

Automated Analysis of Orthogonal Variability Models using Constraint Programming

Fabricia Roos-Frantz

Depto. de Tecnología

UNIJUÍ University, Ijuí, Brazil

frfrantz@unijui.edu.br

David Benavides, Antonio Ruiz-Cortés

Depto. de Lenguajes y Sistemas Informáticos

University of Seville, Seville, Spain

{benavides, aruiz}@us.es

Abstract

Software Product Line (SPL) Engineering is about producing a family of products that share commonalities and variabilities. The variability models are used for variability management in SPLs. Currently, the automated analysis of variability models has become an active research area. In this paper we focus on the automated analysis of Orthogonal Variability Model (OVM), which is a modelling language for representing variability. The automated analysis of OVMs deals with the computer-aided extraction of information from OVMs. The automated analysis of OVMs has been hardly explored and currently has no tooling support. Considering our know-how to analyse feature models, which are the most popular variability models in SPLs, we propose to automate the analysis of OVMs by means of constraint programming. In addition, we propose to extend OVMs with attributes, allowing to add extra-functional information to OVMs. With this proposal we contribute with a step forward toward a tooling support for analysing OVMs.

1 Introduction and motivation

According to Clements and Northrop [8], a Software Product Line (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 com-*

mon set of core assets in a prescribed way”. The main idea behind SPL is the development of a software family instead of a single software product. An SPL is composed of a set of products which are constructed from a common core asset designed for a specific domain.

In the SPL context, the variability models document the variability of a product line, i.e the possible combinations of features in a system. In this context, a feature might be defined as an increment in the functionality of a system [4]. Variability models are important in SPL Engineering due to their role in documenting and managing of variability, making easier the task of management and development of an SPL [16, 10]. Nowadays, there are different kinds of variability models used in SPL, such as feature models, decision models and orthogonal variability models.

The *Orthogonal Variability Model (OVM)* is a modelling language for representing variability in SPL [13]. Every variability in the SPL is documented in the OVM model by means of variation points with their respective variations, but not the commonalities. Those features that are common to all software products would be documented in other artefact models, such as requirement models, design models, etc. Therefore, the SPL variability is explicitly represented by the OVM models.

The automated analysis of variability models deals with the computer-aided extraction of information from such models [6]. It is an important task in the context of SPL, since

it is practically impossible to do it manually, and besides it is error prone [6, 4]. In addition, the variability models are one of the main artefacts of the domain engineering [13] and therefore their analysis in an early stage of development is essential to the success of the SPL. Although the automated analysis of variability models is an active research area, the majority of the researches has focused on feature models. In [6], the authors review a number of proposals providing automated support for the analysis of feature models, by using different logical paradigm or formalism ,e.g. Description logic, Propositional logic, Constraint programming. Most of them use BDD¹, SAT² or CSP³ off-the-shelf solvers to automate the analysis.

To the best of our knowledge, only Metzger et al. explored the automated analysis of OVM [11]. They propose an indirect way to automatically analyse OVMs, i.e. by means of the transformation of OVM into VFD⁺ and in doing so, they reuse the semantics of analysis operations on VFD⁺. To carry out this transformation they provide an ad hoc algorithm. In order to automate the analysis they map the analysis operations to a propositional formula and use the solver SAT4j.

The contribution of this paper is twofold. First, we extend OVMs with attributes to support the modelling of extra-functional aspects. Second, we propose the use of constraint programming to provide automated analysis of *Extended OVMs*.

The remainder is organized as follows: Section 2 describe the OVM language and the proposed Extended OVM; Section 3 discusses about the analysis operations on OVMs; Section 4 describes our proposal where we provide a mapping from an Extended OVM to a Constraint Satisfaction Problem (CSP). In addition, we define some of the automated analysis operations on Extended OVM

through CSP; and Section 5 presents our conclusions.

2 The OVM language

Orthogonal Variability Model is a modelling language for defining the variability of SPL [13, 11]. OVM provides a separate view of the variability documenting explicitly the variation points in a separate model. In the OVM models the first-classes are: *variation points (VP)* and *variants*. A *variation point* documents the functional aspects that vary in the SPL, i.e those aspects that represent a variability, which must be chosen by the customer or engineer of the SPL. A *variant* is related to a variation point and documents how this variation point can vary. An OVM represents all possible configurations of an SPL, where configurations mean all possible combinations of variation points and variants.

2.1 OVM Notation

In Figure 1 we show a possible OVM of an SPL in the E-shop domain, it is partially inspired by [12]. A VP, graphically represented by a triangle, can be either *mandatory* or *optional*. A mandatory VP (solid line) must always be bound, i.e. its variants must be chosen. An optional VP (dashed line) may or may not be bound. For instance, any configuration represented by the model in Figure 1 must have *customertype* and *current* resources and may or may not have *customer-profile* resource.

In addition to the VPs and variants, OVM defines two kinds of relationship between elements, *variability* and *constraint* dependencies. A variability dependency is a relationship between a variant and its parent VP, which can be *Mandatory*, *Optional* or *Alternative*, as follows:

- *Mandatory variability dependency*. The variant must be chosen whenever its parent VP is bound. For instance, between the *cardtype* VP and the *creditcard* variant there is a mandatory dependency, it means that always that *cardtype* is part

¹JavaBDD solver, <http://javabdd.sourceforge.net>

²SAT4j solver <http://www.sat4j.org>

³Constraint Satisfaction Problem www.4c.ucc.ie/

⁴Varied Feature Diagram (VFD⁺) is a formal “back-end” language used to define semantics and automating analysis

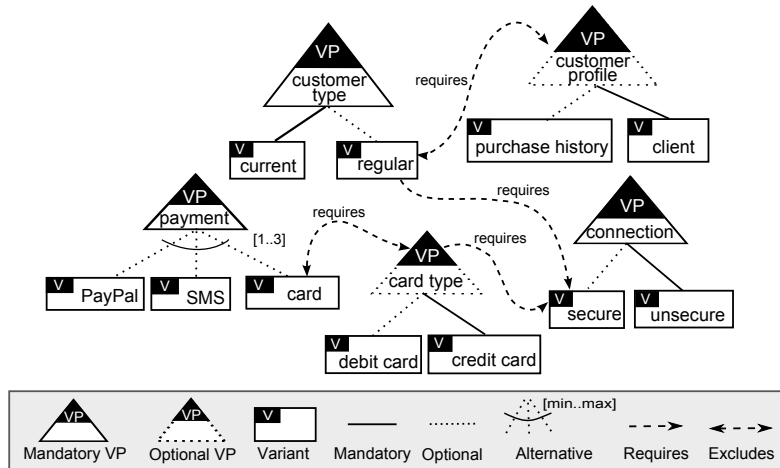


Figure 1: A sample of orthogonal variability model of an SPL in the E-shop domain

of a configuration, *creditcard* must be as well.

- *Optional variability dependency.* The variant can, but not have to be chosen whenever its parent VP is bound. If we take the example, we can observe the relationship between *cardtype* and *debitcard*, it means that even if the software product offers payment by card, the debit card type may or may not be offered.
- *Alternative variability dependency.* Consists of a group of optional dependencies and a given cardinality $[min..max]$. The 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. For instance, the product line offers three different payment methods, *Paypal*, *SMS* and *card*. The cardinality $[1..3]$ says that at least one and at most 3 methods can be part of the configuration.

A constraint dependency is a relationship between variants, between variants and VPs,

and between VPs. These relationships are defined graphically and can be of two types, namely *Requires* and *Excludes*, as follows:

- *Requires constraint dependency.* A requires specifies an implication, i.e. if a variant or a VP called *A* requires another variant or VP called *B*, then if *A* is chosen, *B* has to be chosen as well. For instance, if a configuration has a regular *customer*, it must include the *secure* connection.
- *Excludes constraint dependency.* An excludes specifies a mutual exclusion, i.e. if a variant or a VP called *A* excludes another variant or VP called *B*, *B* can not be bound whenever *A* is chosen, and vice versa.

2.2 Extended OVM

Extra-functional aspects are crucial when modelling an SPL [6, 7], thus it is important that modelling techniques deal with them [16]. However, the OVM deals only with variability related to the functional aspects offered by the SPL and therefore does not address extra-functional variability. In

order to deal with extra-functional aspects, we propose to extend OVM with attributes.

In Figure 1, the variation points and variants represent functional variability. Every configuration represented by this model differs because of its functional variability. For instance, consider the following configurations C1 and C2, they differ because C1 offers payment through SMS and C2 offers payment through PayPal method.

```
C1 = {customertype,connection,payment,SMS,
      current,unsecure}
C2 = {customertype,connection,payment,
      PayPal,current,unsecure}
```

However, adding attributes to OVM, we associate extra-functional aspects with the variation points and variants. For instance, the *payment* variation point could have extra-functional variability related to it, such as availability, efficiency, development time, and so on. In doing so, it is possible differ one configuration from another also by the extra-functional aspects. For instance, if the variant *secure* connection offered different key lengths for encrypted remote communication, with Extended OVM we could define an attribute for it called *keylength*, which can vary from 128 to 1024 bits. Thus, those configurations that offers the same secure connection resource can differ according to their attribute *keylengths*.

In addition to adding attributes, we propose the possibility of relationships amongst attributes, e.g. a value of an attribute can be a relationship between values of other attributes, e.g. *price/time*; and also relationships amongst attributes and variable elements (i.e. a variation point or a variant). Before introducing how we extend OVM, we would like to make clear the following concepts:

- *Attribute*: the attribute of a variable element is any property of a variable element that can be measured.
- *Attribute value*: any value belonging to the domain value, or a complex constraint.

- *Domain Value*: the range of possible values of an attribute. Every attribute has a domain. The domain can be discrete (e.g. integers, boolean), continuous (e.g. real) or a complex type.
- *Complex constraint*: consists of a relationship among attributes or among attributes and variable elements. For instance: “If attribute *A* of a variation point *VP* is greater than a value *X*, then variant *V* can not be part of the product”.

Figure 2 shows an example of how to associate extra-functional aspects to OVMs. In this example, we show an excerpt from the OVM of Figure 1 with extra-functional features. An attribute has a name, a domain and a value ⁵. In this example each variant has two attributes: “*cost*” and “*confidentiality*”. *Confidentiality* (expressed in levels) refers to the privacy level of payment details, and *cost* concerns the development cost of each variant or variation point. The cost attribute has a real domain and the confidentiality attribute has an integer domain. The value of attribute cost of each variant takes a range of values in the real domain, and the value of confidentiality is an integer from 1 to 5. And finally, the payment variation point has an attribute called *cost*, which is a real number and its value is the sum of costs of payment variants.

3 Analysis of Extended OVMs

In the SPL community is well known that variability in product lines is increasing, the variability models may have thousands of variants [7]. Furthermore, these variants usually have complex dependencies between them [3]. Therefore, it is necessary to rely on automatic support to analyse and manage variability models. In [6], Benavides et al. say the automated analysis of feature models deals with the computer-aided extraction of information from feature models. We use the same definition to the automated analysis of OVMs, considering that it deals with

⁵The values in the example are just illustrative

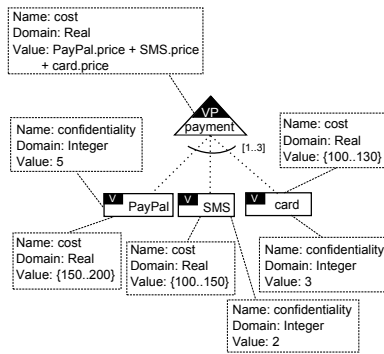


Figure 2: Extended Orthogonal Variability Model

the computer-aided extraction of information from OVMs.

What kind of information would be interesting to extract from OVMs? For instance, we may want to know how many configurations are represented in a model, or even to know whether a specific configuration belongs to the model. In [6] the authors surveyed a number of approaches addressing automated analysis of feature models. Such analysis is done by means of analysis operations, which are specifically defined to analyse feature models and comment on the properties of such models. Taking into account the research results obtained on analysis of feature models, we reuse the know-how of this area in order to introduce analysis operations on OVMs.

In our previous work [14] we took the first step towards the automated analysis of OVMs. In that work, we suggested some analysis operations on OVMs. In this paper, we propose some more analysis operations on *Extended OVM*, which can be applied also to existing OVM. These operations observe the properties of a model without modifying it, by taking an *Extended OVM* model as input and providing a response as result. The information obtained during the analysis process can be useful to guide marketing strategies and technical decisions. In the following we describe some operations:

Number of configurations. This operation returns the total number of configurations represented by the Extended OVM. For instance, the model depicted in Figure 1 represents 36 configurations. One of them is {*customertype, connection, payment, SMS, current, unsecure*}. This operation provides information about flexibility and complexity of the SPL. In the E-Shop example of Figure 1, if we simply remove the requires from *card* to *cardtype* the number of products raises to 52.

All configurations. This operation takes as input an Extended OVM and returns all configurations represented by such model, i.e all the possible combinations of variation points. It is worth highlighting that in an OVM model a configuration could be empty, since there is no root as in feature models. In other words, if there is no mandatory variation point in an OVM, there would be no mandatory elements, what would lead to an empty configuration. By applying this operation to the OVM of Figure 1, we obtained 36 configurations, three of them are detailed bellow:

- C1 = {*customertype, connection, payment, SMS, current, unsecure*}
- C2 = {*customertype, connection, payment, PayPal, current, unsecure*}
- C3 = {*customertype, connection, payment, PayPal, SMS, current, unsecure*}

Void model. Checks whether an Extended OVM is void or not, i.e. if it represents at least one valid configuration. An Extended OVM may becomes void due to the wrong usage of *excludes constraint dependencies*. In Figure 3, we can see an example of a void OVM, where there is no valid configuration due to the excludes between *A* and *D*.

Valid configuration. Takes an Extended OVM model and a configuration (set of variation points and variants) as input and returns a value that determines whether the configuration belongs to the set of configurations represented by the model or not. As an example of this operation, we can take as input the following products C1, C2 and C3 and the OVM

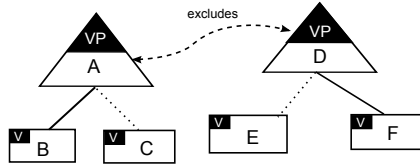


Figure 3: A void OVM

model in Figure 1. Then, we receive as result that C1 and C2 are valid configurations, however C3 is not valid, because it does not have the mandatory variant *current*.

- C1 = {customertype,connection,payment,SMS,current,unsecure}
- C2 = {customertype,connection,payment,card,current,unsecure}
- C3 = {customertype,connection,payment,SMS,unsecure}

Valid partial configuration. It takes an Extended OVM model and a partial configuration as input and returns a value informing whether the configuration is valid or not, i.e. a partial configuration is valid if it does not include any contradiction. Given an Extended OVM with a set of variants and variation points V , a partial configuration is a 2-tuple of the form (S, R) such that $S, R \subseteq V$ being S the set of variation points and variants to be selected and R the set of variation points and variants to be removed such that $(S \cap R = \emptyset) \wedge (S \cup R \subseteq V)$. As an example, considering the model in Figure 1, the following partial configurations PC1 and PC2 are respectively not valid and valid:

- PC1 = ({customertype,connection,current,regular}, {secure,SMS})
- PC2 = {customertype,connection,current,payment}, {secure,card}

PC1 is not a valid partial configuration because it selects *regular* customer type and removes *secure* connection, which is explicitly required by the SPL. PC2 is a valid partial configuration since it does not include any contradiction. This operation is helpful specially during the product derivation stage.

Filter. It takes as input an Extended OVM model and a configuration (potentially par-

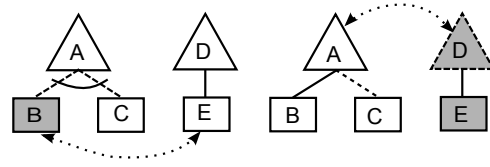


Figure 4: Common cases of dead nodes in OVM models. Grey nodes are dead

tial) and returns the set of configurations including the input configuration that can be derived from the model. For instance, the set of products of the OVM model in Figure 1 applying the partial configuration $(S, R) = (\{regular, Paypal\}, \{SMS, debitcard\})$ is:

- P1 = {customertype,customerprofile,connection,payment,client,PayPal,current,secure,unsecure,regular}
- P2 = {customertype,customerprofile,connection,payment,purchasehistory,client,PayPal,current,secure,unsecure,regular}
- P3 = {customertype,customerprofile,connection,payment,cardtype,client,PayPal,card,current,secure,unsecure,creditcard,regular}
- P4 = {customertype,customerprofile,connection,payment,cardtype,purchasehistory,client,PayPal,card,current,secure,unsecure,creditcard,regular}

Dead node. It returns a set of dead nodes (if any), i.e. those variants or variations points that cannot appear in any of the configurations represented by the model. Dead nodes are caused by a wrong usage of constraint dependencies. It is important to detect dead nodes since they give a wrong idea of the variability. In Figure 4 we show some common cases that generate dead nodes in OVM.

Optimization. Finding the optimal solution, and not only any possible solution, would be helpful for solving a constraint problem. Hence, in order to turn up the optimal solution we can associate an objective function with the CSP. Such kind of problem is referred as Constrained Solution Optimization Problem (CSOP) and its main task is to find solutions that maximize or minimize a specified objective function satisfying all the constraints. For instance, if we want to find out the set of solutions that minimize the cost of payment resource in the Extended E-shop example in Figure 2 we can ask for an optimization. First we need to apply a filter to the

model in order to obtain a filtered model with $payment = true$. Second, we define the objective function as $O = payment.cost$. Third, we can ask for the solutions that optimize O .

$$M = filter(E - shop, payment = true)$$

$$O = payment.cost$$

$$S_{opt} = min(M, O)$$

4 Automating the Analysis of Extended OVM

In [6], Benavides et al. define a conceptual framework where they propose a process for the automated analysis of feature models. Based on it, we define the process presented in Figure 5 as the whole process for the automated analysis of OVMs. First, an OVM model is mapping into a logical representation, in this case into CSP. Afterwards, the analysis operations to be applied to the CSP model are defined as CSP primitives. Finally, an off-the-shelf CSP solver is used to automatically analyse the input data and provide the analysis results.

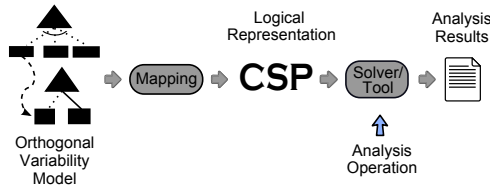


Figure 5: Automated analysis process of OVM using CSP.

4.1 Background: Orthogonal Variability Models and Configurations as CSP

Constraint Programming is a discipline which relies on a set of techniques and algorithms to deal with reasoning and computing [2]. It is devoted to modelling with constraints and to solving the resulting constraint satisfaction problems (CSPs). A *Constraint Satisfaction Problem* [19] is defined as a set of variables and a set of constraints restricting the values

of these variables. For example, $A + B > 1$ is a CSP involving the integer variables A and B . A constraint solver finds a valid set of variable values that simultaneously satisfies all constraints in the CSP. ($A = 2, B = 2$) is thus a valid solution of the CSP $A + B > 1$.

To build the CSP for the automated analysis of OVMs, we construct a set of variables V , representing the variable elements (variation points and the variants) in the OVM. Each configuration of the OVM is a set of values (0 or 1) for these variables. The value of 1 indicates the variable element is present in the configuration and a value of 0 indicates it is not present. More formally, a configuration is a set of variable values of V , such that $\forall v_i \cdot v_i \in V \Rightarrow v_i = 0 \vee v_i = 1$. If $v_i = 1$ indicates that v_i is selected in the configuration. Similarly, if $v_i = 0$ means that v_i is not selected.

In the CSP equivalent to the OVM, each variable v_i can have one or more constraints associated with it corresponding to the configuration rules in the OVM. For example, if v_i excludes v_j , then the CSP would contain the constraint: *if*($v_i = 1$)*then*($v_j = 0$). Therefore, the CSP has a set of constraints C which captures the configuration rules from the OVM. For any given OVM configuration described by the set of variable values of V the correctness of the configuration can be determined by seeing if the values satisfy all constraints in C .

4.2 Related Work

Benavides et al. were the first authors who proposed using constraint programming for analyses on feature models [7, 5]. They provide a set of mapping rules to translate a feature model into a CSP, and a support to feature models with attributes. The authors also provide tool support [18]. Trinidad et al. [17] propose constraint programming and Reiter’s theory of diagnosis to detect and offer explanation of errors in feature models. In [9] the authors describe a tool under development addressing the analysis of feature models using constraint programming. White et


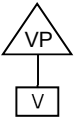
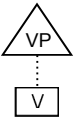
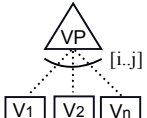
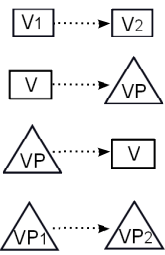
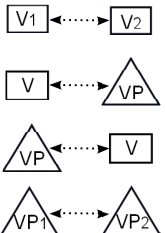
| | Variation Point | CSP Mapping | E-shop Example |
|-------------|---|--|--|
| MANDATORY |  | $vp = 1$ | customertype = 1 payment = 1 connection = 1 |
| | Variability Dependency | CSP Mapping | E-shop Example |
| MANDATORY |  | $vp = v$ | customertype = current customerprofile = client cardtype = creditcard connection = unsecure |
| OPTIONAL |  | if ($vp = 0$) $v = 0$ | if (customertype = 0) regular = 0 if (customerprofile = 0) purchasehistory = 0 if (cardtype = 0) debit card = 0 if (connection = 0) secure = 0 |
| ALTERNATIVE |  | if ($vp > 0$) Sum (v_1, v_2, \dots, v_n) in $\{1..j\}$ else $v_1 = 0, v_2 = 0, \dots, v_n = 0$ | if (payment > 0) Sum (PayPal, SMS, card) in $\{1..3\}$ else PayPal = 0, SMS = 0, card = 0 |
| | Constraint Dependency | CSP Mapping | E-shop Example |
| REQUIRES |  | if ($v_1 > 0$) $v_2 > 0$ if ($v > 0$) $vp > 0$ if ($vp > 0$) $v > 0$ if ($vp_1 > 0$) $vp_2 > 0$ | if (regular > 0) customerprofile > 0 if (customerprofile > 0) regular > 0 if (regular > 0) secure > 0 if (cardtype > 0) secure > 0 if (cardtype > 0) card > 0 if (card > 0) cardtype > 0 |
| EXCLUDES |  | if ($v_1 > 0$) $v_2 = 0$ if ($v > 0$) $vp = 0$ if ($vp > 0$) $v = 0$ if ($vp_1 > 0$) $vp_2 = 0$ | |

Table 1: Mapping from OVM to Constraint Satisfaction Problem (CSP).

al. [20] propose a method to detect conflicts in a given configuration and propose changes in the configuration to solve the problem. Their technique is based on CSP and adding some extra variables in order to detect and correct the possible errors after applying optimization operations.

4.3 Mapping Extended OVM onto CSP

An OVM can be described in terms of restrictions imposed on the set of variables, i.e it can be defined as a CSP in a straightforward way. The modelling of an *Extended OVM* as a CSP can vary due to the solver to be used later to analyse the model. The mapping has the general form: *i*) each variation point and variant maps to a variable of the CSP with a domain of 0..1, *ii*) for each mandatory variation point a constraint assigning 1 to the correspondent variable is added, *iii*) each relationship of the model is mapped into a constraint depending on the type of the relationship, *iv*) attributes are expressed as constraints, and *v*) the resulting CSP is the one defined by the variables of steps *i*, *ii* and *iii* with the corresponding domains and a constraint that is the conjunction of all precedent constraints. The mapping from step *iii* is done as bellow:

Mandatory variability dependency. Let vp be the variation point and v the variant in a *mandatory* variability dependency, then the equivalent constraint is: $vp = v$.

Optional variability dependency. Let vp be the variation point and v the variant in a *optional* variability dependency, then the equivalent constraint is: $if(vp = 0) v = 0$.

Alternative variability dependency. Let vp be the variation point and $v_i \mid i \in [1 \dots n]$ the set of optional variants in an *alternative* variability dependency, and $[m \dots m'] \mid 0 \leq m \leq m' \leq n$ the cardinality of the alternative dependency, then the equivalent constraint is: $if(vp > 0) \text{Sum}(v_1, v_2, \dots, v_n) \text{ in } \{m..m'\}$ else $v_1 = 0, v_2 = 0, v_n = 0$.

Variant Requires Variant constraint dependency. Let $v1$ and $v2$ be the variants in a *Requires* constraint dependency, then the equivalent constraint is: $if(v1 > 0) v2 > 0$.

Variant Requires VP constraint dependency. Let v be the variant and vp the variation point in a *Requires* constraint dependency, then the equivalent constraint is: $if(v > 0) vp > 0$.

VP Requires Variant constraint dependency. Let vp be the variation point and v the variant in a *Requires* constraint dependency, then the equivalent constraint is: $if(vp > 0) v > 0$.

VP Requires VP constraint dependency. Let $vp1$ and $vp2$ be the variation points in a *Requires* constraint dependency, then the equivalent constraint is: $if(vp1 > 0) vp2 > 0$.

Variant Excludes Variant constraint dependency. Let $v1$ and $v2$ be the variants in an *Excludes* constraint dependency, then the equivalent constraint is: $if(v1 > 0) v2 = 0$.

Variant Excludes VP constraint dependency. Let v be the variant and vp the variation point in an *Excludes* constraint dependency, then the equivalent constraint is: $if(v > 0) vp = 0$.

VP Excludes Variant constraint dependency. Let vp be the variation point and v the variant in an *Excludes* constraint dependency, then the equivalent constraint is: $if(vp > 0) v = 0$.

VP Excludes VP constraint dependency. Let $vp1$ and $vp2$ be the variation points in an *Excludes* constraint dependency, then the equivalent constraint is: $if(vp1 > 0) vp2 = 0$.

In Table 1 we show the concrete rules for the mapping of an OVM into a CSP and also the mapping of the E-shop example in Figure 1 into the equivalent CSP. In this paper we provide a general mapping of an *Extend OVM* into a CSP. The detailed mapping of attributes into CSP is out of the scope of this paper. Next we show an example of how would be the equivalent CSP to the *Extended OVM* in Figure 2.

$$\begin{aligned}
& (\text{payment} = 1) \wedge \\
& (\text{if}(\text{payment} > 0) \text{Sum}(\text{PayPal}, \text{SMS}, \text{card}) \text{in} \{1..3\} \\
& \text{else}(\text{PayPal} = 0, \text{SMS} = 0, \text{card} = 0)) \wedge \\
& (\text{payment.cost} = \text{PayPal.cost} + \text{SMS.cost} \\
& + \text{card.cost}) \wedge \\
& ((\text{PayPal.cost} \in [150, 200]) \Leftrightarrow \text{PayPal}) \wedge \\
& ((\text{PayPal.cost} = 0) \Leftrightarrow \neg \text{PayPal}) \wedge \\
& ((\text{PayPal.confidentiality} = 5) \Leftrightarrow \text{PayPal}) \wedge \\
& ((\text{PayPal.confidentiality} = 0) \Leftrightarrow \neg \text{PayPal}) \wedge \\
& ((\text{SMS.cost} \in [100, 150]) \Leftrightarrow \text{SMS}) \wedge \\
& ((\text{SMS.cost} = 0) \Leftrightarrow \neg \text{SMS}) \wedge \\
& ((\text{SMS.confidentiality} = 2) \Leftrightarrow \text{SMS}) \wedge \\
& ((\text{SMS.confidentiality} = 0) \Leftrightarrow \neg \text{SMS}) \wedge \\
& ((\text{card.cost} \in [100, 130]) \Leftrightarrow \text{card}) \wedge \\
& ((\text{card.cost} = 0) \Leftrightarrow \neg \text{card}) \wedge \\
& ((\text{card.confidentiality} = 3) \Leftrightarrow \text{card}) \wedge \\
& ((\text{card.confidentiality} = 0) \Leftrightarrow \neg \text{card})
\end{aligned}$$

4.4 Analysis operations formalization

In the following, we define the analysis operations on OVM as CSP primitives:

Operation 1. (#Configurations). Let M be an Extended OVM and $\#Configurations$ the number of possible configurations of M , then $\#Configurations(M)$ is equal to the solution number of CSP ψ_M .

$$\#Configurations(M) = |\text{sol}(\psi_M)|$$

In the E-shop example of Figure 1 $\#Configurations$ (E-shop) = 36.

Operation 2. (Configurations). Let M be an Extended OVM and configurations the set of configurations of M . Thus, $Configurations(M)$ is the set of solutions of the CSP ψ_M .

$$Configurations(M) = \{s \in \text{sol}(\psi_M)\}$$

Operation 3. (Void Model). Let M be an Extended OVM, M is void if there is at least one solution to the CSP ψ_M .

$$\text{void}(M) \Leftrightarrow \#Configurations(M) = 0$$

Operation 4. (Valid Conf). Let M be an Extended OVM, a configuration C is valid if C belongs to the set of solution of CSP ψ_M .

$$ValidConf(M, C) \Leftrightarrow P \in Configurations(M)$$

Considering the E-shop example of Figure 1 and the configuration C , such that:

$$C = \{\text{customertype}, \text{customerprofile}, \text{connection}, \text{payment}, \text{client}, \text{PayPal}, \text{current}, \text{secure}, \text{unsecure}, \text{regular}\}$$

Then, $ValidConfiguration(M, C) = \text{true}$.

Operation 5. (Valid Partial Configuration (ValidPC)). Let M be an Extended OVM and PC a partial configuration, PC is valid if the set of solutions of the conjunction of CSP ψ_M and PC is not empty.

$$ValidPC(M, PC) \Leftrightarrow |\text{sol}(\psi_M \wedge PC)| > 0$$

Considering the E-shop example of Figure 1 and the partial configuration PC , such that:

$$PC = (\{\text{customertype}, \text{connection}, \text{current}, \text{regular}\}, \{\text{secure}, \text{SMS}\})$$

Then, $ValidPC(M, PC) = \text{false}$

Operation 6. (Filter). Let M be an Extended OVM and F a partial configuration representing a filter, the filtered model of ψ_M , is the conjunction of ψ_M and F .

$$\text{filter}(M, F) = (\psi_M \wedge F)$$

In the E-shop example we could get all possible configurations of the model after applying a filter. For instance, if we want to get all those configurations that offers the three types of payments (PayPal, SMS, card), thus $M = \text{filter}(E - \text{shop}, \{\{\text{PayPal}, \text{SMS}, \text{card}\}, \{\}\})$, and $Configurations(M) = \{s \in \text{sol}(\psi_{E-\text{shop}} \wedge (\text{Paypal} = \text{true} \wedge \text{SMS} = \text{true} \wedge \text{card} = \text{true}))\}$.

Operation 7. (Dead Nodes). Let M be an Extended OVM, ψ_M the equivalent CSP of the form (V, D, C) , and n a variant or a variation point $\in V$. The possible dead nodes, hereinafter *DeadNodes*, is the set of n_i nodes such that $\text{isDeadNode}(n_i) = \text{true}$. The node n is dead if there is no solution S for the equivalent CSP ψ_M , such that n belongs to S .

$$\begin{aligned} isDead(M, n) &\Leftrightarrow \nexists S \in sol(\psi_M) | n \in S. \\ DeadNodes(M) &= \{N | \forall n_i \cdot n_i \in V \Rightarrow \\ isDeadNode(M, n_i) = true\}. \end{aligned}$$

Operation 8. (Optimization). Let M be an Extended OVM, and O an objective function. Optimization is the set of solutions that minimize or maximize O . Thus, $min(M, O) = min(\psi_M, O)$ is the set of solutions that minimize O and $max(M, O) = max(\psi_M, O)$ is the set of solutions that maximize O .

5 Conclusion and future work

The extra-functional aspects of a software product line are important information which should be dealt by the variability modelling techniques. In this paper we have presented how Orthogonal Variability Models (OVMs) can also represent extra-functional information, by adding attributes to these models. Our main goal is to provide automated support to the analysis of OVMs. The automated analysis of OVMs has been hardly explored and there is no tooling support for this purpose. To achieve our goal, we have applied the concepts of analysis operations, techniques and tools used in the context of automated analysis of feature models. We proposed the use of constraint programming for the automated analyses of OVMs, its declarativity eases the formalization of every analysis operation. The availability of off-the-shelf solvers allows to build tools which offer support for every operation described in this paper.

Currently we are working on a proof of concept to demonstrate the feasibility of our proposal. We are extending FaMa framework [18], which is a framework to analyse feature models, to support the analysis of OVMs. We are implementing all the analysis operations presented in this paper with a Java CSP solver called Choco [1]. A prototype is available at <http://www.lsi.us.es/~dbc/material/jisbd10/>, there you also can find the results of the operation *all possible configurations* performed on the OVM model in Figure 1. The results

presented in this paper were generated with this prototype.

In our future work, we intend to provide a tooling support for the analysis of OVMs. In addition, based on [15] we plan to apply metamorphic testing for the automated generation of test data for the analyses of OVMs.

6 Acknowledgements.

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

References

- [1] Choco solver. <http://choco.emn.fr/>, accessed April 2010.
- [2] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.
- [3] D. Batory. Feature models, grammars, and propositional formulas. In *Software Product Lines Conference*, volume 3714 of *LNCS*, pages 7–20. Springer-Verlag, 2005.
- [4] D. Batory, D. Benavides, and A. Ruiz-Cortés. Automated analysis of feature models: Challenges ahead. *Communications of the ACM*, December:45–47, 2006.
- [5] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Using constraint programming to reason on feature models. In *The Seventeenth Int. Conf. on Software Engineering and Knowledge Engineering, SEKE 2005*, pages 677–682, 2005.
- [6] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: a literature review. *Information Systems*, in press.

- [7] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In *17th Int. Conf. Advanced Information Systems Engineering, CAiSE 2005*, volume 3520 of *LNCS*, pages 491–503. Springer-Verlag, 2005.
- [8] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series en Software Engineering. Addison-Wesley, August 2001.
- [9] O. Djebbi, C. Salinesi, and D. Diaz. Deriving product line requirements: the red-pl guidance approach. *Asia-Pacific Software Engineering Conference*, 0:494–501, 2007.
- [10] C. Lianpingn, A. B. Muhammad, and A. Nour. Variability management in software product lines: A systematic review. In *13th Int. Software Product Line Conf., SPLC'09*, San Francisco, USA, 2009.
- [11] 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 *Requirements Engineering Conference, 2007. 15th IEEE Int.*, pages 243–253, 2007.
- [12] K. Petersen, J. M. Zaha, and A. Metzger. Variability-driven selection of services for service compositions. In *ICSOC Workshops*, pages 388–400, 2007.
- [13] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.
- [14] F. Roos-Frantz and S. Segura. Automated analysis of orthogonal variability models. a first step. In *Workshop on Analyses of Software Product Lines*, pages 243–248, 2008.
- [15] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cortés. Automated test data generation on the analyses of feature models: A metamorphic testing approach. In *Third International Conference on Software Testing, Verification and Validation.*, Paris, France, 2010. IEEE press.
- [16] M. Sinnema and S. Deelstra. Classifying variability modeling techniques. *Information & Software Technology*, 49(7):717–739, 2007.
- [17] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, and M. Toro. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software*, 81(6):883–896, Jun 2008.
- [18] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez. Fama framework. In *Software Product Line Conf. Tool Demonstrations*, 2008.
- [19] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1995.
- [20] J. White, D. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated diagnosis of product-line configuration errors in feature models. In *12th. Software Product Line Conference (SPLC)*, pages 225–234, Limerick, Ireland, Sep 2008. IEEE.