International Arab Conference on Information Technology (ACIT'2016)

Managing reuse across MPLs through Partial Derivation

Guendouz Amina
CS Department, Saad Dahlab University
Blida, Algeria
guendouz.amina@yahoo.fr

Bennouar Djamal
LIMPAF Lab, Bouira University
Bouira, Algeria
djamal.bennouar@univ-bouira.dz

Abstract: *Software Product Lines (SPLs) provide systematic reuse only within a particular field. While, in some fields, a single SPL is no longer sufficient to fulfill their requirements due to the large variability amount they include. A set of separated but still interdependent SPLs is then built to handle this issue, commonly known as Multiple Product lines (MPLs). However, reuse between those SPLs must be managed in order to preserve common information among them. In this paper, we propose an approach to systematize reuse across multiple SPLs. Our approach relies on partial derivation and integration of interdependent SPLs at early development stages, avoiding thus inter-SPLs reuse challenges encountered during derivation step.*

Keywords: *MPLs, partial derivation, SPLs integration, Feature models.*

1. Introduction

Software Product Line (SPL) approach has known an increasing success recently in several software fields, owing to the various advantages it brings to software engineering. SPL approach systematizes reuse within a particular field by predicting potential reusable components, developing them in a way that allow them to be easily adapted to several context, and making them available to be reused by final applications development processes. This results in faster applications production with lower cost and effort of development.

However in some domains, adopting a single SPL is no longer sufficient to satisfy the domain needs. Especially in the case of complex and broad fields such as: e-Government. The complexity of those domains and the large variability amount they include resulted in a set of separated but still interdependent SPLs. These SPLs set is commonly known as a Multiple Product Lines (MPL). Distinguishing between SPLs within the same field leads us to lose reuse information between them. MPLs need, then, to manage reuse across the several SPLs they include in order to get larger scale reuse and produce compatible applications more likely to interact easily (such as the case of e-Government domain, where e-Government applications need to interoperate in context of business processes to provide complex services).

In this paper, we propose a new approach to allow integrating SPLs of an MPL at early development stages, avoiding thus inter-SPLs reuse challenges encountered during final applications derivation step. Our proposition relies on partial derivation concept, which is a set of techniques allowing preparing reused SPLs to be systematically integrated with reusing SPLs. Consequently, reuse information between MPL SPLs is preserved and better managed.

The rest of this paper is structured as follow: Section 2 introduces all of SPLs, MPLs and feature model concepts, and motivates our work. Section 3 and 4 defines partial derivation and models integration concepts and illustrate them for feature modeling case. Section 5 reports on a case study illustrating our proposition. Section 6 comments on related work and compare it to ours, while Section 7 summarizes the paper and outlines future work.

2. Background

2.1. Multiple Software Product Lines

A SPL is “A set of software-intensive systems sharing a common, managed set of features that satisfy

the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [13]. SPL approach aims to systematize reuse throughout all software development process: from requirements engineering to the final code and test plans. The purpose is to reduce time and cost of production and to increase software quality by reusing elements (core assets) which have been already tested and secured. These objectives can be realized by putting in common development artefacts such as requirement documents, design diagrams, architectures, codes (reusable components), procedures of test and maintenance, etc.

Therefore, SPL engineering relies on a fundamental distinction between development for reuse and development with reuse [9] [18]. Domain engineering or "development for reuse" consists in developing core assets through the domain analysis, domain design and domain implementation processes. The main outputs of this process are: identification of SPL members (scoping), and extraction of similarity and variability between them. Application engineering or "development with reuse" consists in developing the final products, using core assets and specific requirements expressed by customers. This process is similar to the traditional development process; however, each step is facilitated by reusing outputs from the first process.

Instead of the significant benefits they bring to software engineering, single SPLs are no longer sufficient in some environments due to the emerging of reuse across several interdependent SPLs what is known as MPLs. An MPL is defined as a set of several self-contained but still interdependent product lines that together represent a large-scale or ultra-large-scale system [6]. The emergence of MPLs has given rise to new challenges that need advanced capabilities to support MPLs development. Namely: support for structuring product line models, support for dependencies between product lines and support for distributed product derivation [6]. In this paper we tackle a part of those issues by proposing an approach to facilitate the integration of SPLs within an MPL. We prevent distributed product derivation challenges by preparing the various interdependent SPLs to be integrated in early development stages. Avoiding thus the problems encountered when integrating final applications derived from reused SPLs in reusing SPLs.

2.2. Feature Modeling

One of the most common concepts of SPL engineering is feature model. Feature modeling was first introduced in the Feature-Oriented Domain Analysis (FODA) method by Kang in 1990 [10]. Afterward, it has been widely adopted by SPL community and a number of extensions have been proposed. A feature model is a description of the commonalities and differences between members of a product line. It is generally described by a hierarchy of system features or what is called feature tree, such as a feature is a certain characteristic that each SPL instance may or may not include. It provides an easy-to-understand way to specify and distinguish members of a SPL. It is commonly used to capture functionalities and help delineate commonality and variability in a domain. Thus we adopt feature modeling to illustrate our approach. The feature modeling notation use is expressed as follow:

A feature could be related to its sub-features by four types of relationships: mandatory, optional, single feature with cardinality and choice. All of those types are expressed using cardinality <n, m> as: m>=n. Cardinality means the occurrences number for a single feature and choices number for a feature group such as:

- Mandatory feature is a single feature with cardinality: <1, 1>.
- Optional feature is a single feature with cardinality: <0, 1>.
- A feature that can be included in the system several times is called: single feature with cardinality. The number of its occurrences is denoted by a cardinality interval <n, m> such as: n>=0 and m>1.
- Choice relationship is used to represent a feature that is related to a variable set of sub-features restricted by cardinality. It includes all of and, or, xor variability types as shown in the figure 1. We use this notation to avoid cluttering the diagram with several notations, for us cardinality is sufficient to represent all kinds of choices.

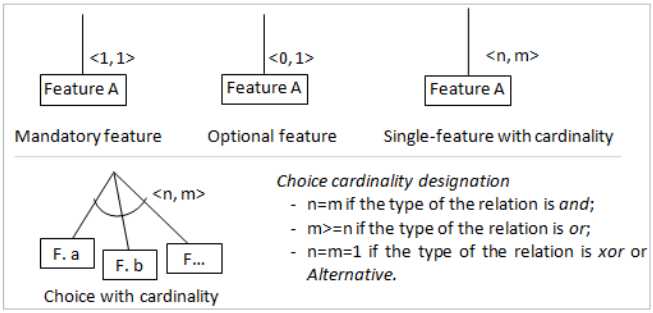

Figure 1. Feature model notation.

2.2. Motivation

Current MPLs tends to integrate components derived from some SPLs in other reusing SPLs within the same MPL at derivation time. This integration way

(integration at derivation time) results in several problems. Reused components that are already developed using particular modeling and implementation techniques must be adapted to fit the new application requirements. What became more complicated if the used modeling and implementation languages are different. Moreover, this procedure needs to be repeated for each context included in the reusing SPL. It means at each derivation time of an application that needs reusing a component from another SPL, this component must be adapted to fit the new application requirements. If the target application needs reusing several components from several other SPLs, the whole adaptation and integration process must be repeated accordingly, what result in delaying derivation and produce several adaptation and integration challenges.

According to Rosenmüller et al. integration at derivation time of SPLs instances may imply handling dependencies between those instances at domain level [17] what is not supported by current modeling languages. Another problem generated by this integration method is cluttering domain models with instances detail instead of keeping distinction between the various abstraction levels.

Reviewing current integration methods, we observe that they are restricted to deal with issues concerning the integration of one instance in one SPL. That ignore the case of integrating a SPL (eventually a SPLs set) with several SPLs. That multiply reuse contexts what is the case in MPLs engineering. Derived components to be reused in a set of separated SPLs must be adapted, not only, according to the various SPLs fields, but also to several contexts included in each field.

Those techniques of composition, adaptation, and configuration lead developers to lose systematic reuse between SPLs of an MPL. While, systematizing reuse was the crucial aim when introducing product line approach in software engineering. So how to keep systematic reuse even among separated SPLs within a broad MPL?

3. Partial Derivation

In order to reply to the before mentioned issues, we propose a new approach aiming to facilitate the integration of separated SPLs within an MPL and maintain the systematic reuse principle as well. Instead of integrating SPLs instances, we suggest integrating SPLs themselves after been partially derived according to the reusing SPL requirements. Partial derivation consist of the resolving (better saying: modifying) a variation points (VP) set (in some cases all VPs) included in the reused SPL to fit requirement of the reusing SPL. The results of this process are not final applications (or components) ready to be used, but rather a set of partially derived artefacts that could be derived completely as part of the reusing SPLs.

Partial derivation is comparable to the specialization concept that was introduced by Czarnecki et al. [3] [4]. Czarnecki defines specialization as the transformation process that takes a feature diagram and yields another feature diagram, such that the set of configurations denoted by the latter diagram is a true subset of the configurations denoted by the former diagram. Successive specialization processes result in a final configuration, this method is called staged configuration [3].

Specialization defers from partial derivation in two crucial ways. On the one side, the purpose of introducing specialization is to allow handling applications derivation through several configuration stages what is needed in the case of software supply chains. Final applications derivation step is then decomposed into several specialization stages each one is performed by a particular actor. Whereas partial derivation aims to integrate SPLs during domain engineering phase. On the other side, specialization is defined to be applied particularly on feature models what is clear from its definition. While partial derivation is applied to all artefacts extracted from the reused SPL domain engineering (including: requirements models, architecture and final code…)

Unlike specialization, the resulting model from a partial derivation procedure does not describe necessarily a sub-set of the systems described by the original model. In some cases, the partially derived model is extended by adding new functionalities or VPs to fulfill the particular needs of the reusing field. This is due to the fact that the resulted model is integrated with the entire reusing SPL (with all its covered contexts) not a particular final application. So new functionalities could be introduced in order to fulfill requirements of the various contexts. We can then distinguish between two partial derivation categories: restriction and expansion techniques. The partial derivation of a model can include transformations from both categories.

- Restricting a model means altering the model in a way that restricts the choices set covered by the resulted model. The set of transformations that could be done in this category are: - to reduce a cardinality interval of a choice VP - to change a VP type from mandatory to optional - to restrict an attribute by assigning a value – to omit a VP or a feature (eventually a component).
- Expanding a model means to modify this model in such a way that expand the choices covered by the resulted model. The set of transformations included in this category are: - to extend a cardinality

interval of a choice VP – to change a VP type from optional to mandatory – to add a VP or a feature (eventually a component).

In order to illustrate the different techniques, we choose to apply them on feature model, since it is a well known and broadly used modeling language (see Section 2.2).

3.1. Restriction Technique

3.1.1. Reducing a Cardinality Interval of a Choice VP

Reducing a cardinality interval means excluding an element or a set of elements from the choices described by the VP. In feature models, we may restrict a choice with cardinality <n, m> to <n′, m′> where n′>= n and m′<=m. A special case of this operation is when we restrict the choice interval to <0, 0>. This implies removing the set of choices with their descendants, and removing the parent feature of the choices set if it is not related to other sub-features. In the case of single feature with cardinality, cardinality interval could be derived in the same way as choice with cardinality. Special cases occur when getting <0, 1> or <1, 1> intervals. If we obtain <0, 1> interval, the feature type is then optional. If we obtain <1, 1> interval, the feature type is then mandatory. In the case of <0, 0> interval, the feature is completely removed from the diagram.

3.1.2. Changing a Variability Type from Mandatory to Optional

A variability type may change if needed by the reusing SPL. A mandatory feature may become optional, that decrease the probability to be included in final reusing applications. Dependencies constraints may be changed or added accordingly.

3.1.3. Restricting an Attribute by Assigning a Value

A restriction way may be to assign a value to an attribute. In feature model case, this could be done by initializing an uninitialized attribute, or changing the initial value.

3.1.4. Removing a Functionality

A restriction operation may be done by removing a functionality. In feature models, we may omit a feature with all its descendants. Omitting a grouped feature or an occurrence from single feature with cardinality cases correspond to what is described in reducing a cardinality interval technique section.

3.2. Expansion Techniques

3.2.1. Extending a Cardinality Interval of a Choice VP

Extending a cardinality interval is done by adding an element or a set of elements to the group described by the VP. In feature modeling case, a choice with cardinality may be extended if features are added to the feature group as long as consistency is preserved. A choice with cardinality <n, m> may be extended to <n′, m′> where n′<= n and m′>=m. A single feature with cardinality may be extended by adding new occurrences. Even a feature without cardinality may become with cardinality if new occurrences are added. If it is related to sub-features, they are eventually multiplied. Constraints related to the altered features must be reviewed and modified if needed to keep models consistence.

3.2.2. Changing a Variability Type from Optional to Mandatory

A functionality may become obligatory for particular reusing contexts. For feature models, an optional feature may be changed to mandatory type if needed, this imply the obligatory existence of this feature in final reusing applications.

3.2.3. Adding a Functionality

In the case of specific new requirements by the reusing SPL, the reused SPL could be extended by new functionalities. A feature model could be then, expanded by adding a new feature (with its descendants), or a feature group. Related constraints are changed or added accordingly. The case of adding a feature to a group, or multiplying a feature occurrences are described by extending a cardinality interval technique.

4. SPLs Integration

Integration means the merging of the various partially-derived SPLs core assets to be reused in a particular field with the SPL of this field. Integration could be performed simultaneously with the reusing SPL domain engineering, such as each partially-derived artifact, from one or several SPLs, is merged with its correspondent reusing artifact at development time. Nevertheless, it could be integrated subsequently since the core assets base is available to reuse.

Reusing SPL must plan to reuse in order to decrease the risk of encountering integration challenges. Moreover, used languages are recommended to be compatible in order to ease integration step, yet they could be adapted using a unified language, for example, before integration. Several works have

studied the merging of SPLs models: Morin et al. [12] [11] present an approach to safely integrating aspects models with variability into existing models. Abele et al. [1] provides an overview on a variability management tool called CVM framework. Among other capabilities, the tool allows composing feature diagrams from several related SPLs. Alférez et al. [2] propose the Variability Modeling Language for Requirements (VML4RE), a multi-view composition language for SPL requirements. VML4RE language supports the composition of elements defined in separate and heterogeneous requirement models using a set of operators. Dhungana et al. [5] present an approach to facilitate variability models integration. They provides a unified perspective to users configuring products in MPL environments, by making the internal technical aspects of using variability models for configuration transparent to the stakeholders performing the configuration.

Reused components concern generally particular features from the reusing SPL. Those components are represented by black boxes that will be replaced by partially-derived SPLs thereafter. The black boxes could be annotated in the feature model by a new features type that we call «Aspect feature». Aspect features denote some SPL parts that are extracted from other SPLs within the same MPL. During integration step, aspect features are replaced by the corresponding partially-derived feature models from the reused SPLs. If the reused SPL feature model is large and merging it with partially-derived feature models increases its complexity, an alternative integration method could be adapted. Aspect features can be considered as referring features, that refer to the partially-derived feature models as there reference models.

Finally, models consistency must be preserved after integration. Therefore, the resulted artifacts' integrity is checked after each integration step. This is done by checking reusing SPLs constraints with regard to the merged models.

5. Case Study

To illustrate the proposed approach we will apply it in an ambitious domain that is e-Government. E-Government is a broad field including several sub-fields: e-Administration, e-Justice, e-Voting, e-Meeting, e-Health, e-Education, etc. Over time, applications in each subfield have taken the form a SPL owing to the commonalities they include. Nevertheless, reuse across those SPLs need to be handled in order to not lose the reuse information between them. Even if e-Government subfields SPLs are separated they still contain commonalities such as: user management functionalities since these SPLs are intended basically to the same user kind which is citizens, security functionalities that are recommended for handling personal data and official documents, other additional functionalities that are widely recommended by e-Government applications as e-Meeting, e-Voting, e-Poll, e-Statistics and so on.

For this case we consider a functionality that is required by several e-Government subfields SPLs which is: e-Meeting. After showing the specification of e-Meeting SPL using feature diagram, we present its partial-derivation to be reused in two separated e-Government subfields SPLs: e-Education and e-Administration.

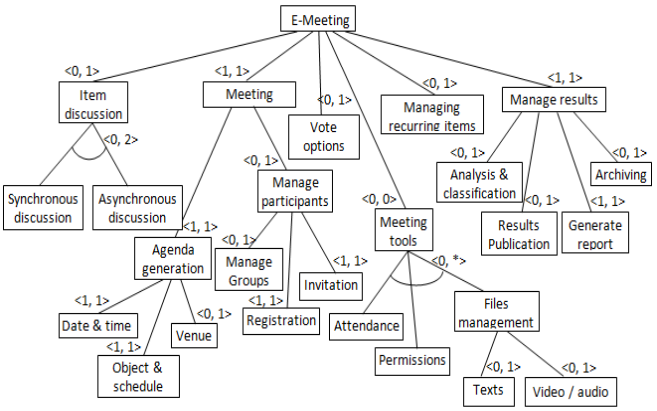

Figure 2.Feature diagram of e-Meeting SPL.

The e-Meeting components can be required by various e-Government SPLs, for instance: online courses meetings, Scientific Council meetings, meetings of elected members of an APC, APW or APN, business leader meetings, etc. Figure 2 depicts the feature model of an e-Meeting SPL that have been designed to produce e-Meeting components which could be integrated in several e-Government subfields SPLs including e-Education and e-Administration SPLs. A similar diagram has been presented in a previous work [7] that we have done for a different purpose.

E-Education SPL provides teachers and students by online application to exchange documents, plan courses, perform online tests, and other functionalities. For this case study, e-Education meetings designate the electronic meeting joining students to their teachers online. The partial derivation of e-Meeting SPL to be reused by e-Education SPL results in the diagram reported in figure 3. E-Education Meetings usually do not need to prepare an item to be discussed; meetings are organized according to an educational planning. Therefore, the feature "Item discussion" has been omitted in addition to all its descendents. Other features like "vote options" and "management of recurring items" that are mostly needed in decision making meetings are also omitted. Meeting reports in this case are not constantly needed; thus, "Generate report" feature type is changed from mandatory to

optional. "Invitation" feature becomes also optional since students do not need to be invited each time they have an online course.

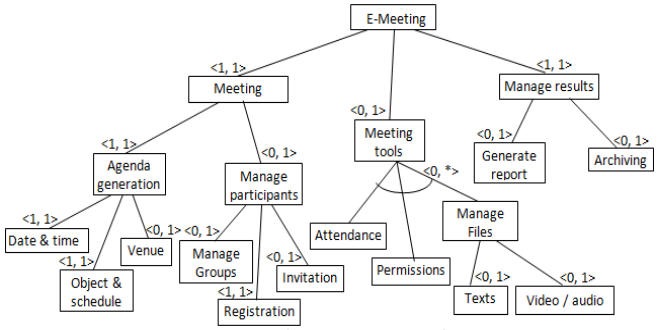

Figure 3.Partially-derived feature diagram for e-Education SPL.

E-Administration SPL is responsible for developing e-applications intended to APCs, APWs, APNs. E-Administration applications provide citizens by several services including: civil state documents download, official events declaration, passports, driving licenses and national identity cards… Administrations need e-Meeting functionality for the smooth running of the elected members meetings. For the case of e-Administration partial derivation there will be only few changes therefore we do not repeat the diagram. All of "vote options" and "publication of results" features take the kind of mandatory features in e-Administration SPL. We keep "Item discussion" and "management of recurring items" features because APCs, APWs and APNs usually discuss items before organizing meetings to take decisions by voting (for example in the case of decision making meetings).

An example on the integration of partially-derived e-Meeting feature diagram with e-Education SPL is depicted in figure 4. E-Meeting feature that has been already denoted as an «Aspect feature» at e-Education SPL domain engineering is replaced by the partially-derived e-Meeting feature diagram.

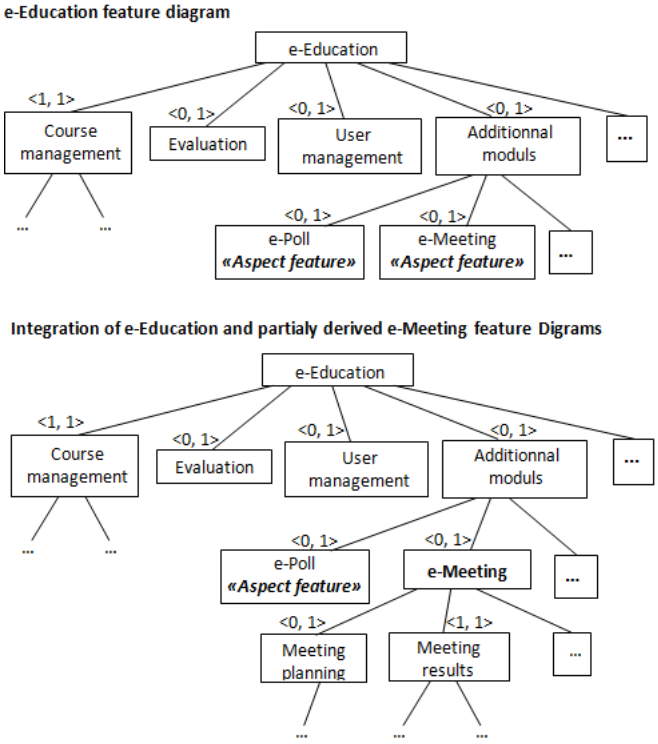

Figure 4. Feature diagrams integration.

If e-Government subfields SPLs are specialized into sub-SPLs such as: e-primary SPL, e-secondary SPL, e-University SPL which represent specialized SPLs of e-Education SPL, and e-APC SPL, e-Daïra SPL, e-Wilaya SPL that are specialized SPLs of e-Administration SPL. Partially derived SPLs could be also more specialized to fit those sub-SPLs. Specialized SPLs allows better variability management since it will be more restricted according to the subfields constraints. Therefore this step is recommended in MPL engineering to decrease complexity and thus reaching better quality.

5. Related Work

Several approaches have been proposed to handle reuse across SPLs within an MPL, but as stated before, they consider generally the integration of SPLs instances.

Marko et al [17] have altered the MPLs structuring problem. They propose to extend the feature model with explicit modeling of SPL instances. The matter is to allow configuring an SPL using multiple instances of another SPL. SPLs and SPLs instances are modeled using classes and objects, such as SPL instantiation correspond to class instantiation. In another work, Marko et al [16] added the notion of composition model aiming to automate the configuration of MPLs. A composition model integrates multiple SPLs by describing for each SPL which instances of other SPLs

it uses. The main difference between this work and ours is that we avoid to delay the composition of SPLs until getting application level, where it is likely to have incompatible instances derived from separate SPLs. Partial derivation and integration of partially derived SPLs with currently developing SPL avoid this problem later, and the resulted composed SPL will be derived as an ordinary SPL.

Reimar et al [15] [14] introduce multi-level interfaces to guaranty the correct collaboration between multiple SPLs. They distinguish between four interfaces: variability-model interfaces, syntactical product-line interfaces, behavioral product-line interfaces, and non-functional property interfaces. Those interfaces aim to detach the direct dependency between SPLs and to enable modular analysis of MPLs correctness. They are defined as follow:

- Variability-model interface: is a specialization of the reused SPL's variability model.
- Syntactical interface: represent a view of an SPL's reusable code artefacts without implementation detail.
- Behavioral interface: is an agreement on the behavior of different methods.
- Non-functional interface: represent non-functional properties of an SPL that other SPLs use.

Apparently, the introduced interfaces represent views on what could be reused from an SPL within an MPL. They are defined collaboration means between SPLs of an MPL. authors do not mention how the interfaces are realized or how one SPL is reused by another one.

Herman and Tim [8] propose to combine feature model with context variability model to model MPLs supporting several dimensions in context space. They use stage configuration to generate specialized feature models. The Context Variability model captures the commonality and variability of the context. The context is the environment in which a product resides. The Context Variability Model is combined with a conventional feature model to create an MPL-Feature model. However, this model needs more work at each reused SPL engineering process and must be maintained over all the development process. Yet, constraints handling the partial derivation are extracted from the reusing SPLs at reuse time only if reuse is required. What avoid more work in precedent steps, and allows considering current requirements of the field.

6. Conclusions and Future Work

In this paper we have presented an approach consisting of two crucial steps to integrate SPLs within an MPL. At first step, SPLs that are requested to be reused by other SPLs are partially-derived according to the reusing SPLs requirements. In the second step, the partially-derived SPLs are merged with the reusing SPLs. The aim is to systematize reuse across MPL SPLs, prevent late integration challenges by integrating SPLs at early development stages, and thus gain in terms of time, cost, and effort of development. We have illustrated the approach by presenting the partial derivation and integration of e-Meeting SPL in two separated e-Government SPLs (e-APC and e-Education).

In the future, we plan to apply this approach to cover other modeling languages at different abstraction levels, particularly: architecture models. Automating the partial derivation and SPLs integration is also one of our major goals. Finally, we intend to test our approach in other application fields in order to improve it.

References

[1] Abele, A., Papadopoulos, Y., Servat, D., Törngren, M., Weber, M., "The CVM framework- A prototype tool for compositional variability management", VaMoS 10, p. 101-105 (2010)

[2] Alférez, M., Santos, J., Moreira, A., Garcia, A., Kulesza, U., Araújo, J., Amaral. V. "Multi-view composition language for software product line requirements", In Software Language Engineering, pp. 103-122. Springer Berlin Heidelberg (2010)

[3] CZARNECKI, K., HELSEN, S., et EISENECKER, U., "Staged configuration using feature models", In: Software Product Lines. Springer Berlin Heidelberg, p. 266-283 (2004)

[4] CZARNECKI, K., HELSEN, S., EISENECKER, U., "Staged configuration through specialization and multilevel configuration of feature models", Software Process: Improvement and Practice, vol. 10, no 2, p. 143-169 (2005)

[5] Dhungana, D., Seichter, D., Botterweck, G., Rabiser, R., Grunbacher, P., Benavides, D., Galindo, J. A., "Configuration of multi product lines by bridging heterogeneous variability modeling approaches", In Software Product Line Conference (SPLC), 2011 15th International, pp. 120-129. IEEE (2011)

[6] Gerald, H., Grünbacher, P., Rabiser, R., "A systematic review and an expert survey on

capabilities supporting multi product lines." Information and Software Technology 54, no. 8, pp 828-852 (2012)

[7] Guendouz, A., Bennouar, D., "Component-based specification of Software Product Line architecture", In proceeding of ICAASE, pp. 100-107 (2014)

[8] Hartmann, H., Tim T. "Using feature diagrams with context variability to model multiple product lines for software supply chains", In Software Product Line Conference, 2008. SPLC'08. 12th International, pp. 12-21. IEEE (2008)

[9] Klaus,P., Böckle,G., and van der Linden,F., "Software Product Line engineering: foundations, principles, and techniques", Springer (2005)

[10] Kyo, C. K., Sholom, G. C., James, A. H., William E. N., A. Spencer, P. "Feature-Oriented Domain Analysis (FODA) feasibility study", technical report CMU/SEI (1990)

[11] Morin, B., Klein, J., Barais, O. Jézéquel, J. M., "A generic weaver for supporting product lines", In Proceedings of the 13th international workshop on Early Aspects, pp. 11-18. ACM (2008)

[12] Morin, B., Vanwormhoudt, G., Lahire, P., Gaignard, A., Barais, O., Jézéquel, J. M., "Managing variability complexity in aspect-oriented modeling", In Model Driven Engineering Languages and Systems, pp. 797-812. Springer Berlin Heidelberg (2008)

[13] Northrop, L.M., Clements, C.C., "A framework for Software Product Line practice", Version 5.0 (online), http://www.sei.cmu.edu/

[14] Reimar, S., "Using Multi-Level Interfaces to Improve Analyses of Multi Product Lines", technical report, Otto-von-Guericke University Magdeburg, Germany (2014)

[15] Reimar, S., Siegmund, N., Thüm, T., "Towards modular analysis of multi product lines", In Proceedings of the 17th International Software Product Line Conference co-located workshops, pp. 96-99. ACM (2013)

[16] Rosenmüller, M., Siegmund, N., "Automating the configuration of Multi Software Product Lines", VaMoS 10, p. 123-130 (2010)

[17] Rosenmüller, M., Siegmund, N. Kästner, C., Syed, S. R., "Modeling dependent software product lines", In Proceedings of the GPCE Workshop on Modularization, Composition and Generative Techniques for Product Line Engineering (McGPLE), pp. 13-18 (2008)

[18] Van der Linden, F., Schmid, K., Rommes, E. "Software Product Lines in action", The Best Industrial Practice in Product Line Engineering, Springer, (2007)