FaMa-OVM: A Tool for the Automated Analysis of OVMs

Fabricia Roos-Frantz, José A. Galindo, David Benavides and Antonio Ruiz-Cortés
University of Seville
41012 Seville, Spain
{fabriciaroos,jagalindo,benavides,aruiz}@us.es

ABSTRACT

Orthogonal Variability Model (OVM) is a modelling language for representing variability in Software Product Line Engineering. The automated analysis of OVMs is defined as the computer-aided extraction of information from such models. In this paper, we present FaMa-OVM, which is a pioneer tool for the automated analysis of OVMs. FaMa-OVM is easy to extend or integrate in other tools. It has been developed as part of the FaMa ecosystem enabling the benefits coming from other tools of that ecosystem as FaMaFW and BeTTy.

Categories and Subject Descriptors

D.2.2.a [**Software Engineering**]: Design Tools and Techniques—*CASE*

Keywords

Software Products Lines, OVMs, Tools

General Terms

Management, Documentation, Verification

1. INTRODUCTION

Variability modelling is an important task in Software Product Line Engineering (SPLE). There are many kinds of models used to represent such variability [3]. One of them is the Orthogonal Variability Model (OVM) [7], which focuses on separating the representation of variability from the representation of the various SPLE artefacts.

The configuration of a product is done by selecting desired and valid options in the variability model during application engineering. For example, in a mobile phone product line, an option could be the selection of the screen resolution, which can be basic or high. This option may demand a constraint to ensure that the products support only one screen resolution simultaneously. In the software product line community, it is well-known that such variability in product lines is increasing [1, 5, 11]. Therefore, variability models may have thousands of options, and these options may have complex dependencies among them. That makes the management of such models practically impossible without an automated tool support; their manual analysis is a difficult and error-prone task.

In addition, the specification of variability can be extended with measurable attributes (e.g., CPU and memory consumption) and constraints on these attributes (e.g., memory consumption should be in a range of values) in order to express some properties about different products [2]. For example, in cases in which there are limitations of resources such as memory capacity and CPU time, the derivation of products that does not satisfy those conditions must be avoided. That fact makes the manual analysis more difficult, leading to a need for automation.

The automated analysis of variability models is an active research topic that has received the attention of many researchers during the last twenty years. It can be defined as the computer-aided extraction of information from variability models [2]. Some examples of analysis on variability models are: checking whether a configuration is valid, checking whether a model is void, etc [2]. Most of the research in this field has been focused on feature models. There are a number of approaches providing automated support for their analysis [2]. They use different logical paradigm or formalism (e.g. Description logic, Propositional logic, Constraint programming) and most of them use BDD[1], SAT[2] or CSP[3] off-the-shelf solvers to automate the analysis.

The existence of other variability models is naturally leading to the need for new techniques and tools to support their automated analysis as well. To the best of our knowledge, only Metzger et al. [6] and we [8] have explored the automated analysis of OVM. In [6], the authors propose an indirect way to automatically analyze OVMs, i.e. by means of the transformation of OVM into VFD^{+}[4] and in doing so, they reuse the semantics of analysis operations on VFD^{+}. In order to automate the analysis they map the analysis operations to the boolean satisfiability problem SAT [4] and use the solver SAT4j. In [8], we proposed an approach to enrich OVMs with attributes and then used Choco solver to automate the analysis of the enriched models.

In this paper, we present FaMa-OVM, which is a tool for

SPLC - Vol. II, Sep 02-07 2012, Salvador, Brazil

[1]JavaBDD solver, http://javabdd.sourceforge.net
[2]SAT4j http://www.sat4j.org
[3]Constraint Satisfaction Problem www.4c.ucc.ie/
[4]Varied Feature Diagram (VFD^{+}) is a formal "back-end" language used to define semantics and automating analysis

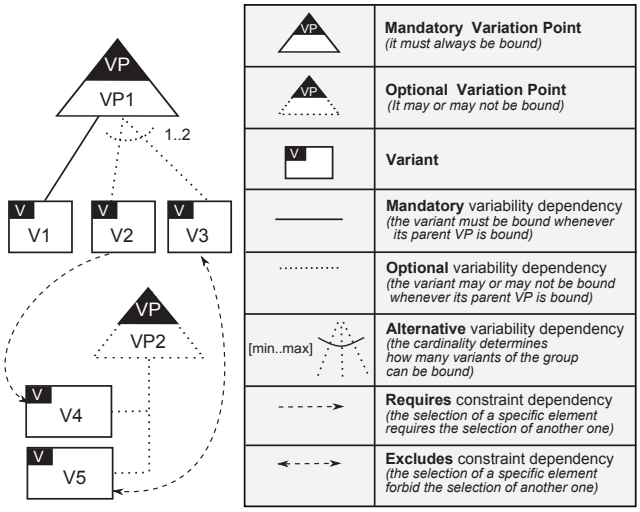

Figure 1: OVM graphical notation

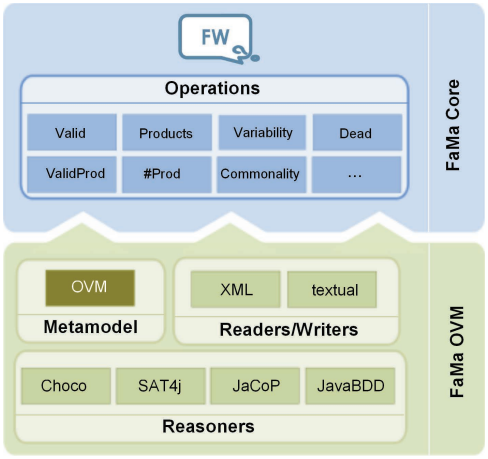

Figure 2: FaMa-OVM an extension of FaMa FW

the automated analysis of OVMs. The main advantage of FaMa-OVM is that it is developed based on a framework devised to facilitate the development of tools for the analysis of variability models. Therefore, it is easy to extend and integrate into other tools. FaMa-OVM provides support for the analysis of OVMs using three different logical paradigms: description logic, propositional logic, and constraint programming. Our tool also supports the analysis of OVMs with attributes, as proposed in [8].

The remainder of this paper continues as follows. Section 2 presents an overview of the OVM language with and without attributes. Section 3 shows the architecture of the FaMa-OVM tool, its capabilities, and its extension points; it also describes the analysis operations supported by our tool, and presents a user-oriented scenario, and finally, Section 4 presents concluding remarks.

2. BACKGROUND

OVM is a modelling language proposed to define the variability of a software product lines [7]. An OVM is composed of two main elements: *variation points (VPs)* and *variants (Vs)*. A variation point documents what can vary within artefacts of the product line, i.e., where differences exist in the final software product, and are chosen by the customer or engineer of the software product line. For instance, products may differ with respect to operating systems they support, with respect to whether they provide access to the internet or not, and so on. A variant is related to a variation point and documents how such variation point can vary. For instance, products may come with two different operating systems. In summary, an OVM provides an abstract representation of all the *variations* of the product line, which are determined by all the possible combinations of variation points and variants in the OVM.

Figure 1 shows an example of a graphical notation for OVM. The meaning of each graphical element and their relationships is described on the right hand side.

All variation points are related to at least one variant and each variant is related to one variation point. Variation points can be either optional or mandatory. A mandatory variation point must always be bound in all the products of the product line. An optional variation point can be optionally bound. Binding a variation point means making a decision about its variants children. The relationships between the elements of OVM can be of two types:

- **Variability dependencies.** They define the rules that constrain the possible options (variants) to a variation point. Variability dependencies can be of three types, namely: *mandatory*, *optional*, and *alternative*. The cardinality in the alternative relationship determines how many variants can be chosen simultaneously.

- **Constraint dependencies.** They define possible dependencies and incompatibilities amongst variant selections. They are of two forms: *excludes* and *requires*.

Recently, we have proposed an approach to associate attributes with OVMs [8]. In that work we defined an attribute as a measurable property of an artefact. We considered only those properties that can be quantified and technically defined. For example, the version of a system.

As defined in [8], an attribute consists of a *name*, a *domain*, a *value*, a *nullValue*, and a *unit*. Name denotes the name of the attribute which does not need to be unique. Domain represents the range of values that the attribute may hold such as Reals, Integers, and any range (e.g. $[1 \ldots 512]$); Value denotes the attribute value which will depend on the concrete type of attribute. NullValue denotes the value that must be taken by the attribute when the variant with which the attribute is related is not selected. Unit denotes a determinate quantity such as metres, currency and kilobytes, adopted as a standard for measurement.

3. THE FaMa-OVM TOOL

FaMa-OVM has been built on top of FaMa FW [10], which is an open source Java framework originally designed for the automated analysis of feature models. In its origins, FaMa FW was designed to support different sorts of feature models but recently it has evolved to support different kinds of variability models and variability description languages.

FaMa FW provides a number of extension points to plug in new artefacts, such as metamodels, readers/writers and

reasoners. Figure 2 shows an overview of FaMa-OVM artefacts. In the following, we report on those artefacts we have plugged in:

- The *OVM metamodel* describes the different OVM elements, and the rules that constraint the combination of those elements. Furthermore, it describes the attributes and their relationship with elements in the OVM, as well as the constraints on attributes.
- The *OVM Readers/writers* allows FaMa-OVM to be capable to store/extract the information present in the metamodel as a textual format.
- The *OVM reasoner* enables a specific solver for the analysis. For the analysis of OVM without attributes, we used three different solvers: SAT4j, JavaBDD and Choco solvers. But, for the analysis of OVM with attributes we used Choco solver. That solver enables working with numerical values, such as integers, which allows to deal with attributes, enabling it to maximize or minimize values.

FaMa-OVM is part of the FaMa ecosystem, once that it was built using the FaMa FW. That fact makes easy to extend or integrate FaMa-OVM into other tools, prototypes or proposals that have been done for the FaMa FM (the feature model version). For instance, we developed a metamorphic generator extending the BeTTy tool [9] which allows us to perform functional testing over our new tool.

BeTTy is an extensible and highly configurable framework supporting BEnchmarking and TestTing on the analYses of feature models, we extended it easily to support also OVMs. BeTTy offers a set of different generators for feature models, but having also a set of abstract classes that allows to extend it to support others variability description approaches.

3.1 Supported analysis operations

FaMa-OVM provides a number of operations to analyse a model without modifying it. Each operation takes an OVM as input and returns a response as result. More details about analysis operations can be found in [2]. Next, we summarize some of the analysis operations provided by FaMa-OVM:

Operations for non-attributed OVMs:

- *Void OVM*: Checks whether an OVM is void or not. An OVM is void when it does not represent any variation.
- *Variations*: Takes as input an OVM model and returns all the variations represented by the input model.
- *Valid Configuration*: Takes an OVM and a configuration (set of variants and/or variation points) as input and returns a value (true or false) informing whether the input configuration is valid to such OVM or not.
- *Filter*: It takes as input an OVM and a configuration and returns the set of variations including the input configuration that can be derived from the model.

In addition to non-attributed operations that also can be performing when using attributes, FaMa-OVM also offers a bunch of operations for attributed models only. Next, we describe two of them.

Operations for attributed Orthogonal Models:

- *Optimal variation*: The optimal variation is the variation that satisfies all the constraints imposed by the model and also minimises or maximises a given objective function. When associating OVM with attributes, we are able to ask for an optimal variation, since the objective function takes into account values of attributes.
- *Optimal variation with attribute condition*: The engineer of a product line may want to verify which is the optimal variation that satisfies some attribute condition and a desired partial configuration.

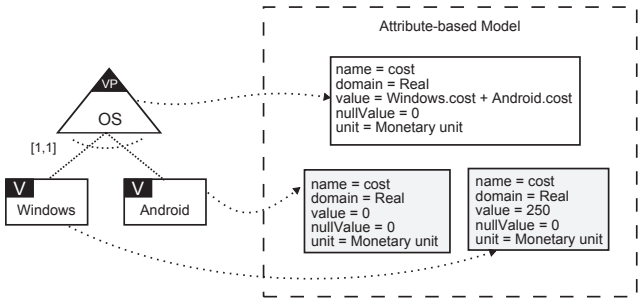

Figure 3: OVM associated with attributes

3.2 An user-oriented scenario

Variability documented in the OVM may be not enough to guarantee the success of a product. Software engineers should be able to realise the impact of their choices on the attributes, and thus, decisions about the most suitable product to be built should take into account those attributes. To demonstrate the relevance and usefulness of the proposed association of OVM with attributes and the automation of its analysis, we present the following user-oriented scenario.

An engineer of a mobile phone product line wants to know what is the impact of her/his choice on the cost of a product. Depending on the selected operating system, costs of a product may change. Figure 3 shows how we represent this variability using OVM and attributes. Products from this product line can have *Windows* or *Android* operating system. Furthermore, there are two basic attributes: *i) cost*, related to *Windows*, and *ii) cost*, related to *Android* (basic attributes are depicted in grey). These attributes determine that each operating system has a different system cost. In addition, there is one derived attribute: *cost*, related to variation point *OS*. This derived attribute expresses the system cost regarding the type of operating system selected, and it is the sum of costs of basic attributes, namely *Windows.cost* and *Android.cost*. Note that we textually represent each attribute by relating the variability element and the attribute with the form <*variability-element.attribute*>. For example, *Windows.cost* defines the relationship between variant *Windows* and attribute *cost*.

Suppose that the engineer wants to find the variation with the lowest system cost. How to find this answer using FaMa-OVM? First, the OVM and the attributes are specified using the FaMaOVM textual format, which is shown in Figure 4.

The textual format consists of four main parts, namely: *i) Relationships*, specifying the variability dependencies between variation points and variants; *ii) Attributes*, specifying basic and derived attributes; *iii) Global Attributes*, specifying global attributes, and *iv) Constraints*, specifying excludes and requires relationships, and domain constraints. Attributes are defined in the form of `<name>:<domain>,<value>,<nullValue>;` where lines are finished with semicolon, and

```
%Relationships
 OS : [1,1]{Windows Android};

%Attributes
 Windows.cost: Integer[1 to 500], 250, 0;
 Android.cost: Integer[1 to 500], 0,0;
 OS.cost: Integer[1 to 1000], Windows.cost + Android.
      cost; , 0;

%Global Attributes

%Constraints
```

Figure 4: OVM and attributes in textual format

the terms are separated by commas. When the value of an attribute is a function, a semicolon after `<value>` is used.

After the input model has been specified in textual format, the FaMa-OVM tool takes this as input, and thus, using a CSP reasoner, analyses it to find the optimal variation. As a result, FaMa-OVM finds that the lowest system cost is the one in which Android is selected, since the total cost of the OS is the sum of Android.cost, which is zero when Android is selected (represented by value), and windows.cost, which is zero when windows is not selected (represented by nullValue).

Note that, in this scenario, the problem seems to be simple to be solved, however, when there are many variation points and relationships amongst variability and attributes are more complex, such analysis is practically impossible to be done manually. Therefore, finding the optimal solution, as opposed to any possible solution, would be helpful for making attribute-aware decisions.

4. CONCLUSIONS AND FUTURE WORK

Although the automated analysis of OVMs has been explored by Metzger et al.[6], they do not offer a tool. FaMa-OVM is the first tool provided to automate such analysis and it is based on a framework for the analysis of feature models, which is a research area with more know-how. We realised that the framework simplified the development of our tool, we did not have to start from scratch.

We summarize the benefits of FaMa-OVM:

- *Tool support for OVM*. The users of OVMs can easily analyze their OVMs by means of a simple public interface, just needing to entry with the OVM (sometimes with some more data, it depends on the operation) and to select the operation to be answered by the tool.
- *Multiple Reasoners*: FaMa-OVM integrates multiple off - the-shelf solvers enabling the analysis of OVMs using three different logical paradigms: propositional logic, description logic, and constraint programming.
- *Extensible tool*: As FaMa-OVM is an extension of FaMa FM, it offers the same advantages as the framework, in particular ease of extending, enabling the analysis of OVM and integration with other the tools that are part of the FaMa ecosystem.
- *Easy to integrate and configure*: FaMa-OVM is easy to integrate with other tools, due to its simple front-end Java interface, and easy to configure, since its configuration is done by means of a unique XML file. For example, it can be easily integrated in the VarMod-Editor[5], enriching the editor with analysis support.

We are currently working to identify new analysis operations that may help to product line engineers when using OVMs. We also are working into increase the quality of our tool by doing performance analysis using the BeTTy tool. In the end, we are also working to quantify how much time and costs we saved extending FaMa FW instead of creating a totally new tool.

[5]see http://www.sse.uni-due.de/varmod

5. ACKNOWLEDGEMENT

This work has been partially supported by the European Commission (FEDER) and the Spanish Government under CICYT project SETI (TIN2009-07366), by the Andalusian Government under ISABEL (TIC-2533), THEOS (TIC-5906) projects and Talentia scholarships, by Evangelischer Entwicklungsdienst e.V. (EED).

6. REFERENCES

[1] D. Batory, D. Benavides, and A. Ruiz-Cortés. Automated analysis of feature models: Challenges ahead. *Communications of the ACM*, December:45–47, 2006.

[2] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: a literature review. *Information Systems*, 35:615–636, 2010.

[3] L. Chen, M. A. Babar, and N. Ali. Variability management in software product lines: a systematic review. In *SPLC*, pages 81–90, 2009.

[4] S. Cook. The complexity of theorem proving procedures. In *Proc. of ACM symposium on Theory of computing*, pages 151–158, 1971.

[5] F. Loesch and E. Ploedereder. Optimization of variability in software product lines. In *Proceedings of the 11th International Software Product Line Conference (SPLC)*, pages 151–162, Washington, DC, USA, 2007. IEEE Computer Society.

[6] A. Metzger, K. Pohl, P. Heymans, P. Schobbens, and G. Saval. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In *Int. Requirements Engineering Conf.*, pages 243–253, 2007.

[7] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Fundations, Principles and Techniques*. Springer–Verlag, Berlin, DE, 2005.

[8] F. Roos-Frantz, D. Benavides, A. Ruiz-Cortés, A. Heuer, and K. Lauenroth. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal*, pages 1–47, 2010. 10.1007/s11219-011-9156-5.

[9] S. Segura, J. Galindo, D. Benavides, J. Parejo, and A. Ruiz-Cortés. Betty: Benchmarking and testing on the automated analysis of feature models. In U. Eisenecker, S. Apel, and S. Gnesi, editors, *Sixth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'12)*, page 63–71, Leipzig, Germany, 2012. ACM, ACM.

[10] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jiménez. Fama framework - poster. In *SPLC*, page 359, Sep 2008.

[11] J. White, B. Doughtery, and D. Schmidt. Selecting highly optimal architectural feature sets with filtered cartesian flattening. *Journal of Systems and Software*, 82(8):1268–1284, 2009.