A Model Driven Software Product Line Process for Developing Applications

Sami Ouali[1], Naoufel Kraïem[1], Zuhoor Al-Khanjari[2], and Youcef Baghdadi[2]

[1] RIADI Lab, ENSI, Compus of Manouba, Tunisia
samiouali@gmail.com, naoufel.kraiem@ensi.rnu.tn
[2] Department of CS, Sultan Qaboos University, Oman
{zuhoor,ybaghdadi}@squ.edu.om

Abstract. Software Product Line Engineering (SPLE) is an approach for software reuse. It concerns to produce customized software products as atomic or composite services to be reused in SOA-based applications. A common set of artifacts is used to build these services in a planned and managed way. The main purpose of SPLE is to explore commonalities and variabilities. The SPLE approach provides a strategic software reuse that can produce quality Software as a Service (SaaS) while cutting cost and reducing time-to-market. This paper proposes a process to construct services as Software Product Lines by using Model Driven techniques. The process combines the use of maps, visual techniques for SPL modeling, especially features diagrams and MD techniques. In addition to the process, we have developed a tool to support map, feature, and class diagrams modeling.

Keywords: Model Driven, Software Product Line, Variability, Process, Tool.

1 Introduction

(SPLE) is an approach for software reuse. It is recognized as a successful approach to reuse in software development [1, 2]. It concerns with producing customized software products as atomic or composite services to be reused in SOA-based applications [23, 24]. SPLE allows companies to realize significant improvements on productivity. The purpose of this approach exploits the commonalities between products while preserving the ability to vary the functionality between these products. The difference between products is their variability, a key success in product lines [3]. SPLE distinguishes two layered levels of engineering [4]: Domain Engineering (DE) and Application Engineering (AE). DE deals with the identification of commonalities and variabilities among products. AE is used to develop a new product from a PL and the results from DE level are used to derive a specific product.

In the proposed process, we focus on both levels of engineering. First, we build the SPL at DE level. Then, we derive a specific product. The problem that we try to tackle in this paper is the discordance goal between the developed products and the real needs of its users.

X. Franch and P. Soffer (Eds.): CAiSE 2013 Workshops, LNBIP 148, pp. 447–454, 2013.

This paper presents a meta-model and a prototyping tool for requirement and feature modeling. The tool implements various models to assist stakeholders in the process of product configuration for SPLs. This paper describes an Eclipse plug-in for map, feature, and class modeling to model SPL, which assists SPL designer in the construction of SPL and the derivation of particular product. Providing tool as an Eclipse plug-in would facilitate the integration of these kinds of modeling with a development environment. The tool, itself, is built by using two Eclipse plugins: EMF, and GMF which provide many possibilities through extensions points.

The remainder of this paper is organized as follows. Section 2 summarizes a meta-model for intention and feature modeling in the context of SPLs. The proposed Model Driven for SPL process is described in section 3. The supporting tool is presented in section 4. Section 5 presents some related work. Finally, a conclusion section highlights our contribution and presents some research perspectives.

2 A Meta-model for SPL

This section describes a meta-model that synthesizes the interesting points such as SPL, intention, features, etc. We choose to transform this meta-model into a UML profile to (i) facilitate the integration with UML models, and (ii) use it in our MD approach. We try throw this meta-model to facilitate the relation between requirements model (MAP), features model and classes diagram. This meta-model is used in our process and in the generation of our tool support.

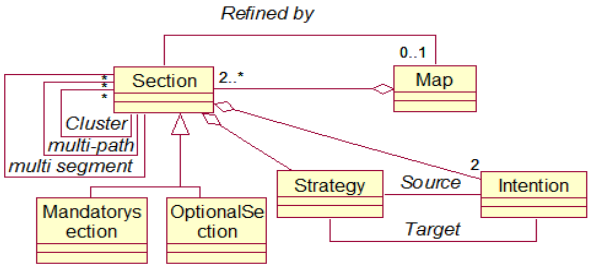


Fig. 1. Modification applied on map meta-model

A MAP [5] involves two or more sections. Each section is composed of two intentions and one strategy (these concepts are presented in section 3.1). A relationship between sections can be a cluster, multi-path or multi segment. To further use the meta-model in our process, we make a modification, as shown in Figure 1, by adding two types of sections: 'OptionalSection' and 'MandatorySection'. These new sections are added for the mapping between map and features model.

A PL contains features, as shown in Figure 2. A product belongs to one PL. It is also composed of features that check some constraints (an exclusion and required relation) through the conflict and require relationships. Figure 2 presents the different concepts of feature Model. The possible relationships between features are presented

by FeatureGroup. There are three relationships between features: 'And', 'Xor', and 'Or'. The relationships between a parent feature and its child features are categorized as: 'Mandatory' or 'Optional'.

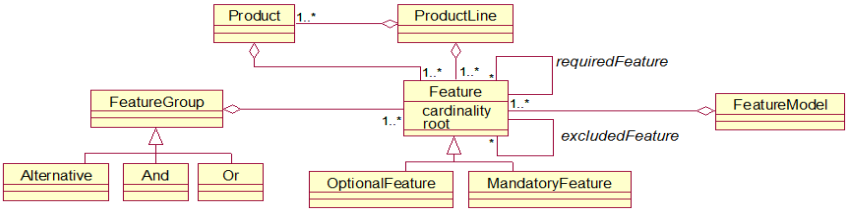


Fig. 2. Relationship between product line and features

Figure 3 presents the relation between the features elaborated in the analysis domain and the components constructed in implementation domain; the goal of implementation domain is to develop the components participating in the architecture of the SPL). A feature is associated to a component. There are two kinds of components: composite or leaf, i.e. simple component. A leaf is a non decomposable component. It has one or more possible implementation. A composite component is decomposed into subcomponents.

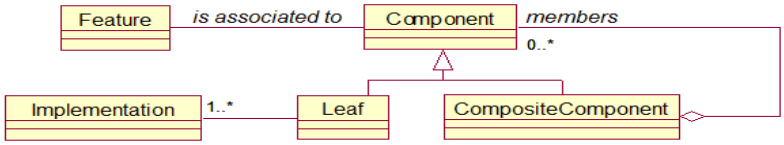


Fig. 3. Relationship between feature and component

Another alternative to the implementation of assets with components is the use of services as shown in Figure 4. There are two kinds of services: atomic service and composite service. An atomic service cannot further be decomposed. A composite service is composed of other services. A feature can be associated to a service.

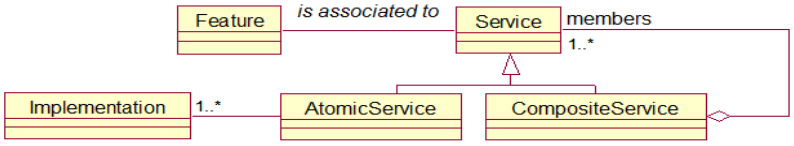


Fig. 4. Relationship between feature and service

3 Model-Driven SPL Process

This section presents the notations used in the model-driven SPL process. These are map model and features model. It also presents some rules and patterns for mapping between map, features, and class model. Both DE and AE are covered in the proposed as shown in Figure 5. The DE process deals with the creation of core assets. These core

assets are PL requirements, analysis models, PL architecture, reusable components and services. This process concerns with the elicitation of intentions and strategies using the map for the design of user's requirements. In the modeling of the requirements, SPL features intentionality is represented as maps [22]. Then, features models derive class diagrams in the design view that used further used to obtain a component architecture or service architecture throw through components or services repository. The component architecture view deals with the creation of the interface type and a component. The interfaces contain the signature of operations. The generated component implements the required and provided interfaces. Service architecture view is also realized by generation of services from design view. The AE process deals with product derivation from the composition of existing artifacts created in the DE (existing components or services present in the components repository or services repository). It exploits the variability of the PL, which allows the satisfaction of users needs.

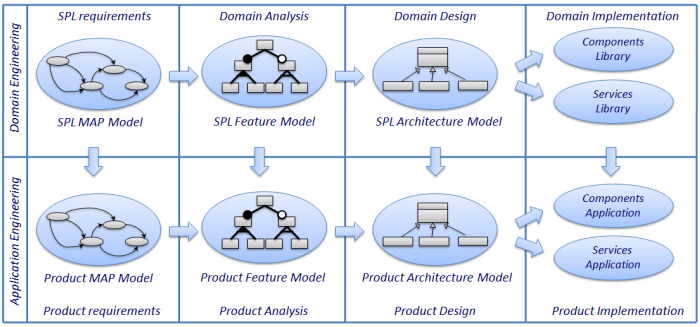


Fig. 5. Approach for SPL construction

3.1 MAP Model

A MAP [5] is a process model expressed in a goal-driven perspective that can provides a representation of a multi-facetted purpose based on a non-deterministic intentions and strategies. The directed nature of the graph shows which goals can follow which one. MAP is considered as intention-oriented process modeling that follows the human intention of achieving a goal [12]. A map is a directed graph from 'Start' to 'Stop', as the strategy shows the flow from the source to the target intention. It is composed of several sections. Intentions are represented as nodes of the graph. Strategies are the relationships between intentions. The key concepts of map are intention, strategy and section. An intention is a goal [6] that expresses a state that is expected to be reached or maintained. A strategy is a way or a means to achieve an intention. A section is a combination of two intentions linked by a strategy.

3.2 Feature Model

Feature modeling is a domain analysis technique, part of the Feature Oriented Domain Analysis (FODA) method [7], for developing software for reuse. This method has been applied in many domains such as telecom systems [8, 9], network protocols [10], and embedded systems [11]. Using feature modeling can help in generating domain

design in SPL. Feature Models allow a representation of all possible products in a SPL, in terms of features. It represents the description of the mandatory features that are present in all the products of the PL along with variant features or optional features that do not appear in all the products.

In a feature model, we can find hierarchical relationships between features. The relationships between a parent feature and its sub features are categorized as mandatory features that are required in all the products, or optional features. A child feature can only appear in the products where its parent feature appears. Relationships between features are mandatory, optional, Xor, and-relationship and or. A features model defines constraints which are compatibility rules. These rules refer to some restrictions in features combinations. They are two types of constraints: requires and excludes.

3.3 Patterns to Automate Transformations

Our process focuses on DE and AE, using goals and features model to establish architecture model. In this architectural model, we deal only with structural view (class diagram). We define some rules to try to automatically derive class diagram from requirement model. However, we recognize that the transformations that we propose cannot be univocal. We distinguish different kinds of transformations: (i) SPL requirements are mapped into feature models, and (ii) the structural information of the feature models is mapped into class diagrams. These transformations are implemented using QVT. We begin with the mapping between requirement model and analysis model, i.e. from MAP model to feature model, by using some transformations rules:

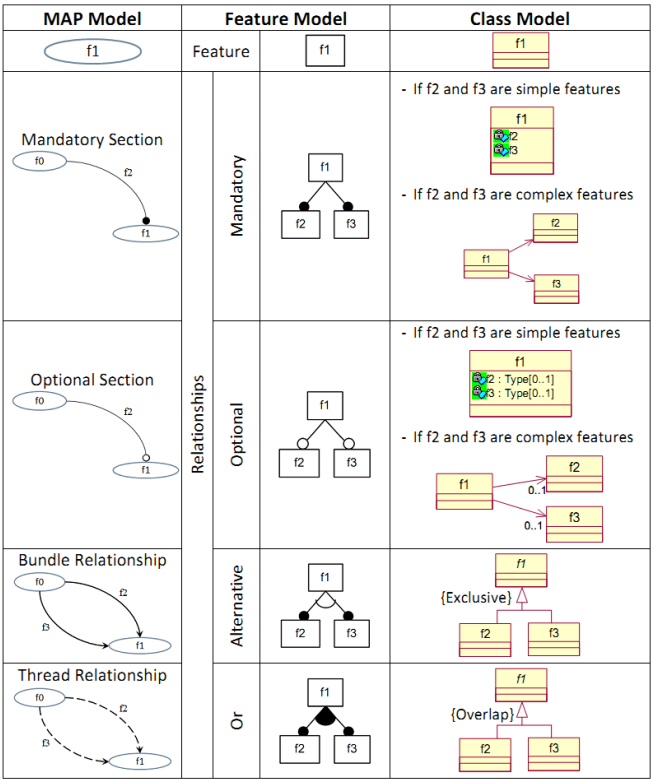
- A thread relationship in a map model representing a AND/OR relation between two strategies is equivalent to an Or-features-group
- A bundle relationship representing a XOR relation between two strategies is equivalent to an Alternative-features-group
- For mandatory and optional features, we have defined new concepts in the MAP meta-model: mandatory and optional section. The same graphical representation in the feature model is used for representing mandatory and optional sections.

Feature model are mapped into classes and attributes as shown in Table 1:

- A feature in the model tree can be simple or complex feature.
- The leaves of the tree can be simple features, in this cases a simple feature is mapped into an attribute of a parent class.
- A non-atomic feature is mapped into a class representing this feature.
- Mandatory features imply a unidirectional association with the right cardinality.
- The optional features are mapped as associations with 0..1 as cardinality.

For alternative and OR groups of features, generalization concept is applied, where an alternative group of features is represented by a generalization of classes with exclusive as constraint between the subclasses, whereas Or group is represented by a generalization of classes with overlap as constraint between subclasses.

Table 1. Rules to automate transformations



4 Supporting Tool

Based on the meta-model presented in section 2, we developed an interactive, visual, tool for requirement modeling, feature capture, and architecture modeling. The tool will be extended to Feature Configuration. Our plug-in implements map modeling, cardinality-based feature modeling through transformations and architectures of SPL construction. This tool is based on eclipse plug-ins. The tool uses the multiple extension possibilities of offered by eclipse platform through two plug-ins: Eclipse Modeling Framework (EMF), and Graphical Modeling Framework (GMF).

These plugins provide many possibilities through extensions points, which can reduce the development effort. These plugins helps in constructing our model-driven approach. The proposed tool is built-by re-using these two Eclipse plugins. Providing a supporting tool for map, feature, and class modeling as an Eclipse plug-in is of paramount importance for with the integration of these kinds of modeling as part of a development environment. Object Constraint Language is used to describe constraints, rules and specifications. OCL statements can be specified as part of EMF Model or invoked directly from programming language (java).

5 Related Work

Several authors have proposed approaches relating to feature models and product architectures. The mapping between requirements and design is a complex task because of the flexibility and adaptability of the SPL, the different technology possibilities, etc. Van Lamsweerde [13] derives software architectures from the formal specifications of a system goal model. Bragança et al. [20] propose to obtain features from use case models. Laguna et al. [21] proposes some patterns to obtain design view and use case models from features models.

Many proposals express variability with UML models. [18] propose the extension of UML Meta model by the adding of new relationships "option" and "alternative". Clauß proposes the use of stereotypes to express variability [19].

Many propositions are concerned with the construction of editor for features modeling. AmiEddi [14] supports feature modeling notation [4] and didn't support feature and group cardinalities. CaptainFeature [15] implements a cardinality-based notation. Some other works try to integrate feature-based configuration like ConfigEditor [11].

We also found some commercial tools (Pure::Variants [16] or GEARS [17]). Pure::Variants support feature modeling and configuration by using a tree-view render without feature cardinalities. It allows modeling of constraints between features and uses Prolog-based constraint solver for the configuration. GEAR allows the modeling and configuration of software variants.

6 Conclusion and Future Work

There is growing interest on the topic of SPL construction to derive particular product. We have aimed at fulfilling stakeholder's requirement of the SPL and the particular requirements. We have focused on the mapping of goal-oriented requirements into software architectures. First, we have defined a Meta model to represent the different concepts related to the construction of SPL. Then, we have defined a process to (i) construct SPL, (ii) derive particular product throw configuration and model derivation. We have emphasized the construction of services for SOA-based applications. The process generates design views by using class diagram derived from a requirement model. The process is supported by rules and mapping patterns and realized by a tool using EMF and GMF plugins to facilitate the construction of SPL.

Our future works includes (i) the formal validation of the proposed mapping patterns and (ii) the generation of assets of the SPL. These assets will be PL requirements, analysis models, PL architecture, reusable components, and services.

References

1. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Boston (2001)
2. Bosch, J.: Design & Use of Software Architectures, Adopting and Evolving a product-line approach. Addison-Wesley (2000)

3. Van Grup, J.: Variability in Software Systems, the key to software reuse (Thesis). University of Groningem, Sweden (2000)
4. Czarnecki, K., Eisenecker, W.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)
5. Rolland, C., Prakash, N., Benjamen, A.: A Multi-Model View of Process Modeling. Requirements Engineering Journal 4(4), 169–187 (1999)
6. Jackson, M.: Software Requirements & Specifications – a Lexicon of Practice, Principles and Prejudices. ACM Press, Addison-Wesley (1995)
7. Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90TR -21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (1990)
8. Griss, M., Favaro, J., Alessandro, M.: Integrating feature modeling with the RSEB. In: Proceedings of the Fifth International Conference on Software Reuse (1998)
9. Lee, K., Kang, K.C., Lee, J.J.: Concepts and guidelines of feature modeling for product line software engineering. In: Gacek, C. (ed.) ICSR 2002. LNCS, vol. 2319, pp. 62–77. Springer, Heidelberg (2002)
10. Barbeau, M., Bordeleau, F.: A protocol stack development tool using generative programming. In: Batory, D., Blum, A., Taha, W. (eds.) GPCE 2002. LNCS, vol. 2487, p. 93. Springer, Heidelberg (2002)
11. Czarnecki, K., Bednasch, T., Unger, P., Eisenecker, U.: Generative programming for embedded software: An industrial experience report. In: Batory, D., Consel, C., Taha, W. (eds.) GPCE 2002. LNCS, vol. 2487, pp. 156–172. Springer, Heidelberg (2002)
12. Soffer, P., Rolland, C.: Combining Intention-Oriented and State-Based Process Modeling. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 47–62. Springer, Heidelberg (2005)
13. van Lamsweerde, A.: From system goals to software architecture. In: Bernardo, M., Inverardi, P. (eds.) SFM 2003. LNCS, vol. 2804, pp. 25–43. Springer, Heidelberg (2003)
14. Selbig, M.: A feature diagram editor: analysis, design, and implementation of its core functionality. Kaiserslautern, Germany (2000)
15. Bednasch, T., Endler, C., Lang, M.: Captain Feature (2002-2004) Tool available on Source Forge at, `https://sourceforge.net/projects/captainfeature/`
16. Pure-systems GmbH. Variant Management with Pure:Consul. Technical White Paper (2003), `http://web.pure-systems.com`
17. Krueger, C.W.: Software mass customization. White paper, `http://www.biglever.com/papers/BigLeverMassCustomization.pdf`
18. Von der Massen, T., Lichter, H.: RequiLine: A Requirements Engineering Tool for Software product lines. In: Software Product-Family Engineering, Siena, Italy (2003)
19. Clauß, M.: Generic modeling using Uml extensions for variability. In: Workshop on Domain Specific Visual Languages at OOPSLA (2001)
20. Bragança, A., Machado, R.J.: Automating Mappings between Use Case Diagrams and Feature Models for Software Product Lines. In: Proceedings of SPLC, Kyoto, Japan (2007)
21. Laguna, M.A., González, B.: Product Line Requirements: Multi-Paradigm Variability Models. In: Proceedings of the 11th Workshop on Requirements Engineering, Spain (2008)
22. Ouali, S., Kraiem, N.: From Intentions to Software Design using an Intentional Software Product Line Meta-Model. In: The Proceeding of the 8th International Conference on Innovations In Information Technology. Al Ain, UAE (2012)
23. Al Rawahi, N., Baghdadi, Y.: Approaches to Identify and Develop Web Services as Instance of SOA Architecture. In: Proc. of the IEEE-ICSSSM 2005, China (2005)
24. Baghdadi, Y.: A Methodology for Web services-based SOA realization. International Journal of Business Information Systems 10(3), 264–297 (2012)