Dynamic Large Scale Product Lines through Modularization Approach

Asmaa Baya, Bouchra El Asri, Ikram Dehmouch and Zineb Mcharfi
IMS Team, SIME Laboratory, ENSIAS, University Mohammed V Rabat, Rabat, Morocco

Keywords: Software Product Line, Feature Model, Modularization, Large Scale Systems.

Abstract: Software product line (SPL) now faces major scalability problems because of technical advances of the past decades. However, using traditional approaches of software engineering to deal with this increasing scalability is not feasible. Therefore, new techniques must be provided in order to resolve scalability issues. For such a purpose, we propose through this paper a modularization approach according to two dimensions: In the first dimension we use Island algorithm in order to obtain structural modules. In the second dimension we decompose obtained modules according to features binding time so as to obtain dynamic sub-modules.

SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Software Product Line (SPL) has been an attractive approach for medium and large companies. Because, it allows the optimization of product development process and planed reuse, through the identification and reuse of common features that are shared by several products. However, there is a risk of not getting a viable return on investment if the pre-developed assets are not sufficiently reused.

In principle, a product line presents concepts of a given business domain. But recently product lines are being used not only to describe variability of a well defined domain, but were extended to other types of requirements such as variability of contexts. As a result, SPL engineering now faces major scalability issues. Indeed, creating and managing such large product lines models is becoming a very complex activity, time consuming and error prone. That's why, effective approach for separation of concerns is becoming a major challenge.

In this paper, we are interested in looking at how large and complex product line can be managed through an appropriate representation mechanism. To achieve that, we explore the possibility of product line modularization approach that gives due consideration to large scale particularities.

The remainder of this paper is organized as follows. In Section 2, we give an overview of large software product lines, and then we present some main concepts like variability and separation of concerns. In section 3, we present our approach for modularization of product lines. Finally, Section 5 concludes the paper.

2 BACKGROUND

2.1 Large Software Product Lines

In the mid of 1990s, software product lines began to draw attention of researchers community and became an independent approach of software engineering (Maier, 1998; Benavides, Segura and Ruiz-Cortés 2010). Over last years, SPL was integrated massively in several business domains such as mobile phones, automotive systems, aerospace, etc. (Pohl, Bockle and Linden 2005; Parra 2011).

Product line engineering is a process that delivers reusable components, which can be reused to develop new applications for the domain instead of developing them from scratch. So, the main advantage that promote the use of software product lines is the possibility of planned, proactive and systematic reuse of the common artifacts (also called core assets), where related products are treated as a product family.

Over the last decades, many technical advances of software engineering were introduced to SPL. As

Baya A., El Asri B., Dehmouch I. and Mcharfi Z..
Dynamic Large Scale Product Lines through Modularization Approach.
DOI: 10.5220/0005460204390444
In *Proceedings of the 17th International Conference on Enterprise Information Systems* (ICEIS-2015), pages 439-444
ISBN: 978-989-758-097-0
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

a result, an increasing number of concerns emerge, and large scale is becoming a main characteristic of all system dimensions: lines of code, number of stakeholders employing the system for different purposes, amount of data stored, etc. So, models are becoming too large for human cognitive abilities, and too imprecise for automated reasoning.

The main complexity introduced by SPL compared to other software engineering approach, is the management of variability. This variability increase in large scale since, systems evolve continuously through successive iterations along multiple dimensions, such as: new stakeholders' requirements, emergence of operating environment constraints, incorporation of new technologies, etc. Consequently, classic management approaches of SPL are hardly applicable; instead, new approaches must be based on:

- Flexible and dynamic structure in order to accommodate changes at all levels (Rosenmüller 2011),
- Agile approaches with several iterations is required for dealing with emergent requirements (Urli et al. 2012),
- Traceability must be taking into account in order to revert the changes made by an evolution through the system life cycle (Lamb, Jirapanthong and Zisman 2011).

2.2 Variability in Space & Variability in Time

Variability is defined as the ability of a software system or artifact to be efficiently extended, changed, customized or configured for use in a particular context (Svahnberg, Gurp and Bosch 2005). So, the developer can specify the particularities of a system corresponding to the specific expectations of a client, and then obtain a concrete variant.

Variability is the main difference between SPL and other Software engineering approaches. So, managing variability in an efficient way is a primary challenge of SPL. Existing work on software variation management can be generally split into two categories:

- The variability in space is the existence of an artifact in different shapes at the same time (Pohl, Bockle and Linden 2005). In others words, at the same time two products may contain variants which have different implementation versions.
- The variability in time is the existence of different versions of an artifact that are valid at different times (Pohl, Bockle and Linden 2005). Variability in time is used to describe change of the artefact versions over time, and their variability dependencies (Elsner et al. 2010). For a product line this means adding, removing, or changing features or their dependencies. Variability in time is used also to describe also binding time of variability. Indeed, there always is a certain amount of variability that cannot be anticipated, therefore, it must be delayed to a later point in the development (compilation, load, runtime, etc.).

2.3 Separation of Concerns in SPL

Product line engineering aims at identifying and exploiting commonalities and variability within a family of software systems. This means that developers must take into account a large number of products, and a large number of stakeholders for each product. So, an increasing number of concerns must be managed by developers.

In order to overcome this difficulty, SPL approaches have proposed different ways to separate concerns (Hubaux, Tun, and Heymans 2013), using several criteria such as:

Functional and non-functional aspects, like in aspect oriented programming (Kiczales et al. 1997), several approaches in SPL separate between functional features that represent the core assets and non functional features that ensure quality of service, adaptation to the execution environment, etc (Noorian, Bagheri and Du 2012; Soltani at al. 2012; Siegmund et al. 2011).

Some approach separate models according to the main characteristics of concerns contained in these models. For example depending on types of variability (Svahnberg, Gurp and Bosch 2005), management constraints (Grunbacher et al. 2009), etc.

The process of configuration allows a separation of concerns because a feature model is specialized in a staged way (necessary features for a stage are selected and other features are removed), according to features binding time (Lee 2013), stakeholders requirements (Czarnecki, Helsen and Eisenecker 2005), a defined workflow of configuration (Hubaux et al. 2010.)

Although the need for separation of concerns in SPL is recognized, there is no consensus about the best criteria to use, and the efficient way to implement this separation. The issue addressed in this paper is how to separate concerns in a too large

SPL in such way that later composition become easy.

3 TOWARDS DYNAMIC MODULARIZATION APPROACH

Our work aims at proposing a new modularization approach for large scale SPL, which we confound in this work with feature models. Based on the challenges discussed in the previous section, we are convinced that using these large models in their initial state is not feasible. That's why; we propose to decompose our models in order to simplify their management. In what follows, we present the concept of modularization and then we detail each proposed dimension, and explain adopted techniques.

3.1 Modularization

Modularization is a well known software technique. It allows developers to decompose a large system into manageable subsystems, which can be developed and checked in isolation (Apel et al. 2013; Ostermann et al. 2011). The main idea behind modular reasoning is information hiding. Indeed, a module is composed of an internal and external part. The internal part hides implementation details of the module, and is not accessible for the rest of modules. The external part is called an interface and describes a contract with the rest of the world (Kästner, Apel and Ostermann 2011).

Over the last decades much research has provided modularity mechanisms. Herbsleb and Grinter (1999) propose a Hierarchical decomposition of models. This means that the system is divided into several hierarchical levels of abstraction. New approaches propose to design patterns of decomposition (Wojcik et al. 2006). So, after identifying quality requirements developer choose the adequate patterns depending on the system architecture. Based on these approaches, Penzenstadler (2010) propose a well defined catalogue of decomposition criteria.

On the basis of presented approaches of modularity as well as the analysis of large-scale systems, we designed our own vision of modularity. In fact, the purpose of modularity in this work is to facilitate the models composition and ensure flexibility and tolerance to change. To achieve that, we opted for the decomposition depending on two dimensions: The first dimension allow a structural decomposition of the feature model. The second dimension is decomposition according to features binding time, which offer o dynamic views of the model.

3.2 First Dimension

The purpose of this first decomposition is to obtain modules that contain significant information about a coherent sub-topic. In other words, the concepts within a module must be semantically related to each other, and weakly related to the outside of the module. Thus, the dependence between the different concepts of the model is the key element to consider in this decomposition.

Our goal in what follows is to detect the features contained in a feature model FM, and are strongly linked together, so that they may constitute separate modules m_i. In order to achieve this goal, we make use of the 'island' algorithm (Batagelj 2003). This algorithm gets the most important sub-graphs contained in a complex network. Such an algorithm, both general and efficient has been used in several scientific fields such as genetics (Whitley, Rana, & Heckendorn 1998), as it was taken up in the computer field for the modularization of large networks such as ontologies (Stuckenschmidt, Schlicht 2009).

In the present work, we were inspired by the work of Stuckenschmidt and Schlicht (2009). This work deals with the decomposition of ontologies according to the structure of its classes. In our case, we try to introduce specific characteristics of feature models by performing some transformations before applying the island algorithm. In the following, we detail the steps of the decomposition according to our first dimension.

3.2.1 Step 1

We consider a feature model FM, composed of several features $C = \{c_1, c_2, \dots, c_n\}$. We consider the set $D = \{d_1, d_2, \dots, d_m\}$ of dependencies between features. Since we think in terms of dependencies between features, the first action would be to list all the existing dependencies. For this, we will transform our FM into a graph whose nodes are the features. In what concern relations of this graph, we will resume the explicit dependencies contained in the FM, as we will explicit the implicit dependencies.

Explicit dependencies: In a FM, we find the following dependencies:

- The directed relationships "parents → children" between the features in spite of their types (mandatory, optional, or-group, and xor-group);
- The constraints between features are also considered as directed relationships from c_i to c_j if « c_i implies c_j » or « c_i excludes c_j». Henceforth, instead of textual expression of constraints, we will represent them graphically in the feature model as dotted arches in order to reason about the graphical model only. Fig. 1 shows the graphical representation of constraints.

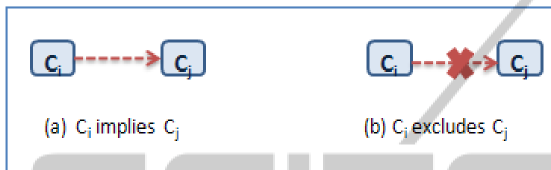

Figure 1: Graphical representation of constraints.

Implicit dependencies: In a feature model, the features are related to their parent with vertical and hierarchical relations or related by constraints with horizontal relationships. However, the features of a xor-goup are linked by implicit mutual exclusion relations between them. In order to explicit this relation, we propose to transform xor-goup in complete graph as shown in Fig. 2.

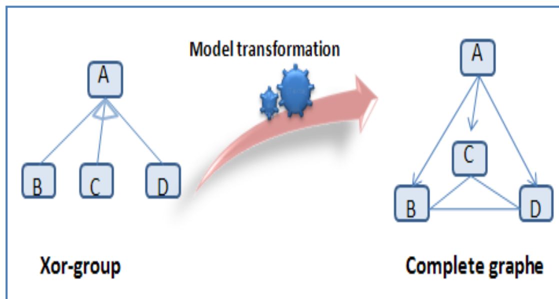

Figure 2: Transformation of xor-group to complete graph.

3.2.2 Step 2

The second step is to assign weights to dependencies of the feature model. This weight reflects the strength of the dependency between two features. Indeed, the strength of the dependency between a feature c_i and a feature c_j is proportional to the total number of dependency strengths of relations with other features of the model. Thus, the weight P (c_i,c_j) is calculated by dividing the sum of the weights of all relations between c_i and c_j by the sum of the weights of all relations that c_i has to other features.

$$P(c_i, c_j) = \frac{p_{ij} + p_{ji}}{\sum_k p_{ik} + p_{ki}} \qquad (1)$$

Where:
p_{ij}: is the weight of the oriented relationship between c_i and c_j. We assume that this value is equal to 1.

3.2.3 Step 3

At this level, we have a graph that contains all the dependencies between features, in addition to the weight of each dependency. So, we can apply the island algorithm to this graph in order to determine the modules.

We consider a model FM transformed to graph G = {C, D, P}. A set of features I is considered an island in G if and only if it induces a connected sub-graph and the lines inside the island I are stronger related among them than with the neighboring features. In other words, there is a maximum spanning tree that contains all the features of I such that:

$$\min_{c_i, c_j \in C, c_j \notin I} P(c_i, c_j) < \min_{c_i, c_j \in I} P(c_i, c_j) \qquad (2)$$

As a result of applying island algorithm, we may obtain very large modules that represent the same difficulties of the initial model. In this case, the island algorithm can be applied iteratively and at several modules to refine the result. We may also obtain isolated features that don't belong to any module. In this case, these features must be assigned to the island of the neighboring features they have the strongest relation to.

3.3 Second Dimension

In this work, our ultimate aim is to propose a modularization approach that deals with special issues of SPL. That's why we propose a decomposition approach that attach due importance to variability concept especially to « Binding-time ». In fact, developer can introduce variability at several binding-times: either at a high abstraction level like design, and compilation, we name that an "early binding", or at a low abstraction level like load or runtime, and we name that a "late binding".

In order to introduce this crucial aspect of variability in our contribution, we propose to decompose the resulted modules of the first decomposition to sub-modules depending on the binding times of their features. Indeed, the features of a given module that are mandatory or those varying but whose inclusion in the resulting system

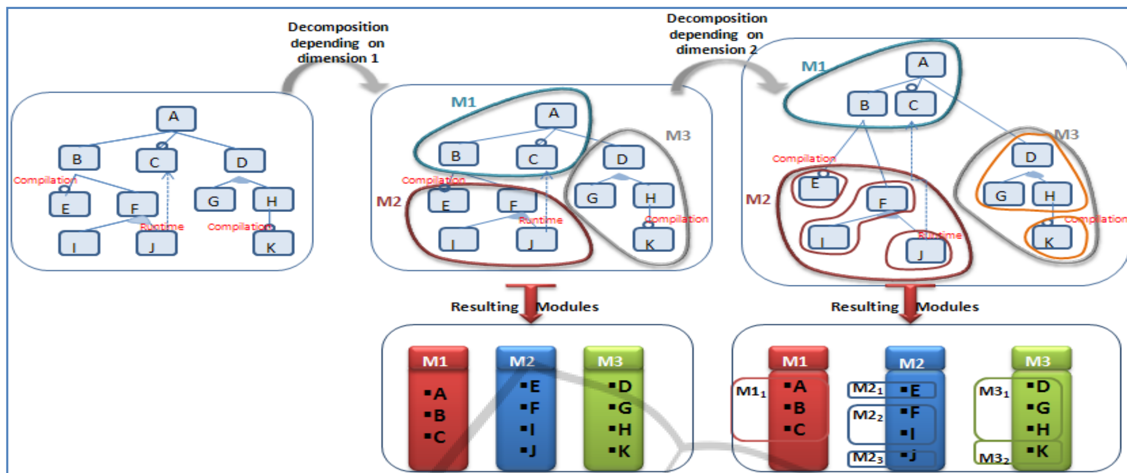

Figure 3: Example of feature model modularization.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

is decided at modeling level must belong to the same sub-module. Then, features are assembled in sub-modules depending to their binding-times (compilation or runtime). Fig. 3 illustrates the modularization following the two dimensions described above. In this example, a feature module is composed to three modules M_1, M_2 and M_3 according to the first dimension. Then, these modules are composed to sub-modules. Each sub module corresponds to a binding-time: design, compilation and execution.

4 CONCLUSIONS

The work presented in this paper provides the conceptual foundations of composition in large scale systems. We recognize that this approach handles the operation of composition at a high level of abstraction. So, additional techniques that deal specially with the details of implementation must be combined with this approach. For example, we propose to use the superimposition of code as a complementary approach (Apel and Lengauer 2008; Apel et al. 2009).

The basic aim of this approach is to analyse the different aspects of variability and try to manage it in a large scale SPL. To achieve that, we propose to decompose a large feature model into manageable modules depending to the features type. Each module is defined by an interface that exposes necessary information to communicate with other modules. Then we propose to define interfaces at several level of abstraction, in order to allow a dynamic binding so as to allow insertion of features changes that arise during the SPL life cycle. Then, we propose to compose these modules in order to obtain composite interface of the resulted FM.

The main advantage of this approach is to allow agility through a proactive process. In fact, stakeholders can express new requirements or constraints. So, changes can be inserted through module interfaces even after compilation. We must notice that additional mechanism must be inserted in order to coordinate between changes at interface level and changes at implementation level using a complementary approach of composition that implements these changes. Interfaces can also be used as a support to explicit traceability of varying features. Hence, traceability information could be used to analyze the design change impact when evolving SPL.

REFERENCES

Apel, S, Batory, D, Kästner, C & Saake, G 2013, *'Feature-oriented software product lines: Concepts and implementation'*, Springer Science & Business Media, Berlin.

Apel, S, Janda, F, Trujillo, S & Kästner, C 2009, 'Model superimposition in software product lines', in the *Proceedings of the International Conference on Model Transformation*, pp. 4-19.

Apel, S & Lengauer, C 2008, 'Superimposition: A language-independent approach to software composition', in *the Proceedings of the International. Symposium of Software Composition*, pp. 20–35.

Batagelj, V 2003, 'Analysis of large networks – islands', presented *at Dagstuhl seminar 03361: Algorithmic Aspects of Large and Complex Networks*.

Benavides, D, Segura, S & Ruiz-Cortés, A 2010, 'Automated analysis of feature models 20 years later: a literature review', *Information Systems*, vol. 35, no.

6, pp. 615-363.

Czarnecki, K, Helsen, S & Eisenecker, WU 2005. 'Staged configuration through specialization and multi-level configuration of feature models', in *Software Process: Improvement and Practice,* vol. 10, no. 2, pp 143–169.

Elsner, C, Botterweck, G, Lohmann, D & Schröder-Preikschat,W 2010, 'Variability in time - product line variability and evolution revisited', in *the proceedings of the 4th International workshop on Modelling Variability of Software-intensive Systems*, Essen, Germany, pp.131-137.

Grunbacher, P, Rabiser, R, Dhungana, D & Lehofer, M 2009, 'Structuring the product line modeling space: Strategies and examples', in *the 3rd International Workshop on Variability Modelling of Software-Intensive Systems*, Seville, pp. 77–82.

Herbsleb, JD & Grinter, RE 1999, 'Splitting the organization and integrating the code: Conway's law revisited', in *the Proceedings of the 21st international conference on Software engineering*, Los Angeles, pp. 85-95.

Hubaux, A, Heymans, P, Schobbens, PY & Deridder, D 2010. 'Towards multi-view feature-based configuration', in *16th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'10).* Springer-Verlag.

Hubaux, A, Tun, TT & Heymans, P 2013, 'Separation of concerns in feature diagram languages: a systematic survey', *ACM Computing Surveys*, vol. 45, no. 4, article no. 51.

Kästner, C, Apel, S & Ostermann, K 2011, 'The road to feature modularity?', in *the Proceedings of the 15th International Software Product Line Conference*, New York, pp. 21-26.

Kiczales, G, Lamping, J, Mendhekar, A, Maeda, C, Lopes, CV, Loingtier, JM & Irwin, J 1997, 'Aspect Oriented Programming' , in *ECOOP 97*, pp 220-242.

Lamb, LC, Jirapanthong, W & Zisman, A 2011, 'Formalizing traceability relations for product lines', *in ICSE Workshop on Traceability in Emerging Forms of Software Engineering(TEFSE),* pp 42–45.

Lee, K 2013. 'Variability and Aspect Orientation'. *Systems and Software Variability Management*, Springer, Berlin, pp 293-300.

Maier, MW 1998, 'Architecting principles for systems-of-systems', *Systems Engineering*, vol. 1, no. 4, pp. 267-284.

Noorian, M, Bagheri, E & Du, W 2012, 'Non-functional Properties in Software Product Lines: A Taxonomy for Classification', in *SEKE*, pp. 663-667.

Ostermann, K, Giarrusso, PG, Kästner, C & Rendel, T 2011, 'Revisiting information hiding: Reflections on classical and non classical modularity', in *the Proceedings of the 25th European Conference on Object-Oriented Programming*, Lancaster, pp. 155-178.

Parra, C 2011, 'Towards dynamic software product lines: unifying design and runtime adaptations'. Europe Laboratory, phD thesis, University of Lille.

Penzenstadler, B 2010, 'DeSyRe: decomposition of systems and their requirements: transition from system to subsystem using a criteria catalogue and systematic requirements refinement', phD thesis, Technical University of Munich.

Pohl, K, Bockle, G & Linden. FJ 2005, *Software product line engineering: foundations, principles and techniques*, Springer, New York.

Rosenmüller, M 2011, 'Towards flexible feature composition: Static and dynamic binding in software product lines', phD thesis, University of Magdeburg.

Siegmund, N, Rosenmuller, M, Kastner, C, Giarrusso, PG, Apel, S & Kolesnikov, SS 2011, 'Scalable prediction of non-functional properties in software product lines', in *proceedings of the 15th Software Product Line Conference*, Munich, pp.160–169.

Soltani, S, Asadi, M, Gašević, D, Hatala, M & Bagheri, E 2012, 'Automated planning for feature model configuration based on functional and non-functional requirements', in *Proceedings of the 16th International Software Product Line Conference*, Salvador, Brazil.

Stuckenschmidt, H, Schlicht, A (2009), 'Structure-Based Partitioning of Large Ontologies. Modular Ontologies'. *Lecture Notes in Computer Science*, Springer, vol. 5445.

Svahnberg, M, Gurp, JV & Bosch, J 2005, 'A taxonomy of variability realization techniques: Research articles', *Software Practice and Experience*, vol. 35, no. 8, pp.705–754.

Urli, S, Blay-Fornarino, M, Collet, P & Mosser, S 2012, 'Using composite feature models to support agile software product line evolution', in *Proceedings of the 6th International Workshop on Models and Evolution,* New York, pp. 21-26.

Whitley, D, Rana, S & Heckendorn, RB (1998), 'The Island Model Genetic Algorithm: On Separability, Population Size and Convergence', *in Journal of Computing and Information Technology,* vol. 7, pp. 33-47.

Wojcik, R, Bachmann, F, Bass, L, Clements, P, Merson, P, Nord, R & Wood, B 2006, 'Attribute-driven design (ADD)', *Technical Report* CMU/SEI-2006-TR-023, Carnegie Mellon University.