A Multimodel Approach for Specifying the Requirements Variability on Software Product Lines

David Blanes dblanes@dsic.upv.es
ISSI Research Group, Department of Information Systems and Computation
Universitat Politècnica de València
Camino de Vera, s/n, 46022, Valencia, Spain

Javier González-Huerta jagonzalez@dsic.upv.es
ISSI Research Group, Department of Information Systems and Computation
Universitat Politècnica de València
Camino de Vera, s/n, 46022, Valencia, Spain

Emilio Insfran *einsfran@dsic.upv.es*
ISSI Research Group, Department of Information Systems and Computation
Universitat Politècnica de València
Camino de Vera, s/n, 46022, Valencia, Spain

Abstract

Requirements engineering is a key activity on any software development project. In Software Product Line development, this activity is even more important since requirements may encompass commonality and variability. Furthermore, a requirement specification usually is composed by more than one model. In this context it is necessary to specify the requirements variability in the different models of a Software Product Line requirements specification. In order to solve this issue, this paper proposes a multimodel approach for specifying the requirements for the products of a Software Product Line. This multimodel is used in a model-driven development process in order to obtain the requirements of a single product by applying model transformations. This solution increases the flexibility allowing developers to add more views depending on the domain and to obtain the product requirements by using model-transformations, whereas improves the productivity. The feasibility of the approach is illustrated through a running example.
Keywords: Requirements Engineering, Software Product Line Engineering, Model-Driven Development, Domain Engineering, Application Engineering.

1. Introduction

Software Product Lines (SPL) development is a paradigm to produce a family of software products based on an intensive reuse strategy. The SPL development is usually divided in two main processes: the domain engineering and the application engineering. At the *domain engineering* a set of core assets is built. At the *application engineering* these core assets are used to produce the specific products.

The Requirements Engineering (RE) at SPL development needs to cope with common, variable, and product-specific requirements not only for a single product but for the whole set of products. On these products, many requirements will be common across the product line, and many of them will be optional. Requirements common across the SPL are written with variation points that can be filled in or exercised to create product-specific requirements [4]. In this context, the problem of dealing with these variation points across the requirements specification is a topic of interest in the SPL development.

The Model-Driven Development (MDD) is a paradigm where the models are the first class citizens of the software development process. MDD is based on an intensive reuse

strategy by using model transformation among models at different abstraction levels. The MDD approach helps to establish in the traceability among different models. Furthermore, the transformations among models help to increase productivity. In order to create software product lines by applying MDD, product variability has to be represented on every modelling level and preserved under model refinement [18].

In a previous work we presented Feature-Driven Requirements Engineering approach (FeDRE) [14], which provides support to the requirements specification of SPLs. In this paper, we integrate FeDRE in a multimodel infrastructure in order to establish the relationships among the functional requirements viewpoint with the variability viewpoint, and to obtain the requirements of the product under development by applying model transformations. The contributions of this paper are twofold: i) The extension of FeDRE in order to provide support to the specification of requirements variability in multiple models at the domain engineering stage by applying model-driven engineering principles; ii) the integration the Common Variability Language (CVL) [15] in the multimodel in order to obtain the product requirements from the domain requirements by applying model transformations at the application engineering stage.

This paper is structured as follows: Section 2 discusses several proposals which deal with the specification of requirements variability on SPL. Section 3 presents the multimodel for representing the requirements variability of a family of products. Section 4 presents the proposed process to define and specify the requirements of a SPL which permits the automatic derivation of the product requirements. Section 4 shows an example to illustrate the feasibility of the approach. Finally on Section 5 draws the conclusions and further work.

2. Related Work

Several approaches deal with the requirements specification, and its variability in SPL development. On the one hand, some proposals are focused on how to enrich requirement models to deal with the variability (e.g. [6]). On the other hand, some proposals use technics to compose the requirements with the variability (e.g. [1], [3], [10]) with different purposes.

The approach presented in [6] proposes the creation of relationships between elements in the activity diagrams and the corresponding features by using annotations and presence conditions, which are used to indicate the existence of model elements depending of the feature configuration. In FeDRE we express the variability in a different model in order to avoid polluting the functional requirements specification with variability information.

Another solution is to use a compositional strategy (e.g., Modeling Language for Requirements (VML4RE)). VML4RE [1] presents two main contributions. First, the VML4RE language is a Domain Specific Language (DSL) able to compose requirements with feature models; and second, a requirements process that uses: i) a feature model to perform the variability identification; ii) use cases and activity diagrams to describe the SPL domain requirements; and a iii) a VML4RE model to relate the feature model with the requirement models. FeDRE uses the CVL language for describing the requirements variability and for obtaining the product requirements, instead a specific DSL for each problem as VML4RE does. Another compositional strategy is applied on the Modeling Scenario Variability as Crosscutting Mechanisms (MSVCM) [3] approach, which is focused on obtaining the product requirements by means of the following artifacts: use case model, feature model, product configuration, and configuration knowledge. These artifacts are taken as input in the weaving process which composes the product specific use case model. MSVCM describe the scenarios by means of textual templates, whereas in FeDRE we specify the use cases by means of activity diagrams. Another compositional proposal is applied on Goknil et al. [10] where a multimodel approach is defined for reasoning about requirements and their relations. The process that guides the requirements specification is not covered like in FeDRE. Moreover, this latter approach does not provide automation for obtaining the product requirements.

As conclusion, we analyzed several proposals that cover the specification of requirements. Annotative proposals (e.g. Model Templates [6]) has the disadvantage of polluting the functional specifications with variability information. In FeDRE we consider to

have two models of variability: external and internal; which are related among them in the multimodel. Finally, other proposals that used compositional techniques (e.g. [1], [3], [10]), do not provide guidelines to obtain the requirements from the feature model; oppositely, in FeDRE we bring a solution to guide how to specify the requirements and its variability, taking as input the feature model, and we are able to obtain the product requirements automatically.

3. A Multimodel for Modelling Requirements Variability SPLs

In this Section we provide an overview of the multimodel for representing the requirements of the products that comprise the SPL. A multimodel is a set of interrelated models that represents the different viewpoints of a particular system and the relationships among elements on these viewpoints [2]. In a previous work the multimodel was used to guide the software architect in the derivation and improvement of product architectures in a model driven software product line development process [12]. This multimodel included initially four viewpoints: the variability, functional, quality, and transformation viewpoints. In this paper we adapt this multimodel in order to deal with the traceability among the external variability, represented in the variability viewpoint, and the requirements viewpoint with the aim to obtaining the product requirements through model transformations. The goal of this strategy is to provide a framework to model the requirements variability in a domain-independent way. The multimodel for this purpose is composed of two viewpoints:

- **The variability viewpoint** expresses the commonalities and variability within the product line. This viewpoint expresses the external variability of the SPL. Its main element is the *feature*, which is a user-visible aspect or characteristic of a system [5]. The variability view of the multimodel has been defined using a variant [11] of the cardinality-based feature model [7], defined specifically for its application in a model-driven product line development context.
- **The requirements viewpoint** expresses the variability at the requirements level which realizes the external variability. We express this variability by using the CVL language [9] which allows expressing the variability of a Domain Specific Language (DSL). CVL allows establishing variability points over a base model defined in a DSL. This variability is defined by means of Meta Object Facility (MOF) compliant metamodels. The CVL approach works with two models: the *CVL model*, which defines variation points in the base model; and the *CVL resolution model* which resolves the variability for generating resolved models. The main CVL entity to represent the variability abstractions is the Variability Specification (Vspec).

We define a relationship among the features from the variability viewpoint and the VSpec elements. This solution allows, at the application engineering, obtaining a CVL resolution model that it will be used to obtain the application requirements through model transformations.

4. A Model-driven Approach to Obtain the Product Requirements

In this Section we present a process with different activities in order to specify the requirements of the family of products and their variability in order to obtain the product requirements. The process covers the two main SPL processes: Domain Engineering, and Application Engineering. Fig. 1 shows the main activities of the FeDRE approach.

4.1. Domain Engineering

The Domain Engineering activities take as input three main artifacts from the Scoping activity [14] First, the *Feature Model* [13] represents the SPL external variability by using: features, SPL variations, and constraints among the features in the SPL. Second, the *Feature Specification* is a template that captures the detailed information of the features. Finally, the *Product Map* is an optional document where the set of relationships among features and products are described.

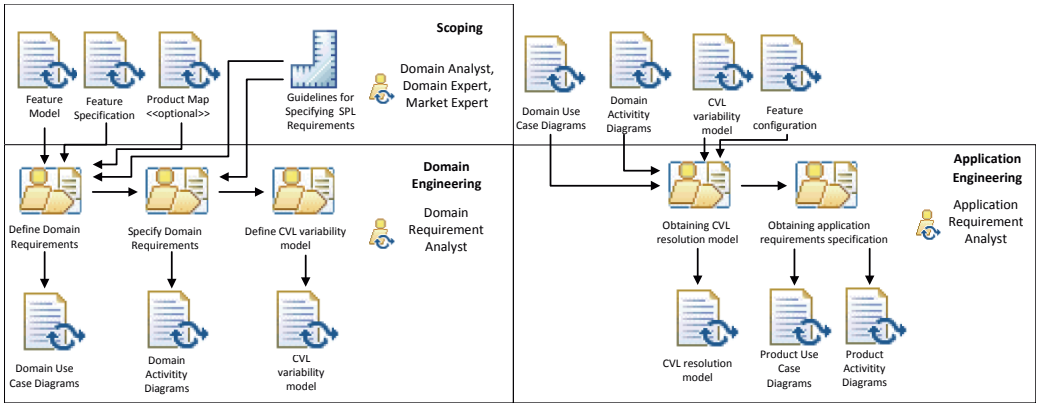

Fig. 1. FeDRE process: a) Domain Engineering; b) Application Engineering

FeDRE was defined using and extending the PLUSS approach [8] considering the feature model as the main artifact for specifying SPL requirements. The aim of the approach is to perform the requirements specification in a systematic way by using the features identified in the scoping activities through the use of a set of guidelines[1]. The multimodel is used to keep the traceability among the SPL external variability and the requirements variability (the internal variability), which is presented through a CVL model in the requirements viewpoint. In FeDRE we support two requirements variability mechanisms: at the use case level by using "extends" relations, or within a*ctivity diagrams* by using *Decision Nodes during the use case specification*. Three main activities have been defined for the Domain Engineering: *Define Domain Requirements, Specify Domain Requirements*, and *Define CVL variability model*. In this Section we describe each one of these activities. The *Domain Requirements Definition* activity is usually performed in an iterative and incremental manner. Sets of selected features from the Feature Model can therefore be defined as units of increments for the specification. The guidelines assist the Domain Requirements Analyst when identifying the requirements using as input the Feature Model. Firstly, the Domain Requirements Analyst must decide which of these features would be specified by means use cases (*which*). After deciding which features need to be specified as use cases, the *Domain Requirements Analyst* had to identify which use cases should be associated to each feature (*what*). Since some use cases with similar behavior may be identified for different features that have the same parent, the *Domain Requirements Analyst* should decide where to relocate the specification for this use case (so as to avoid the redundant specification of similar behavior). When this happens, the use case is specified only once at the parent feature level. After the identification, use cases are modelled. Finally, the Domain Requirements Analyst specifies the variability at the use case diagrams by using "extends" relationships. The output of this activity is the use case diagram.

Specify Domain Requirements

The next step is to specify these domain requirements by using UML activity diagrams, which represents workflows of activities with choice support, iteration and concurrency. We use activity diagrams extended with *Activity Partitions*. The partitions divide the nodes and edges to constrain and show a view of the contained nodes [17]. Every *Actor* has an Activity Partition that represents the action flow. The System will have always an *Activity Partition* since we model the interactions among different actors and the System. Other element is the *Decision Node*, which represent controls nodes that choose between outgoing flows. The Decision Nodes allow expressing multiple alternatives, which should be used to represent variability.

[1] The FeDRE guidelines are available at: http://bit.ly/fedreguidelines

Define CVL Variability Model

The CVL Language allows us to represent the requirement's variability independently from the DSL used to model the requirements. The variation points in CVL refer to base objects with the aim to define the base model modifications precisely. In this approach the *base model* is composed of: the use case, and the activity diagrams. This solution allows setting up variation points without modifying the DSL models.

4.2. Application Engineering

On the application engineering we obtain the requirements for a specific product taking as input the multimodel and the feature configuration. Two main activities are considered based on model transformations: *obtaining the CVL resolution model, and obtaining the application requirements specification*. The first activity takes as input the multimodel and the feature configuration and produces a CVL resolution model. This CVL resolution model is taken as input in the second activity together with the CVL variability model and the base model (the use case diagrams and the activity diagrams) to obtain the product requirements through CVL transformations.

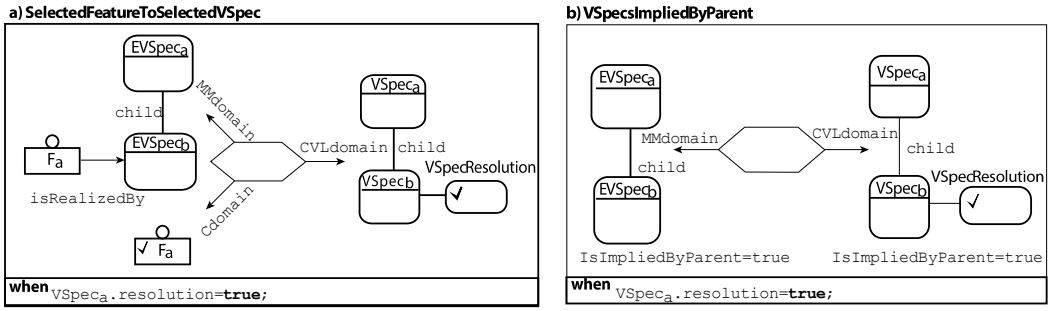

Fig. 2. Transformations defined in QVT-Relations graphical notation [16]

Obtaining the CVL Resolution Model

The input artifacts in this activity are the: the multimodel, and the feature configuration. A *feature configuration* is a set of features, which satisfies the constraints defined on the feature model. The CVL resolution model is generated through a QVT Relations [16] model transformation that take as input: the multimodel (which includes the feature model, the CVL variability model, and their relationships) and the feature configuration. The transformation process comprises a set of QVT relationships between two domains: the *source domain* that is the metamodel which supports the configuration model; and the *target domain*, that is the metamodel which supports the CVL modeling language. Three transformations are defined: *RootFeatureToRootVSpec*, which creates the CVL element container when exists a root feature on the configuration model; *SelectedFeatureToSelectedVspec* (see Fig. 2.a), which positively resolves the set of VSpecs which realizes the features selected in the configuration model; and *VSpecsImpliedByParent* (see Fig. 2.b), which positively resolves a VSpec if its field *IsImpliedByParent* is defined as true and it is child of another VSpec that has been previously positively resolves. After applying the model transformations we obtain the CVL resolution model which contains the set of VSpecs which had been positively resolved.

Obtaining the Application Requirements Specification

The CVL resolution model resolves the variability in a base model. In this activity we use the traceability between the base model (use case diagram, and activity diagrams) and the CVL specification. The CVL transformation uses the resolution model and the CVL specification for obtaining the requirement specification models for the product.

5. Example: the Savi mobile Emergency Notification Family of Products

In this Section we illustrate the approach with an example. The example is based on a real application in the SAVi[2] SPL, which is a set of mobile applications that allow sending emergency notifications. Additionally, some products can send notification by using the Twitter and Facebook social networks.

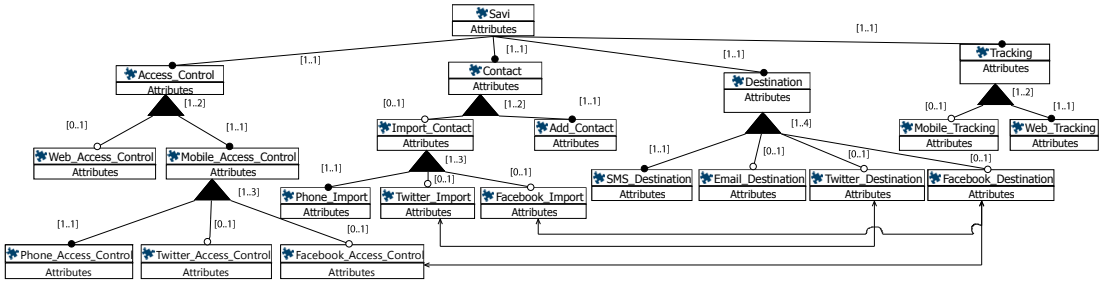

Fig. 3. Savi Feature Model

5.1. Domain Engineering

This activity specifies the SPL requirements for domain engineering by applying the FeDRE guidelines. Fig. 3 shows the Feature Diagram for the Savi product line. In the identification of the requirements we select a group of features in the increment unit for the current interaction. The *Domain Requirements Analyst* has to decide which of these features would be specified by use cases. According to the first task of the guidelines the specification starts with the feature *Access Control* and its children, because they are a group of features that share functionality. The FeDRE guidelines allow us to use two variability mechanisms, in this example we use both mechanisms in order to illustrate the approach. For the *Create User* use case (see Fig. 4.a) two possible extensions are defined by means the "extend" relationships. The extend use case are: *Link Twitter*, and *Link Facebook*. Each of them is associated with an actor that represents the external systems (Twitter, and Facebook).

The second type of variability is at the activity diagram level (Fig. 5.a). The activity diagram has one normal flow of activity, which it is the mobile login. Two alternatives may be included: *Twitter Login* and *Facebook Login* depending on the configuration.

5.2. Application Engineering

In this activity we take the configuration as input with the feature model and CVL variability model from the Domain Engineering. A feature configuration is shown on Fig. 4.b. For this product the customer desires an application with a *Mobile Access Control* and additionally *Twitter Access Control* support. After apply the transformation we have the CVL resolution model. This CVL resolution model contains which VSpecs are going to be resolved positive or negative. Finally, taken as input the CVL resolution model, the CVL transformation is executed in order to generate the product requirements.

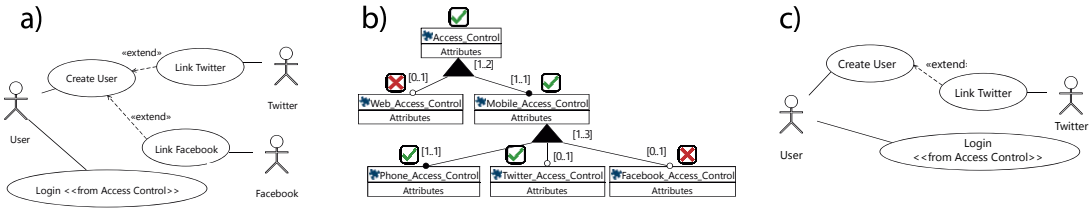

Fig. 4. a) Domain Use Case diagram, b) Configuration, c) Product Use Case Diagram

[2] Available at: http://goo.gl/1Q49O

The result produces a use case diagram (see Fig. 4.c) associated to the features *Mobile Access Control, Phone Access Control, and Facebook Access Control* where the base use case is extended with the use case *Link Twitter*. The use case *Link Facebook* and its relationship "extends" to the base use case is removed since the feature *Facebook Access Control* is not selected. The use case *Login* will be extended by using the activity diagram variability mechanisms. Fig. 5.b shows the generated activity diagram for this product. Two possible alternatives are modeled. In the main action flow, the user sends the data to the System and the System shows a response with a confirmation of an error message. In the alternative flow, the user request to login with the Twitter data. In this case the System sends the data to the *Twitter System* (represented in a different partition). The response of the *Twitter System* is sent to the *System* again and the appropriate message is shown.

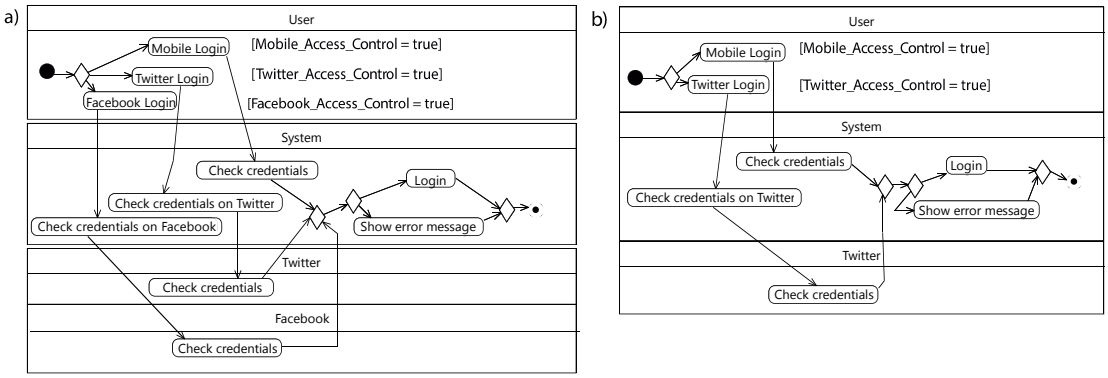

Fig. 5. a) Domain Activity Diagram, b) Product Activity Diagram for the Login Use Case

6. Conclusions and Future Work

This paper presents an approach which extends FeDRE for specifying the requirements of a family of products with a model-driven solution. The approach supports the two main processes of a SPL: i) *Domain Engineering*, where the requirements and its variability are modeled by using a multimodel that allows establishing the relationships among the requirements viewpoint and the variability viewpoint; ii) *Application Engineering*, where a strategy based on model transformations is used to obtain the product requirements. This solution provides support to the explicit representation of the traceability among the external variability (at SPL level) and the internal variability (at use cases and activity diagrams level) without polluting the requirement models with variability information. Moreover, this approach allows obtaining the product requirements by using model transformations improving the flexibility, reusability. An example based on a real software product line which manages an emergency notification system is used to illustrate the feasibility of the approach.

Future work will include experiments for assessing the feasibility of the approach applied in different domains. Concretely, it has been planned a case study based on a Cloud Computing SPL. We also pretend to extend the multimodel by considering the non-functional requirements as a new view. Furthermore, we are investigating how to assure the consistency among the different viewpoints that conform the multimodel.

Acknowledgements

This research was supported by the FPU program (AP2009-4635) from the Spanish Ministry of Education and Science, and the ValI+D program (ACIF/2011/235) Generalitat Valenciana.

References

1. Alférez, M., Santos, J.P., Moreira, A., Garcia, A., Kulesza, U., Araújo, J., Amaral, V.: Multi-view Composition Language for Software Product Line Requirements. In:

Proceedings of the Second international conference on Software Language Engineering (SLE). pp. 103–122. Springer, Denver, USA (2009)
2. Barkmeyer, E.J., Feeney, A.B., Denno, P., Flater, D.W., Libes, D.E., Steves, M.P., Wallace, E.K.: Concepts for Automating Systems Integration. National Institute of Standards and Technology, Bureau Drive, Gaithersburg, USA (2003)
3. Bonifácio, R., Borba, P.: Modeling scenario variability as crosscutting mechanisms. In: Proceedings of the 8th ACM international conference on Aspect-oriented software development (AOSD). pp. 125–136. ACM, Charlottesville, Virginia, USA (2009)
4. Clements, P., Northrop, L.: A Framework for Software Product Line Practice, Version 5.0, http://www.sei.cmu.edu/productlines/framework.html, Accessed: April 01, 2014, (2007)
5. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley Professional, Boston, MA, USA (2001)
6. Czarnecki, K., Antkiewicz, M.: Mapping features to models: A template approach based on superimposed variants. In: 4th International Conference on Generative Programming and Component Engineering (GPCE). pp. 423–437. Springer, Tallin, Estonia (2005)
7. Czarnecki, K., Kim, C.H.P.: Cardinality-based feature modeling and constraints: A progress report. In: International Workshop on Software Factories, held at Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA). pp. 16–20. ACM Press, San Diego, California, USA (2005)
8. Eriksson, Magnuss, Börstler, Jürgen, Borg, K.: The PLUSS approach - domain modeling with features, use cases and use case realizations. In: Proceedings on 9th International Conference on Software Product Lines (SPLC). pp. 33–44. Lecture Notes in Computer Science, Rennes, France (2005)
9. Fleurey, F., Haugen, Ø., Møller-Pedersen, B.: A generic language and tool for variability modeling. SINTEF, Oslo, Norway (2009)
10. Goknil, A., Kurtev, I., Millo, J.-V.: A Metamodeling Approach for Reasoning on Multiple Requirements Models. In: Proceedings of the 17th IEEE International Enterprise Distributed Object Computing Conference (EDOC). pp. 159–166. Vancouver, BC, Canada (2013)
11. Gómez, A., Ramos, I.: Cardinality-based feature modeling and model-driven engineering: Fitting them together. In: Int. Workshop on Variability Modeling of Software intensive Systems (VAMOS). pp. 61–68. Universität Duisburg-Essen, Linz, Austria (2010)
12. González-Huerta, J., Insfrán, E., Abrahão, S.: Defining and Validating a Multimodel Approach for Product Architecture Derivation and Improvement. In: 16th International Conference on Model-Driven Engineering Languajes and Systems (MODELS). pp. 388–404. (2013)
13. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. T, Software Engineering Institute, Carnegie Melon University (1990)
14. Oliveira, R.P. de, Insfran, E., Abrahão, S., Gonzalez-Huerta, J., Blanes, D., Cohen, S., Almeida, E.S. de: A Feature-Driven Requirements Engineering Approach for Software Product Lines. In: Proceedings of the VII Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS). pp. 1–10. IEEE, Brasília, Brazil (2013)
15. OMG: Common Variability Language (CVL) Revised Submission, http://www.omgwiki.org/variability/doku.php, Accessed: April 01, 2014, (2012)
16. OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, http://www.omg.org/spec/QVT/1.0/PDF/, Accessed: April 01, 2014, (2008)
17. OMG: UML 2.4.1 Superstructure Specification, http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF, Accessed: April 01, 2014, (2007)
18. Schaefer, I.: Variability Modelling for Model-Driven Development of Software Product Lines. In: Int. Workshop on Variability Modeling of Software intensive Systems (VAMOS). pp. 85–92. Linz, Austria (2010)