties changes. Second, aspects can be no effective or spurious for signature changes on advised methods of target classes (*The fragile pointcut problem* [5]). Likewise, [5] [6] also indicate, aspect modules, like the AspectJ aspects [1], compromise the independent development of base code and aspects modules since developers of base code and aspects must contact each other and present a global knowledge about program modules, i.e., aspects and classes.

To isolate crosscutting concerns and getting modular AOP programs without implicit dependencies, [5] proposed JPI to introduce join point interfaces on AOP to eliminate implicit pointcut / advice association between aspects and advised classes [7]. Like classic AOP [7], for JPI applications, aspects represent crosscutting functionalities, but without a pointcut

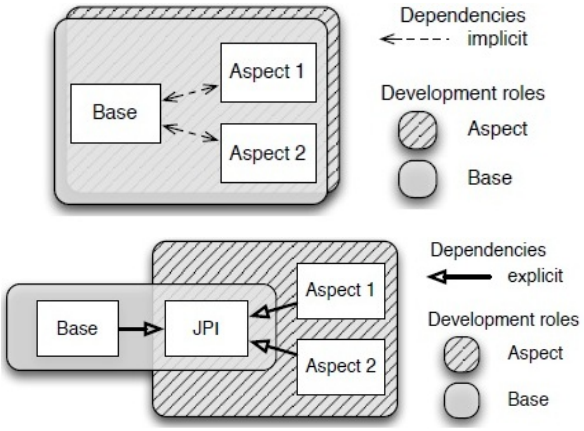


Fig. 2. Dependencies of classic AOP components and JPI solution to associate base and aspect modules (figure adapated from [6]).

PC rule. Aspects only indicate join point interfaces to implement. In addition, in JPI, for non-oblivious code, advised classes must exhibit explicitly JPI instances. Figures 2 illustrates dependencies between aspects and classes in classic AOP and JPI applications, respectively.

Even though JPI and traditional like AspectJ AOP approaches modularize adequately homogeneous crosscutting concerns, for heterogeneous crosscutting concerns modularization, these approaches do not respect the OOP nature and sometimes basic OOP principles like information hiding.

Taking in account main benefits of JPI and FOP, a JPI and FOP symbiosis seems highly adequate [2]. Following these idea, next section proposes JPI Feature Models to look for a complete JPI + FOP software development approach that takes into account main advantages and properties of each paradigm for the modular software product line production. It is important to remark, main authors of this paper already proposed and applied JPI collaboration diagram in a JPI FOP context [3].

III. JPI Feature Models

To preserve described main JPI and FOP principles, JPI Feature Model extends traditional feature model to support a new kind of feature, *JPI feature*; thus, features can exhibit one or more of these new kind of feature. In addition, a new association, a directed edge, between features and JPI features to represent *exhibits* action. Exhibits edges are edges ending in a white triangle in the target feature. Furthermore, like [10], this proposal modularizes feature models and distinguishes between *base concerns features* model *BCFM* and *crosscutting concerns features model CCFM*. Thus, features of a BCFM can exhibit CCFM units. In addition, in BCFM, it is necessary to define rules for the *exhibits* association, i.e., a pointcut rule in traditional AOP approaches. For that reason, in this proposal, defining a feature diagram for pointcut rules is not required. To relate a CCFM to BCFM, the 1st one includes an *implements* rule to define JPI feature that they implement giving details regarding the kind of advice, i.e., if its features appear before the BCFM or after it as well, to respect part of traditional AOP and JPI composition ideas. Nevertheless, an around composition, for a conceptual model like feature model is not worth since around can be expressed by a sequence of before and after advices. Thus, the graphic association is useful for documentation, and as a reference for the rules definition.

Since a CCFM is a feature model, a CCFM can exhibits other CCFM units, for which case the source are the base concerns feature model. Thus, differentiating between BCFM and CCFM depends of the context and current model associations. In general, a features module can define *implements* rules to link to usually external feature, feature of other modules, which exhibit it.

Figures 3 and 4 present rules details for exhibits and implements rules of BCFM and CCFM, respectively. Note that *JPIx* depicts a join point interface that CCFM instances *implements* and BCFM instances *exhibits*. Note that for exhibit rules, for an advised feature *F*, it is possible to know directly its children and the association kind among *F* and its children. Figure 3 is short since a BCFM B that exhibits a JPI feature only needs to indicate a kind of advice, before or after.

IV. JPI Feature Models Details

To preserve described main JPI and FOP principles, JPI Feature Models extend classic feature models to support a new kind of feature, *JPI feature*; thus, features can exhibit one or more of these new kind of feature. In addition, a new association, a directed edge, between features and JPI features to represent *exhibits* action. Exhibits edges are edges ending in a white triangle in the target feature. Furthermore, like [10], this proposal modularizes feature models and distinguishes between *base concerns features* model *BCFM* and *crosscutting concerns features model CCFM*. Thus, features of a BCFM can exhibit CCFM units. In addition, in BCFM, it is necessary to define rules for the *exhibits* association, i.e., a pointcut rule in traditional AOP approaches. For that reason, in this proposal, defining a feature diagram for pointcut rules is not required. To relate a CCFM to BCFM, the 1st one includes an *implements* rule to define JPI feature that they implement giving details regarding the kind of advice, i.e., if its features appear before the BCFM or after it as well, to respect part of traditional AOP and JPI composition ideas. Nevertheless, an around composition, for a conceptual model like feature model is not worth. Thus, the graphic association is useful for documentation, and as a reference for the rules definition.

Since a CCFM is a feature model, a CCFM can exhibits other CCFM units, for which case the source are the base concerns feature model. Thus, differentiating between BCFM and CCFM depends of the context and current model associations. In general, a features module can define *implements* rules to link to usually external feature, feature of other modules, which exhibit it.

Figures 3 and 4 present rules details for exhibits and implements rules of BCFM and CCFM, respectively. Note that *JPIx* depicts a join point interface which CCFM instances *implements* and BCFM instances *exhibits*. Note that for exhibit rules, for an advised feature F, it is possible to know directly its children and kind of association among F and its children.

A exhibits JPIx: children(B) && children(C)

A exhibits JPIx: A.children(C) | | A.children(D)

A exhibits JPIx: children.Num>5 && A.children.Num < 10

A exhibits JPIx: XOR(B) && NOT A.OR(C)

A exhibits JPIx: children(B) && chosen(B)

Fig. 3. Exhibits rule syntax for CCFM instances.

B implements JPIx : before | after

Fig. 4. Implements rule syntax for CCFM instances.

Figure 4 is short since a BCFM B that implements a JPI feature only needs to indicate a kind of advice, before or after. Figure 3 includes details about components to define rules for *implements* rules. Thus, logical connectors &&, II, NOT, =>, and <==> are usable for conjunction, for not exclusive selection, for implication, and for equivalence, respectively. Likewise, rules support relational operators >, <, =, and their combination. Moreover, rules support functions to obtain elements and properties of a current analyzed JPI feature model like children(x) to know if x is a child feature of a cited or analyzed feature and children**.**num to know the number of children of a current analyzed feature. Previous rules are also definable for other identified features. For example, for a feature *f*, f**.**children(x) is true if a feature x is a child of *f*, f**.**children**.**num to obtain the children number of *f*. To know about children set properties, XOR(f1) and OR(f1) are true if feature *f1* is either part of a XOR or part of an OR association for a current analyzed feature mother; and chosen(f) to know if feature *f* is part of a current configuration. It is important to remark that is also possible to use f1**.**OR(f2) and f1**.**XOR(f2) to refer associations in which *f1* is the father and *f2* is a child member of an OR and XOR association, respectively.

Next section describes a case study applying this proposal. Since this example was already modeled using Aspect-Oriented Feature Models [10], it is possible to highlights main pros and cons of JPI Feature Models.

V. JPI Feature Models Application

Figures 5 and 6 [10] shows part of a case study of an e-Shop already modeled using aspect-oriented feature models. In the complete example. For this example, Password represents a crosscutting concerns feature since it appears different times in the original model. This article uses this example to apply JPI Feature Models.

For the s-Shop, Figure 7 presents the feature diagram for the BackOffice concern in which feature Administration

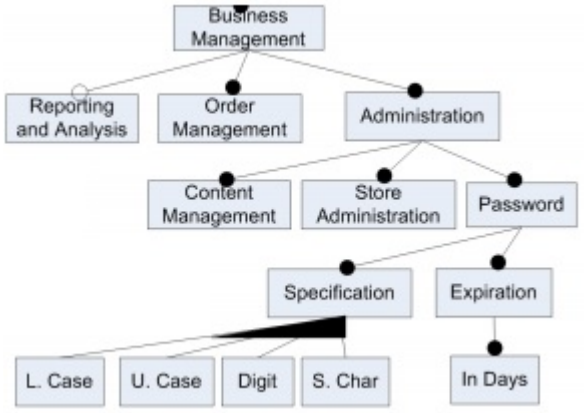


Fig. 5. Part I of Feature Diagram of e-Shop.

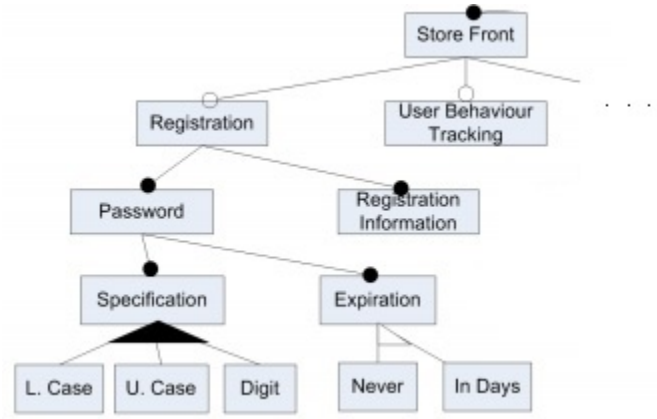
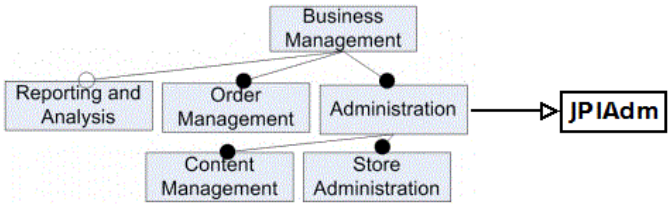


Fig. 6. Part II of Feature Diagram of e-Shop

exhibits JPIAdm. Next, Figure 8 presents the feature diagram for the UserInterface concern in which feature Registration exhibits JPIReg. Note that Figure 7 and 8 presents BCFM instances that includes a direct and simple exhibition rule, i.e., exhibition rules without formulas. Figure 9 depicts the associated aspect that implements JPIAdm and Figure 10 illustrates the CCFM that implements JPIReg, respectively. BCFM of User Interface presents a rule for exhibits: that is, Registration Information is a child feature of Registration. For CCFM instances, each figure includes a simple rule, i.e., a direct association without formulas.

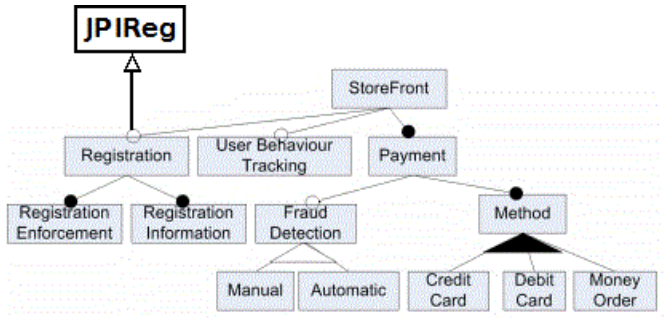
Note that JPI represent a modularization approach for which obliviousness is not present. For this reason, advised components of a model, features in this case, needs to exhibit JPI instances, and indicate rules for the advise accomplishment. Comparing both e-Shop model examples, since this proposal only advises one situation, it presents one CCFM. Thus, for other possble advices, for a different CCFM behavior, a new CCFM is required. That was not the case in [10] since it presents only one aspect. Nevertheless, that article presents 2 pointcut diagrams, one for each advised feature. A clear advantage of this proposal is the

inclusion of kind of advices before and after. In addition, this proposal defines rules for exhibits and implements, thus advised features are no more oblivious.



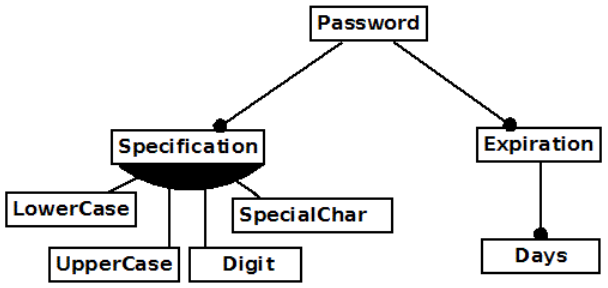
Administration exhibits JPIAdm

Fig. 7. BCFM of the e-Shop system BackOffice Concern.



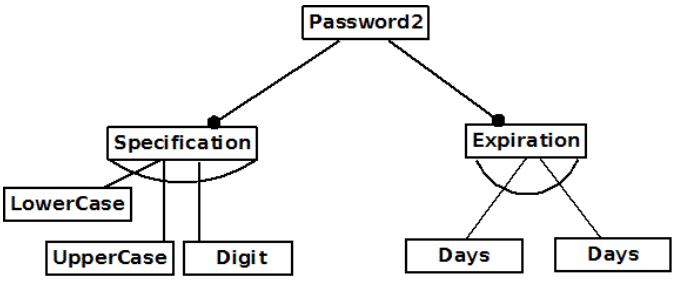
Registration exhibits JPIReg: children(Registration Information)

Fig. 8. BCFM of the e-Shop system for User Interface Concern.



Password implements JPIAdm: after

Fig. 9. CCFM of the e-Shop system for Pasword that implements JPIAdm.



Password2 implements JPIReg: after

Fig. 10. CCFM of the e-Shop system for Password2 that implements JPIReg.

VI. Conclusions

It is important to highlight that JPI Feature Model looks for a complete JPI + FOP software development paradigm; thus, for no changing view or approach in each of its stages. Clearly, since the feature model design, thinking on features, aspects, and join point interfaces represent a high requirement.

As this article mentioned and illustrated, JPI requires defining new CCFM for each advisable behavior. Regardless this "conceptual" issue, JPI permits eliminating obliviousness, and it seems promising for a JPI + FOP symbiosis approach.

As a part of our current research and future works, advancing on testing of JPI + FOP symbiosis proposal as well as extending other models approaches and tools to support this proposed paradigm and testing its effectiveness and modularity on real cases.

References

[1] Eclipse, "The AspectJ Project", Eclipse.org, 2015. [Online]. Available: https://eclipse.org/aspectj/. [Accessed: 25- Sep- 2015].

[2] C. Vidal, R. Villarroel, C. Pereira, "JPIAspectZ: A Formal Specification Language for Aspect-Oriented JPI Applications". In Proceedings of the XXXIII Chilean Computer Science Society Conference, Talca, Chile, November 2014.

[3] C. Vidal, D. Benavides, J. A. Galindo, P. Leger, "Exploring the Sinergies between Join Point Interfaces and Feature-Oriented Programming". In Proceedings of XX JISBD, Jornadas de Ingeniería de Software y Bases de Datos, Universidad de Cantabria, Santander, Spain, September 2015.

[4] D. Benavides, S. Segura, A. Ruiz-Cortés, "Automated Analysis of Feature Model 20 Years Later: A Literature Review". Journal Information Systems, 35(6), 615-836, 2010.

[5] E. Bodden, "Closure Joinpoints: Block Joinpoints without Surprises". In Proceedings of the Tenth International Conference on Aspect-Oriented Software Development, AOSD '11. ACM, New York, NY, USA, 117-128, 2011.

[6] E. Bodden, É. Tanter and M. Inostroza, "Join point interfaces for safe and flexible decoupling of aspect", *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 1, pp. 1-41, 2014.

[7] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, J. Irwin, "Aspect-Oriented Programming". In ECOOP, SpringerVerlag, 1997.

[8] K. Christian, S. Apel, D. Batory, "A Case Study Implementing Features Using AspectJ". In Proceedings of 11th International Software Product Line Conference, SPLC'07, Kyoto, Japan, 223-232, 2007.

[9] M. Antkiewicz, K. Bak, A. Murashkin, R. Olaechea, J. H. Liang, K. Czarnecki, “Clafer tools for product line engineering”. In Proceedings of the 17th Internation Software Product Line Conference, SPLC’13, Tokio, Japan, 130-135, 2013.

[10] M. Bošković, G. Mussbacher, E. Bagheri, E., D. Amyot, D. Gašević, M. Hatala, “Aspect-Oriented Feature Models (Position Paper)”. In Proceedings of the 15th International Workshop on Aspect-Oriented Modeling at 13th International Conference on Model Driven Engineering Languages and Systems, Oslo, Norway, 2010.

[11] Q. Zhang, R. Khedri, J. Jaskolka, “An Aspect-Oriented Language for Feature-Modeling”. Journal of Ambient Intelligence and Humanized Computing, 5(3), 343-356, June 2014.

[12] R. Stoiber, S. Meier, M. Glinz, “Visualizing Product Line Domain Variability by Aspect-Oriented Modeling”. 2nd International Workshop on Requirements Engineering Visualization, REV 2007, 2007.

[13] S. Apel, D. Batory, C. Kastner, G. Saake, “Feature-Oriented Software Product Lines”. Springer, 2013.

[14] S. Apel and C. Kastner, “An Overview of Feature-Oriented Software Development”. Journal of Object Technology, 8(5):49–84, 2009.

[15] S. Apel, T. Leich, G. Saake, “Aspectual Mixin Layers”. In Proceedings of the 28th International conferernce on Software Engineering ICSE, Shangai, China, 122-131.