Global Journal of Computer Science and Technology
Volume 12 Issue 1 Version 1.0 January 2012
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Intentional Software Product Line

By Sami Ouali, Naoufel Kraiem, Henda Ben Ghezala
National School of Computer Sciences University, Tunisia

Abstract - Software product line engineering optimizes the development of individual systems by leveraging their common characteristics and managing their differences in a systematic way. These differences are called variabilities. We argue that it is difficult for business people to fully benefit of the SPL if it remains at the software level. The paper proposes a move towards a description of software product line in intentional terms, i.e. intentions and strategies to achieve business goals. We present ISPL, the model to describe intentional Software Product Line. Thereafter, we propose our process to show how to use this model.

Keywords : *Software Product Line, variability, intentional level, comparison framework, features modeling and metamodels.*

GJCST Classification: *D.4.6,*

Strictly as per the compliance and regulations of:

Intentional Software Product Line

Sami Ouali [α], Naoufel Kraiem [Ω], Henda Ben Ghezala [β]

Abstract - Software product line engineering optimizes the development of individual systems by leveraging their common characteristics and managing their differences in a systematic way. These differences are called variabilities. We argue that it is difficult for business people to fully benefit of the SPL if it remains at the software level. The paper proposes a move towards a description of software product line in intentional terms, i.e. intentions and strategies to achieve business goals. We present ISPL, the model to describe intentional Software Product Line. Thereafter, we propose our process to show how to use this model.

Keywords : Software Product Line, variability, intentional level, comparison framework, features modeling and metamodels.

I. Introduction

Software product line engineering optimizes the development of individual systems by leveraging their common characteristics and managing their differences in a systematic way (Clements & Northrop, 2001). These differences are called variabilities. In software product line engineering, two kinds of variability can be distinguished: product line variability and Software variability. Software variability refers to the ability of a software system to be efficiently extended, changed, customized or configured for use in a particular context (Svahnberg et al., 2005). While product line variability describes the variation between the systems that belong to a product line (Coplien et al., 1998; Pohl et al., 2005; Kang et al., 2002) in terms of properties and qualities, like features that are provided or requirements that are fulfilled. Defining product line variability concerns the determination of what should vary between the systems in a product line. In SPLE, single system can be built rapidly from reusable assets, such as a set of components.

The framework analysis which we proposed in our previous work (Ouali et al., 2011) allows us to identify many drawbacks of existing SPL construction methods. In these methods, apart requirement approaches ones, the problem is the matching between users' needs and the product offered by developers. Many writers have observed that there is a "conceptual mismatch" (Woodfield, 1997; Kaabi, 2007). The position adopted in this paper is to suggest a move to intention-driven SPL to bridge the gap between high level users' goals and low level software product line obtained. We present in this paper a model for intentional SPL modeling.

Our process is based on goal modeling, feature modeling and metamodels. Goal models model stakeholder intentions to fulfill the system-to-be. Feature modeling allows us to model the common and variable properties of product-line members throughout all stages of product-line engineering. Metamodels allow the expression of common and variable characteristics of a set of applications. A metamodel represents the concepts, relationships, and semantics of a domain.

This paper is organized as follows. A brief description of different concept concerning software product line and variability is presented in the next section. Our previous work, which is the comparison framework, is described in section 3. An intentional software product line model is presented in section 4. In section 5 we present our proposed process. The section 6 concludes this work with our contribution and research perspectives.

II. Software Product Line and Variability Concepts

Software product lines are recognized as a successful approach to reuse in software development (Clements & Northrop, 2001; Bosch, 2000). The idea behind software product line is to economically exploit the commonalities between software products, but also to preserve the ability to vary the functionality between these products. These differences refer to the variability which is a key success factor in product lines and reuse. This approach is based on the undertaking of the development of a set of products as a single, coherent development activity. Indeed, products are built from a collection of artifacts from a core asset base that have been specifically designed for use. Core assets include not only the architecture and its documentation but also specifications, software components, tools…

Variability is the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context (Van Grup, 2000). Another definition presents variability as the ability of a system, an asset, or a development environment to support the production of a set of artifacts that differ from each other in a preplanned fashion (Czarnecki & Eisenecker, 2000). In this definition variability means the ability of a core asset to adapt to usages in the different product contexts that are within the product line scope. Indeed,

Author [α] : National School of Computer Sciences University, Tunisia. Telephone: +21623695033 E-mail : samiouali@gmail.com
Author [Ω] : Higher Institute of Computer Science of El Manar and RIADI Labs, Tunisia. E-mail : Naoufel.kraiem@ ensi. rnu.tn
Author [β] : department of Informatics at the National School of Computer Sciences of Tunis and the director of RIADI Labs, Tunisia. E-mail : henda.bg@cck.rnu.tn.

variations in a product line context must be anticipated.

The purpose of Variability modeling is to present an overview of a product line's commonality and variability. Variability modeling terms concerns also commonality modeling. The content of a variability model serves as a basis for defining variability within the artifacts that make up the product-line infrastructure as well as for configuring individual product instances and deriving them from the infrastructure.

SPL engineering is defined (Czarnecki & Eisenecker, 2000) by distinguishing two levels of engineering: Domain Engineering and Application Engineering as presented in Fig. 1.

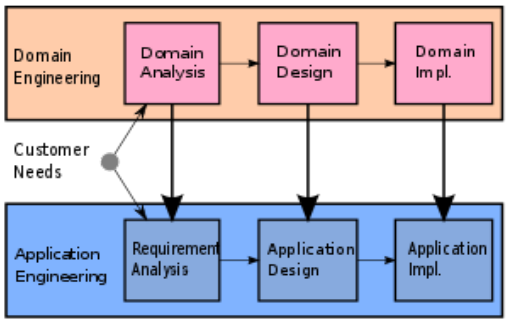

Fig.1 : SPL Engineering levels

Domain Engineering corresponds to the study of the area of product line, identifying commonalities and variabilities among products, the establishment of a generic software architecture and the implementation of this architecture. Indeed, the domain engineering consists on the construction of reusable components known as asset which will be reused for the products building.

Application Engineering is used to find the optimal use for the development of a new product from a product line by reducing costs and development time and improve the quality. At this level, the results of the domain engineering are used for the derivation of a particular product. This derivation corresponds to the decision-making towards the variation points.

In the literature, the majority of variability research concerns requirements and architecture. But some works deals with implementation, verification and validation, traceability and software product line management. The literature basically proposes methods or techniques that address only a specific portion of SPL development.

III. Comparison Framework

We have elaborated a framework to compare different approaches for the construction of SPL. The idea is to consider a central concept (SPL) on four different points of view. Defining a comparison framework has proved its effectiveness in improving the understanding of various engineering disciplines (process, requirements, information systems…) (Rolland, 1998; Jarke & Pohl, 1993). Therefore, it can be helpful for the better understanding of the field of engineering SPLs. As a result, our framework (Fig. 1) is presented in (Ouali et al., 2011).

The framework analysis allows us to identify the following main drawbacks of existing SPL construction methods. We realize that we have a lack of sufficient tool support for them and for their interactivity with their users. The SPL approaches themselves are not enough automated for deriving automatically a product from a SPL. In addition, these methods didn't cover all aspects of SPL engineering. Indeed, every method tries to focus on a particular part of SPL construction process. Finally, in these methods, apart requirement approaches ones, the problem is the matching between users' needs and the product offered by developers. Many writers have observed that there is a "conceptual mismatch" (Woodfield, 1997; Kaabi, 2007).

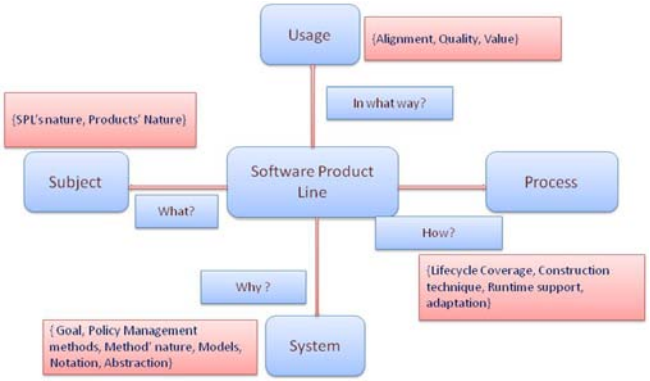

Fig.2 : Software Product Line comparison framework evolution

We try in the next section of this paper to resolve this last drawback by the proposal of a model for intentional SPL modeling. We try to establish the matching between users' needs and the product offered by developers by the expression of users' needs in an intentional way.

IV. Intentional Software Product Line Meta-Model

This section describes a meta-model synthesizing the different interesting points that we previously identified after a state-of-the-art (software product line, intention, feature…). We chose to transform this meta-model into a UML profile to facilitate the integration into UML models and to use it in our MDA approach.

a) Meta-model Description

As depicted in Fig. 3, a *product line* contains *features*. A *product* belongs to one *product line* and is composed of *features*. These *features* associated to a product must check some constraints (mutual exclusion

and require relation) throw the conflict and require relationships. The recommends relationship concerns another feature that could be pertinent.

An intentional software product line is a set of *features* captured at the business level, in business comprehensible terms and described in an intentional perspective. In this perspective, we focus on the *intention* it allows to achieve rather than on the functionality it performs. A *feature* is a set of related *requirements* that allows the user to satisfy an *intention*. We have two specializations of features which are *MandatoryFeature* and *VariantFeature*. Mandatory features are features which must be present in every configuration of a product from the product line.

A variant feature is modeled as a set of variation point. The metamodel allows atomic variation points *(Variant)* or composite ones *(Composite VariationPoint)* for a variant feature. We use the composite pattern to compose a variation point.

In our meta-model, we use a part of an existing meta-model map (Rolland et al., 1999c) which is a Process Model in which a non-deterministic ordering of intentions and strategies has been included. Map is a labeled directed graph with intentions as nodes and strategies as edges between intentions. A *map* consists of a number of sections. Each section is a triplet formed by a source *intention*, a target *intention* and a *strategy*. A *strategy* is a manner to achieve an *intention*.

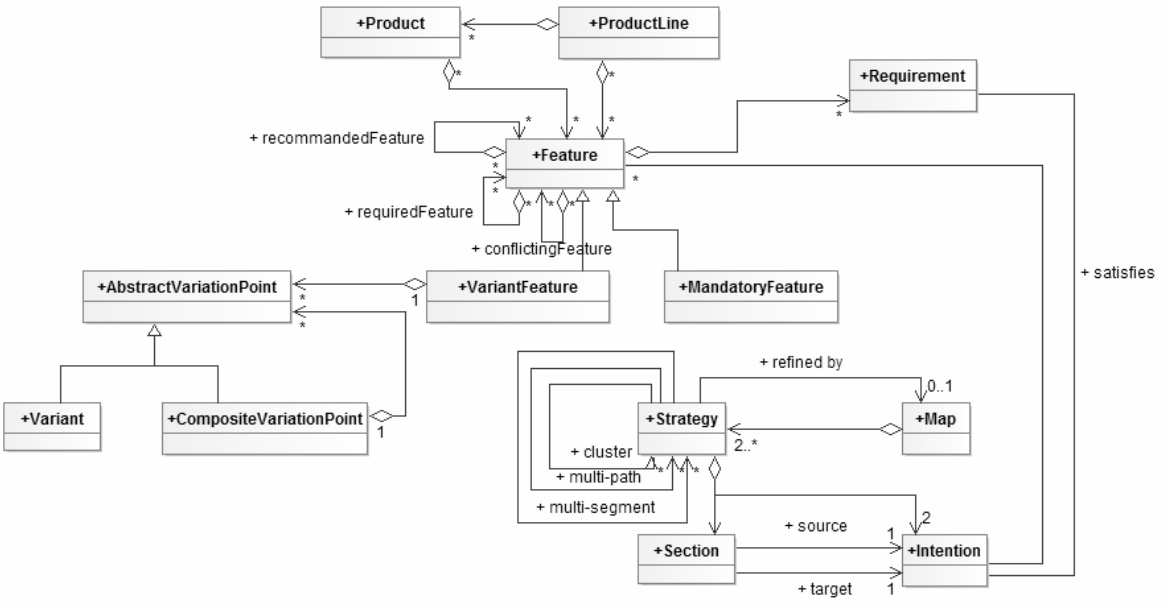

Fig.3 : Above is the example of single column image. Images must be of very high quality.

V. Proposed Process

To avoid the drawbacks of the existing methods, we try to propose a new process for the construction of SPL. This process is a flexible approach for automatically building SPL based on variability models. This process is based basically on goal modeling, features modeling, metamodels, constraints…

In our process, we try to cover domain engineering and application engineering. The domain engineering process involves the creation of core assets. In this process, our interest concerns the elicitation of intentions and strategies using the MAP for the design of users' requirements. A map is a process model expressed in a goal driven perspective which can provides a process representation system based on goals and strategies. The directed nature of the graph shows which goals can follow which one. MAP is considered as Intention-oriented process modeling which follows the human intention of achieving a goal as a force which drives the process (Soffer & Rolland, 2005). Having represented software product line features intentionality as maps, we will proceed in our process to determine features and their composition according to the Intentional Software Product line. This approach is presented in Fig. 4. Users' intentions are captured and modeled using Map Model to obtain an SPL Model. This model contains an intentional view. Variability in intentional software product line modelling is mandatory and due to the need to introduce flexibility in intention achievement. We use features diagrams to model variability in software product line. We try to capture commonality and variability of domain and to reuse it for the derivation of a specific requirement model in application Level.

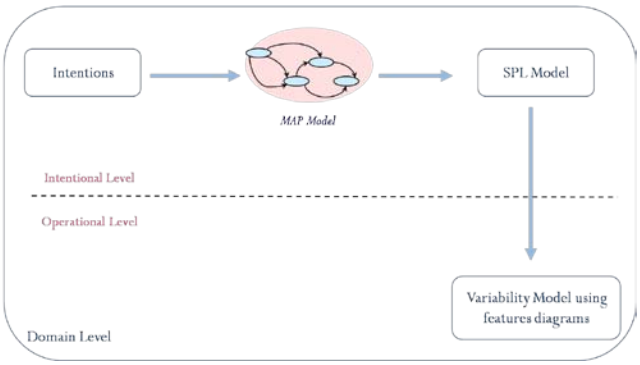

Fig.4 : Domain Level Engineering using Intentional Software Product Line Model

We try to manage variability in SPL construction process (functions, structures, behaviors, technologies). Our strategy follows feature modeling approach, MDA approach and the managing of the constraints. We base our work on the creation of features models representing the SPL structure. We use state machine to model the behavior in the SPL. This process is based on the automatic transformation of models until obtaining executable applications. The process is flexible because SPL developer has a lot of possibilities for the creation of SPL and its constraints. It permits the generation of a flexible SPL suitable to the users' requirements elicited in the beginning of the creation process and new ones.

VI. Conclusion

In this paper, our contribution was the proposal of a model combining software product line, variability, requirements and intentions. This suggested model clarifies the notion of an intentional software product line to model SPL in intentional context. It was build to respond to the following purpose: to focus on the intention it allows to achieve rather than on the functionality it performs. An intentional software product line is captured at the business level, in business comprehensible terms and described in an intentional perspective. This model will be useful to improve the method used for software product line construction by avoiding the conceptual mismatch. We try to establish the matching between users' needs and the product offered by developers by the expression of users' needs in an intentional way.

In this paper, we have presented a proposal to manage variability during the SPLs construction process using a MAP for goals modeling, features diagrams allows us to model the common and variable properties of product-line members throughout all stages of product-line engineering, metamodels allow the expression of common and variable characteristics of a set of applications.

Our future work will be the proposal of a tool support to improve interactivity with users and to cover the overall lifecycle of SPL. This tool support will be based on Eclipse plug-in for feature modeling using the Eclipse Modeling Framework (EMF), which significantly reduced our development effort. Our tool support is based on generative development for goal modeling, feature modeling and metamodels. Integrating goals modeling, feature modeling and metamodels as part of a development environment helps to optimally support modeling variability in different artifacts including implementation code, models, documentation, development process guidance...

References References Referencias

1. Clements, P. & Northrop, L. (2001). *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA.
2. Svahnberg, M., Van Gurp, J., Bosch J. (2005). A taxonomy of variability realization techniques, In: *Software Practice & Experience*, Vol. 35, No. 8, pp. 705–754.
3. Coplien, J., Hoffman, D., Weiss, D. (1998). Commonality and variability in software engineering. In: *IEEE Software*, Vol. 15, No. 6, pp. 37 – 45.
4. Pohl, K., Böckle, G., Van der Linden, F. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer.
5. Kang, K. C., Lee, J., Donohoe, P. (2002). Feature-oriented project line engineering. In: *IEEE Software*, Vol. 19, No. 4, pp. 58–65.
6. Woodfield, S. N. (1997). *The Impedance Mismatch between Conceptual Models and Implementation Environments*, ER'97 Workshop on Behavioral Models and Design Transformations: Issues and Opportunities in Conceptual Modeling, UCLA, Los Angeles, California.
7. Kaabi, R. (2007). *Une Approche Méthodologique pour la Modélisation Intentionnelle des Services et leur Opérationnalisation* (Thèse de doctorat). Sorbonne: Université de Paris I.
8. Ouali, S., Kraïem, N. and Ben Ghezala, H. (2011). A Flexible Process for SPL construction. *Journal of Computer Science and Engineering (JCSE)*, Volume 8, Issue 1.
9. Rolland, C. (1998). *A Comprehensive View of Process Engineering*, Proceeding of the 10th International Conference CAiSE'98, LNCS 1413, Springer Verlag Pernici, C. Thanos (Eds), Pisa, Italie, p. 1-24.
10. Jarke, M. & Pohl, K. (1993). *Requirements Engineering: An Integrated View of Representation, Process and Domain*, in Procedings 4th Euro. Software Conf., Springer Verlag.
11. Bosch, J. (2000). Design & Use of Software Architectures, Adopting and Evolving a product-line approach, Addison-Wesley, ISBN 0-201-67494-7.
12. Van Grup, J. (2000). *Variability in Software Systems, the key to software reuse* (Thesis). Sweden: University of Groningem.

13. Czarnecki, K. and Eisenecker, W. (2000). Generative Programming: Methods, Tools, and Applications. Addison-Wesley.
14. Rolland, C., Prakash, N. and Benjamen, A. (1999c). A Multi-Model View of Process Modelling, *Requirements Engineering Journal*, 4:4, 169-187.
15. Soffer, P. & Rolland, C. (2005). *Combining Intention-Oriented and State-Based Process Modelling*, Proceedings of the International Conference on ER'05, LNCS, pp47-62.

This page is intentionally left blank