Actas de los Talleres de las Jornadas de Ing. del Software y BBDD, Vol. 3, No. 2, 2009

Feature Model to Orthogonal Variability Model Transformations. A First Step

Fabricia Roos-Frantz[1] David Benavides[2], Antonio Ruiz-Cortés[2]

[1] Unijuí, Departmento de Tecnologia - Ijuí, RS, Brasil
`frfrantz@unijui.edu.br`
[2] University of Seville, ETSI Informática. Avda. Reina Mercedes, s/n. Sevilla 41012, Spain
`{benavides, aruiz}@us.es`

Abstract. Feature Model (FM) and Orthogonal Variability Model (OVM) are both modelling approaches employed to represent variability in software product line engineering. The former is the most popular and it is mainly applied to domain engineering. The later is a more recent approach mainly used to document variability in design and realisation artifacts. In the scenario of interest of our research, which focuses on Application Lifecycle Management environment, it would be useful rely on the FM to OVM transformation. To the best of our knowledge, in the literature, there is no proposal for such transformation. In this paper, we propose an algorithm to transform FM into OVM. This algorithm transforms the variable features of a FM into an OVM, thus providing an explicit view of variability of software product line. When working on these transformation, some issues came to light, such as how to preserve semantics. We discuss some of them and suggest a possible solution to transform FM into OVM by extending OVM.

Key words: Sofwtare Product Lines, Transformations, Feature Models, Orthogonal Variability Model

1 Introduction and Motivation

Software Product Line (SPL) Engineering paradigm [14,10] is one of the most recent ways of software reuse. According to Clements et al. an SPL *"is a group of products that share a common, managed set of features"*. Variability models (VMs) are used to document and manage these features, in order to allow the organisation manage and evolve its products. Such models document common and variable parts of the Product Line (PL). The common parts are called commonality, since they are features that form part of all products of the PL, and the variable parts are called variability, since they are part of some of those products. SPL Engineering consists of two main processes: *Domain Engineering* and *Application Engineering* [10]. According to Pohl et al., *"Domain Engineering is the process of SPL engineering in which the commonality and the variability of the product line are defined and realised"*, and *"Application Engineering is the process of SPL engineering in which the applications of the product line are built by reusing domain artifacts and exploiting the product line variability"*.

Feature Model (FM) and Orthogonal Variability Model (OVM) are both modelling approaches employed to represent variability in SPL Engineering. The former is a common approach employed to represent an SPL by means of a hierarchical decomposition of features, which yields a feature tree, comprising commonalities and variabilities. The later is a more recent approach mainly used to document variability in design and realisation artifacts [12,11]. Its main goal is to document variable features of the SPL without take into account the common features.

In 2006, a report by the Forrester consulting company [2] coined a term that has become one of the most relevant topics in the software engineering community: Application Lifecycle Management (ALM). ALM is defined as "*the coordination of development life–cycle activities, including requirements, modeling, development, build, and testing, through: 1) enforcement of processes that span these activities; 2) management of relationships between development artifacts used or produced by these activities; and 3) reporting on progress of the development effort as a whole*". Our current research aims at developing a reference architecture for ALM environments promoting process–quality standards compliance and integrating software engineering tools keeping traceability among artifacts. In this scenario, which focuses on ALM environment, it is necessary to have in place a FM to OVM (FM2OVM) transformation. For instance, it will be of practical importance being able to have interoperability between FM and OVM, proving a tool that can work with a VM in different views. One view of all products in an SPL with their variabilities and commonalities represented with FMs and another view of an SPL, where only the variation points are considered (cf. Fig. 1).

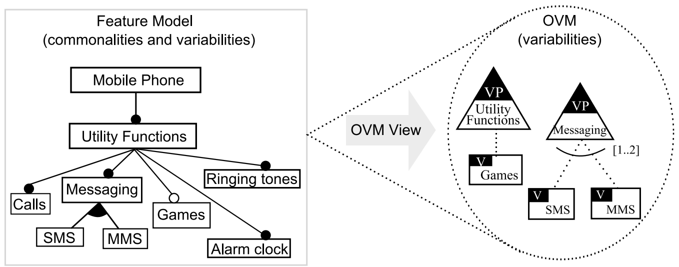

Fig. 1. Different views of variability in an SPL.

In addition, the translation FM2OVM will be helpful in our research work on automated analysis of OVM [8]. In order to achieve this, we intend to use FAMA Framework [16] which is a tool for the automated analysis of VMs. Its main goal is to provide an extensible framework where current research on VM automated analysis might be developed and easily integrated into a final product. FAMA receives as input a model conforms to a FM metamodel and performs several analysis operations on this FM by using different solvers. We aim to use model-to-model transformation in order to generate a target model conforms to an OVM metamodel from a source model conforms to a FM metamodel, and thus to be able to analyse this resulting FM by using FAMA.

Such analysis could be thought as a subset of the automated analysis of FMs so that the analysis of a specific OVM model can yield the same results as the analysis of the equivalent FM.

To the best of our knowledge, in the literature, there is no proposal for FM2OVM transformation. In this paper, we propose an algorithm to carry out such transformation. This algorithm transforms the variable part of a FM into an OVM, thus providing an explicit view of variability of SPL, since the OVM model will represent the variability documented in the FM in a more explicit way. When working on this transformation, some issues came to light, such as how to preserve semantics of decomposition of features in the tree. We discuss some of these issues and suggest a possible solution to FM2OVM transformation by extending OVM.

The remainder of this paper is structured as follows: Section 2, sets the basic terminology around FM and OVM; Section 3, describes our transformation proposal; Section 4, presents some discussions and open issues concerned with FM2OVM transformation. Finally, we draw conclusions and future work in Section 5.

2 Preliminaries

2.1 Feature Models

A FM represents graphically a PL by means of combinations of features. It has been introduced by the software design community to represent in an abstract way the commonalities and variabilities of an SPL. A FM is composed of two main elements: features and relationships between them. Features are structured in a tree or DAG (Directed acyclic graph) where one of these features is the root.

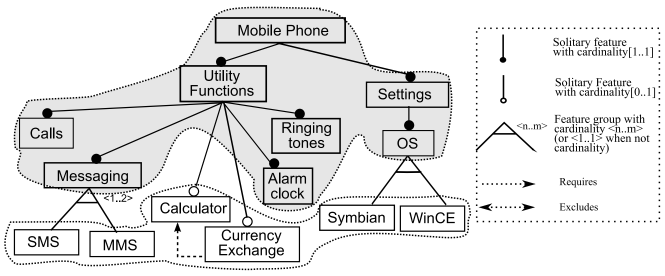

Fig. 2. An example of a mobile phone product line using FM.

Fig. 2 depicts an example of a FM inspired by the mobile phone industry and graphical notation based in Czarnecki's FM [9]. This example defines a PL where every product contains eight *solitary features* with card [1..1] (e.g *Calls* and *Messaging*). Furthermore, the PL has five variable features, which can be selected or left out at will. For example, the *grouped features SMS* and *MMS* with card [1..2] are possible choices of their parent *Messaging*, and one or two of them must be chosen. Each product must have one, and only one, OS (*Symbian* or *WinCE*) due to its card [1..1]. And the *solitary features* with card [0..1], such as *Calculator* can be selected or left out. The constraint *Requires* imposes limits on its possible combinations. It says that when *Currency Exchange* is chosen, *Calculator* must be chosen as well. The areas limited by dashed lines illustrate which features are common and which are variable in this PL. The grey area illustrates the common features of the PL, and the blank area illustrates the variable features.

There are many analysis operations that can be used to check FMs, e.g. check if a model is valid or check if a product is valid for a given model. These and other checks on FMs were reviewed by Benavides et al. in [4]. In our paper we are interested in three operations: calculate the number of products of a FM, retrieve all possible products of a FM, and obtain the set of core features (CF), i.e, features which are part of all products.

By applying those analysis operations on our example in Fig. 2 we obtain 18 possible products. Each product in the Mobile Phone PL consists of core features (CF = {Mobile Phone, Utility Functions, Calls, Messaging, Alarm clock, Ringing tones, Settings, OS}) and variable features, as follows:

P1=	CF ∪ {SMS,Symbian}	P10=	CF ∪ {MMS,WinCE,Calculator}
P2=	CF ∪ {SMS,WinCE}	P11=	CF ∪ {SMS,MMS,Symbian,Calculator}
P3=	CF ∪ {MMS,Symbian}	P12=	CF ∪ {SMS,MMS,WinCE,Calculator}
P4=	CF ∪ {MMS,WinCE}	P13=	CF ∪ {SMS,Symbian,Calculator,Currency Exchange}
P5=	CF ∪ {SMS,MMS,Symbian}	P14=	CF ∪ {SMS,WinCE,Calculator,Currency Exchange}
P6=	CF ∪ {SMS,MMS,WinCE}	P15=	CF ∪ {MMS,Symbian,Calculator,Currency Exchange}
P7=	CF ∪ {SMS,Symbian,Calculator}	P16=	CF ∪ {MMS,WinCE,Calculator,Currency Exchange}
P8=	CF ∪ {SMS,WinCE,Calculator}	P17=	CF ∪ {SMS,MMS,Symbian,Calculator,Currency Exchange}
P9=	CF ∪ {MMS,Symbian,Calculator}	P18=	CF ∪ {SMS,MMS,WinCE,Calculator,Currency Exchange}

2.2 Orthogonal Variability Model

The OVM approach was proposed to document the variability on SPL in an orthogonal way [10]. In this model the first-classes are: *variation points (VP)* and *variants (V)*. A *VP* documents what vary in the SPL and a *variant* documents how a *VP* can vary. All the variabilities in the PL artifacts are documented in the OVM model, but not the commonalities. The common parts would be documented in others PL models, such as requirements, design, etc. Therefore, only variabilities are represented by an OVM model, whilst on FMs commonalities are documented as well.

At this section we use the OVM abstract syntax proposed by Metzger et al. [1] which is similar to that defined in [10] by means of a metamodel. Each VP must be related to at least one variant and each variant must be related to one and only one VP. A mandatory VP must always be bound, i.e, all the products of the PL must have it and its variants must always be chosen. An optional VP does not have to be bound, it may or may not be chosen to a specific product. Always that a VP is bounded, its mandatory variants must be chosen and its optional variants may be chosen, but do not have. In OVM, optional variants may be grouped in *alternative choices*. This group is associated to a cardinality *[min...max]*. Cardinality determines how many variants may be chosen in an alternative choice, at least *min* and at most *max* variants of the group. When the cardinality is *[1...1]*, for default, it is not shown.

In OVM, constraints between nodes are defined graphically. They can be between variants and variants, variants and VPs, or VPs and VPs. These constraints are:*excludes* or *requires*. Fig. 3 depicts an example of an OVM which represents the same PL as the FM of the Fig. 2, and graphical notation based on [10]. In this case, the OVM model documents the variability represented in the FM in Fig. 2.

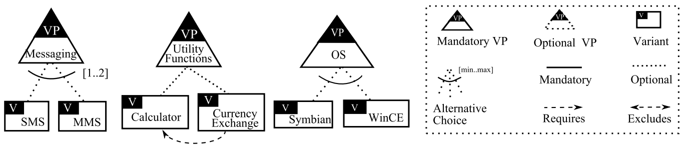

Fig. 3. OVM example: mobile phone product line.

In the same way that in Subsection 2.1 we calculate the number of products of an OVM and its set of products. In our previous work in [8], we described informally some analysis operations on OVM, such as how to obtain all the possible products. Using this operation we obtain 18 products of the OVM in Fig. 3, and they are all the same as in Subsection 2.1, excepting the CF that in this case is {Utility Functions, Messaging, OS}. We can note that the number of products represented by each model is the same, and although the CF is different, both models represent the same variability.

3 Transforming FM to OVM

Our main goal is to obtain the OVM equivalent from a given FM. In order to achieve this, it is necessary that every variability represented in the FM is transformed into a variation point in the OVM. This section describes a possible way to transform FM into OVM. We propose a FM2OVM algorithm in such a way that the variable parts of FM is transformed into OVM. Hence, the target model leaves aside the commonalities and gives an explicit view of the variability represented in the source FM. To our transformation we use the metamodel presented in [6] as the abstract syntax of FM. To comply with space restrictions we do not show the metamodel. As OVM metamodel we propose a metamodel based on the OVM abstract syntax defined by Metzger et al. in [1] (cf. Fig. 4). It was adapted in order to use the same concepts used in the FM metamodel aforementioned, e.g. *Set*, *Binary*, *Grouped* and *Solitary*. The element variation point has two types of relations, *Set* and *Binary*. The former is equivalent to the *alternative choice*, which has at least 2 *grouped variants* and a cardinality. The latter, is equivalent to *mandatory* or *optional*, which has one *solitary variant*. The OVM model may or may not have constraints. Some Object Constraint Language (OCL) statements were included to ensure some properties, as for instance the OCL at *context Set*. It ensures that the attribute *Card.max* is less than or equal to the number of grouped variant in the *set* relation and that *Card.min* is greater than or equals to 0, and *Card.min* is less than or equals to *Card.max*.

Our transformation algorithm has four preconditions: (1) the FM must be syntactically correct, it must satisfy the metamodel FM; (2) the FM must be valid, it represents at least one product [5,4,3]; (3) the FM does not have dead features, i.e. features that do not appear in any product [15,3]; and, (4) the FM has variability, at least one of its features is a variant feature, i.e. features that do not appear in all the products [3].

In addition to the preconditions, there are some postconditions. We consider that the translation is satisfactory if: (1) the target OVM model is syntactically correct according to the metamodel in Fig. 4; and, (2) the products of the source model without core features are the same as the ones of the target model without core features.

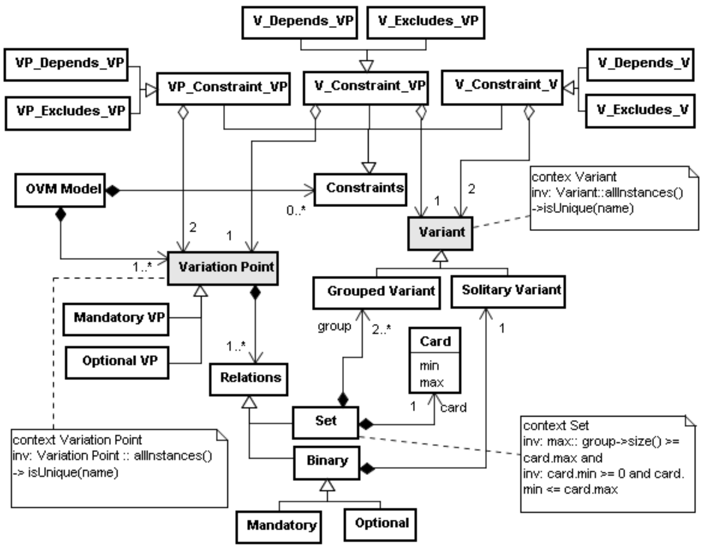

Fig. 4. OVM metamodel based on [1].

The activity diagram of Fig. 5 illustrates how we propose to transform a FM into an OVM. We use the activity diagram as a tool to facilitate the understanding of the transformation algorithm, in fact it represents the main steps of our algorithm. In the following we describe the corresponding activities and the choices which determine the succession of them. The algorithm traverses the tree in preorder and performs the following operations recursively at each node, starting with the root node:

"Select Feature n in FM:" the algorithm visits the node; *"n in core features or n is the root?"* concerns the distinction between features that are common to all products (core features) and those that are not (variabilities). When n is a CF or root it does not become an element in OVM; *"Transform parent(n) in VP:"* when n is not a CF, its parent will be transformed in a VP; *"parent(n) in core feature?"* when transforming the parent(n) in VP, the type of such VP depends of it is a variability or a commonality; *"Set VP type as OPTIONAL VP:"* when parent(n) is a variability, then VP is Setted as OP-

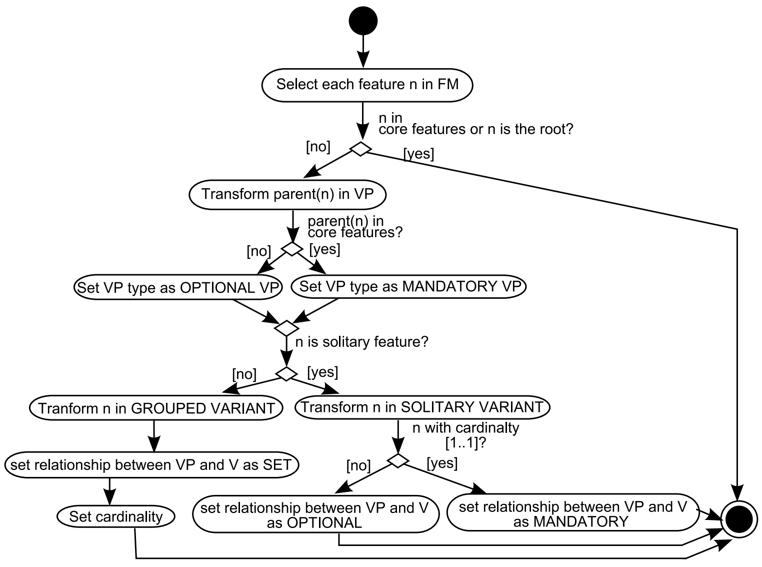
Select each feature n in FM
n in core features or n is the root?
[no]
[yes]
Transform parent(n) in VP
parent(n) in core features?
[no]
[yes]
Set VP type as OPTIONAL VP
Set VP type as MANDATORY VP
n is solitary feature?
[no]
[yes]
Tranform n in GROUPED VARIANT
Transform n in SOLITARY VARIANT
n with cardinalty [1..1]?
set relationship between VP and V as SET
[no]
[yes]
Set cardinality
set relationship between VP and V as OPTIONAL
set relationship between VP and V as MANDATORY

Fig. 5. FM2OVM algorithm.

TIONAL VP; *"Set VP type as MANDATORY VP:"* when parent(n) is a commonality, then VP is setted as MANDATORY VP; *"n is solitary feature?"* the type of n determines the type of variant when transforming n into variant; *"Transform n in GROUPED VARIANT:"* the grouped feature n in FM will be transformed in grouped variant; *"Set relationship between VP and V as SET:"* when a variant is a grouped variant, the relationship with its parent VP is SET; *"Set cardinality:"* when the relationship is SET, it has a cardinality; *"Transform n in SOLITARY VARIANT:"* the solitary feature n in FM will be transformed into solitary variant; *"n with cardinality [1..1]?"* the cardinality of a solitary feature determines if the variant will be optional or mandatory; *"Set relationship between VP and V as OPTIONAL:"* when solitary feature with cardinality [0..1] is transformed into variant, the relationship with its parent VP is OPTIONAL; *"Set relationship between VP and V as MANDATORY:"* when solitary feature with cardinality [1..1] is transformed into variant, the relationship with its parent VP is MANDATORY.

Now we can use the examples in Fig. 2 and 3 to illustrate the transformation. Taking into account that the FM fulfils the preconditions, we are able to translate a FM into an OVM. Then, by applying the algorithm transformation (cf. Fig. 5) from the FM in Fig. 2 we obtain the OVM in Fig. 3. The resulting OVM is syntactically correct and if we omit all the CF of the FM set products and also of the OVM set products, we have two equivalent set of products, namely:

P1=	{SMS,Symbian}	P10=	{MMS,WinCE,Calculator}
P2=	{SMS,WinCE}	P11=	{SMS,MMS,Symbian,Calculator}
P3=	{MMS,Symbian}	P12=	{SMS,MMS,WinCE,Calculator}
P4=	{MMS,WinCE}	P13=	{SMS,Symbian,Calculator,Currency Exchange}
P5=	{SMS,MMS,Symbian}	P14=	{SMS,WinCE,Calculator,Currency Exchange}

P6=	{SMS,MMS,WinCE}	P15=	{MMS,Symbian,Calculator,Currency Exchange}
P7=	{SMS,Symbian,Calculator}	P16=	{MMS,WinCE,Calculator,Currency Exchange}
P8=	{SMS,WinCE,Calculator}	P17=	{SMS,MMS,Symbian,Calculator,Currency Exchange}
P9=	{MMS,Symbian,Calculator}	P18=	{SMS,MMS,WinCE,Calculator,Currency Exchange}

It therefore follows that given a FM, which represent a PL (a set of products with commonalities and variabilities), the translation FM2OVM results an OVM which represent the same PL as the source FM, but representing only the PL variability.

4 Discussions and Open Issues

To translate a FM into an OVM in such a way that the target OVM can express all the variability documented in the source model without losing semantics demands to deal with some issues. In this section we comment some of them which came to light when working on a way to find out a satisfactory transformation.

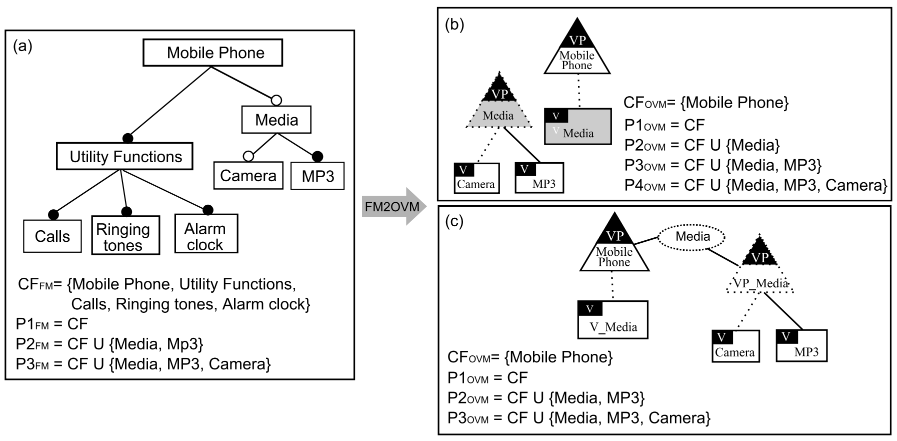

Fig. 6. FM2OVM transformation.(a) semantic problem and (b) OVM extension.

Intermediate features. When working on the transformation, we have found that some features in a FM corresponding at the same time to a VP and to a variant related to a VP (See feature *Media* in Fig. 6 (a)). Media is a possible variation of Mobile Phone and also is a variation point which has two options *MP3* and *Camera*. These features are called intermediate features, i.e they are neither leaves nor root, and besides they are not core features. In this case one feature should become two elements in the OVM (one variant and one VP) (See Fig. 6 (b)).

Semantics of decomposition. Another issue is how to preserve the semantics of decomposition of features? Decomposition of features concerns the relationship between a feature parent and its child. In Fig. 6 (a) the *Mobile Phone* is decomposed in *Media*, and *Media* is decomposed in *MP3*. The decomposition means that if a feature *n* is in the product, its parent must be as well. For instance, if *MP3* is part of the product, its parent *Media* and all its ancestors also must be, thus *Mobile Phone* must be part of the product. OVM, in contrast to FM, does not have a hierarchical structure, then there is no way to preserve semantics without constraints. If we observe the Fig. 6 (b), we have *Media* in P2$_{OVM}$ and do not have *MP3*. When the variant Media is chosen, the variation point Media should be chosen as well, because it represents how the feature *Media* can vary. Thus the *MP3* variant shoul be selected. Then, if in our transformation we simply transform each intermediate feature in a variant and a VP, and each leaf feature in a variant, we do not preserve semantics and the OVM in Fig. 6 (b) will have one product more then the original FM. Therefore, the P2$_{OVM}$ should not be allowed.

How to preserve semantics. One alternative is by means of constraints. For example, if we define a constraint *Includes* from Media variant to Media VP, we ensure that always Media variant is chosen, Media VP also is. But if *Mobile Phone* VP were optional, we will need a constraint from Media VP to Media variant, to ensure that MP3 is not chosen without the variant Media. However, using constraint we solve partially the problem because it can generate confusion and mixing of concepts.

Extending OVM. In order to provide a way to maintain semantic decomposition after the transformation, we propose an extension for OVM, by adding a new operator called "*decomposition dependency*". This operator specifies a relationship between a variant and a VP, and this variant can not be child of such VP. A decomposition dependency between a variant and a VP says that always the variant is chosen the VP must be chosen as well. On the other hand, the VP only can be chosen if the variant is also selected. With the decomposition operator the variant and the variation point works as a unique node. In Fig. 6 (c) we illustrate the use of *decomposition* operator. We use the ellipse as graphical notation of it. See the ellipse *Media*, which is connected with the *variant Media* and the *variation point Media*. Now, the set of products is exactly the same as in FM. In addition, if we consider the OVM abstract syntax defined in [1] the names of variants and VPs must be qualified (e.g. V::Media and VP::Media) in order not to replicate names.

5 Conclusions and Future Work

In our scenario of research it is useful to have the transformation FM2OVM, and up to now to the best of our knowledge, there is no work providing such solution. Such transformation would allow us provide a tool support to work with the interoperability between both languages. Moreover, it would be a step forward in our work on analysis of OVM using FAMA framework. In this paper we proposed an algorithm to transform a FM into an OVM. Particularly, we obtain a view of the variability of a FM by means of an OVM. When defining our algorithm we have found that the main issue to fulfil the postconditions concerns the structural difference between FM and OVM diagrams. To translate a tree into OVM is not a straight task due to it usually has more than two levels. Regardless the technique used to do this transformation, this problem comes forward.

In order to achieve the defined postconditions we propose an extension for OVM, thus enabling to obtain the OVM diagram equivalent to the FM diagram.

Our future work focuses on the implementation of the algorithm by using a model transformation language, such as Query View Transformation (QVT) [13] or Atlas Transformation Language (ATL) [7]. In addition, we intend to validate this algorithm by using a wider set of examples.

Acknowledgement. This work was partially supported by Spanish Government under CICYT project Web-Factories (TIN2006-00472) and project SETI (TIN2009-07366) and by the Andalusian Government under project ISABEL (TIC-2533) and Evangelischer Entwicklungsdienst e.V. (EED).

References

1. Metzger A., Pohl K., Heymans P., Schobbens P., and Saval G. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In *15th IEEE Int'l Requirements Engineering Conference*, pages 243–253, 2007.
2. Schwaber C. The changing face of application life-cycle management. Forrester Research, 2006.
3. Benavides D. *On the automated analysis of software product lines using feature models*. PhD thesis, University of Sevilla, 2007.
4. Benavides D., Ruiz-Cortés A., Trinidad P., and Segura S. A survey on the automated analyses of feature models. In *JISBD*, pages 367–376, 2006.
5. Benavides D., Trinidad P., and Ruiz-Cortés A. Automated reasoning on feature models. In *Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, volume 3520 of *LNCS*, pages 491–503. Springer–Verlag, 2005.
6. Benavides D., Trujillo S., and Trinidad P. On the modularization of feature models. In *In First European Workshop on Model Transformation*, 2005.
7. Jouault F., Allilaire F., Bézivin J., and Kurtev I. ATL: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, 2008.
8. Roos-Frantz F. and Segura S. Automated analysis of orthogonal variability models. a first. In *1st Workshop on Analyses of Software Product Lines (ASPL08)*, 2008.
9. Czarnecki K., Helsen S., and Eisenecker U.W. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.
10. Pohl K., Böckle G., and F. J. van der Linden. *Software Product Line Engineering: Fundations, Principles and Techniques*. Springer–Verlag, Berlin, DE, 2005.
11. Bencomo N. *Supporting the Modelling and Generation of Reflective Middleware Families and Applications using Dynamic Variability*. PhD thesis, Lancaster University, 2008.
12. Loughran N., Sánchez P., Garcia A., and Fuentes L. Language support for managing variability in architectural models. In *Software Composition*, pages 36–51, 2008.
13. OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Final Adopted Specification ptc/05-11-01.
14. Clements P and Northrop L. *Software Product Lines: Practices and Patterns.* SEI Series en Software Engineering. Addison–Wesley, August 2001.
15. Trinidad P., Benavides D., Durán A., Ruiz-Cortés A., and Toro M. Automated error analysis for the agilization of feature modeling. *J. of Systems and Software*, 81(6):883–896, 2008.
16. Trinidad P., Benavides D., Ruiz-Cortés A., Segura S., and Jimenez A. Fama framework. In *Software Product Line Conference Tool Demonstrations (in press)*, 2008.