Configurable feature models

Pablo Trinidad*, Antonio Ruiz-Cortés, and Jesús García-Galán

Universidad de Sevilla
{ptrinidad,aruiz,jegalan}@us.es
http://www.isa.us.es

Abstract. Feature models represent all the products that can be built under a variability-intensive system such as a software product line, but they are not fully configurable. There exist no explicit effort in defining configuration models that enable making decisions on attributes and cardinalities in feature models that use these artefacts. In this paper we present configurable feature models as an evolution from feature models that integrate configuration models within, improving the configurability of variability-intensive systems.

Keywords: software product lines, configurable feature models, configuration, model

1 Introduction

Since the set of products that can be manufactured in a Software Product Line (SPL) can be huge, with thousands of products [4, 7] it is necessary to have models that make it possible both to represent the complete set of products in a compact manner and to enable its systematic and automated management. Feature Models (FMs) [3] are one of the most widely used models for this purpose, proposing a compact representation of all the products in an SPL in terms of their features. A feature is a user-visible aspect or characteristic of the domain. Features are connected by means of relationships among them, forming a tree-like structure. Relationships constrain the way in which features can be combined. Besides features, FMs might use cardinalities to group features in the so-called Cardinality-Based Feature Models (CBFMs) and/or attributes to remark non-functional characteristics of products in the so-called Extended Feature Models (EFMs).

The process by which one or more users define the product that best fits their needs by making successive decisions on a particular FM is called *configuration process* [5]. This process is successfully accomplished when a threefold condition fulfils: there are no more user decisions to make, there exists only one product

* This work has been partially supported by the European Commission (FEDER) and Spanish Government under TAPAS project (TIN2012-32273) and IPT-2012-0890-3 (SaaS-FireWall) and the Andalusian Government under COPAS (P12-TIC-1867) and THEOS (TIC-5906) projects

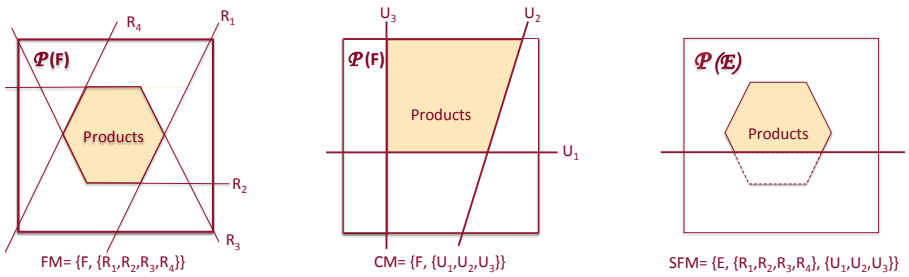
in the SPL satisfying each and every user decision, and there are no contradictions among decisions. The decisions made by users in a configuration process are collected by the so-called Configuration Model (CM), or simply a *configuration*. CMs have not been specifically studied in SPLs, being widely accepted the three-set model that represents user decisions in terms of selected, removed and undecided features. We have found three limitations in CMs that we aim to save in this paper.

First, despite cardinalities and attributes are used in Extended and Cardinality-Based Feature Models (ECBFMs) to represent relevant information, current CMs are unable to represent decisions on them, only enabling the decision making on features. It impedes a user making decisions about attributes such as 'I want a Smart Home System (SHS) that costs less than 2.000 €', or about cardinalities such as 'I want a SHS with two Internet connections'. We affirm that current FMs are not *fully-configurable* since CMs are unable to make decisions on any element of the FM.

Second, a strong relationship between elements in the FM and in the CM arises when attributes and cardinalities come into play. A CM must ensure that a user decision on an attribute is made within the attribute domain, which is specified in the FM. The same happens for cardinalities, whose valid values for user decisions are modelled in the FM. FMs and CMs share elements and the values those elements can take in a user decision. There are two main approaches to deal with information sharing: keeping two separate models in constant synchronisation, communicating any change in the FM to CMs, or combining FMs and CMs in a unique model around the common elements.

Third, some authors [2, 9] have used annotations on FMs to depict decisions on features. But there is no graphical representation that supports attributes and cardinalities.

From these problems we consider that it is possible to extend CMs to enable fully-configurable ECBFMs. We propose Configurable Feature Models (CFMs) [8] as a kind of model that *(i)* combines FMs and CMs in a unique model, enabling the representation of user decisions on attributes and cardinalities; *(ii)* takes advantage of the information that FMs and CMs share; and *(iii)* represents CMs together with FMs in a backwards-compatible graphical notation. In Section 2, we interpret FMs and CMs as a set of constraints on the same set of elements. In Section 3 we describe the main concepts in CFMs. In Section 4 we propose an abstract model for CFMs. Two different representations of CFMs are presented, each of them for a different purpose. In Section 5, we firstly present the configurable feature diagrams as a graphical representation of CFMs. in Section 6, a UML configurable feature metamodel is proposed as a formalisation of CFMs that enable their transformation to other metamodels. Last, Section 7 presents the next steps to make the most of CFMs.



(a) Feature model (b) Configuration model (c) Configurable feature model

Fig. 1. A visual metaphore of FMs, CMs and CFMs

2 Rationale

A FM describes a set of products in terms of features and relationships. A product is a subset of features that satisfy all the relationships. From these definitions, we can interpret a FM in terms of the following sets. Let F be the set of features in a SPL, and let $\mathcal{P}(F)$ be the set of all the possible combination of features. The relationships can be interpreted as constraints on $\mathcal{P}(F)$. These constraints define the set of products (P) as a subset $P \subseteq \mathcal{P}(F)$.

CMs represent the user decisions in terms of three sets of selected, removed and undecided features. User decisions can be interpreted as another different way to constrain the set of all the possible combination of features $\mathcal{P}(F)$ with a different kind of constraints. Therefore FMs and CMs can be regarded as two different models that describe subsets of $\mathcal{P}(F)$.

This vision also fits into CBFMs and EFMs where cardinalities and attributes are added to the set of features. In this case, a new set that represents all the features, cardinalities and attributes in an ECBFM is necessary. This set is denoted as the set of elements E. In this case, relationships can be interpreted as constraints on $\mathcal{P}(E)$, i.e. the space of all the possible combinations of elements. Symmetrically, user decisions in CMs could also be interpreted as constraint on $\mathcal{P}(E)$ if they were fully-configurable, but current CMs keep on interpreting the CM as a subset of $\mathcal{P}(F)$.

In order to enable fully-configurable FMs, we propose extending CMs to support cardinalities and attributes besides features. The resulting models must also satisfy three main requirements in order to provide fully-configurable FMs: *(i)* user decisions can only refer to elements in E so it must avoid any reference to invalid cardinalities or attribute values; *(ii)* CMs must allow multiple users making decisions at the same time, and *(iii)* CMs must distinguish between user and automatic decisions.

As a solution, we propose incorporating FMs and CMs together in a single model that we have coined as Configurable Feature Model (CFM). The resulting model stores together the set of elements E, and two sets of constraints, one for

relationships and another one for user decisions (see Figure 1). A CFM supports multiuser configurations and distinguishes between user and automatic decisions. CFMs are the first fully-configurable FMs, able to represent user decisions on attributes and cardinalities. Table 1 compares the capability of dealing with features, cardinalities and attributes in CFMs, basic FMs, CBFMs, EFMs and ECBFMs. In next Section we propose an abstract model for CFMs and the relevant concepts that arise from their use.

Table 1. A comparison of the use of elements in different kinds of feature models

Kind of constraint	Basic FM	CBFM	EFM	ECBFM	CFM
Relationship	F	F, C	F, A	F, C, A	F, C, A
User decisions	F	F	F	F	F, C, A

F = Features, C = Cardinals, A = Attributes

3 Main concepts

A CFM contains information about the elements in a SPL, which are features, cardinalities and attributes, hitherto stored in FMs. Depending on the decisions a user can make on elements, each element has its own set of states. So features have a selected or removed state; Cardinalities have as many states as cardinals in its range; Attributes have as many states as values they can take. An assignment of states for every element in the CFM defines the characteristics of what is called a *potential product*.

To determine which of the potential products can be built within the SPL, a set of *relationships* constrains the valid combinations of states, defining what is called the set of *SPL products*. A CFM also collects all user decisions, which are constraints that define a subset of potential products named *configuration*. The set of SPL products that are also found in the configuration is called the set of *compatible products*, and brings together the SPL products that meet all user decisions.

CFMs distinguish among user and automatic decisions. An automatic decisions is the one that is made by an Automated Analysis of Feature Models (AAFM) operation to support the configuration process. An automatic decision can be defined as the one that if a user makes it, the set of compatible products remains unaltered. They are computed and stored separately from user decisions unless a user explicitly makes such a decision.

Figure 2 proposes a geometrical interpretation of the concepts presented in this Section.

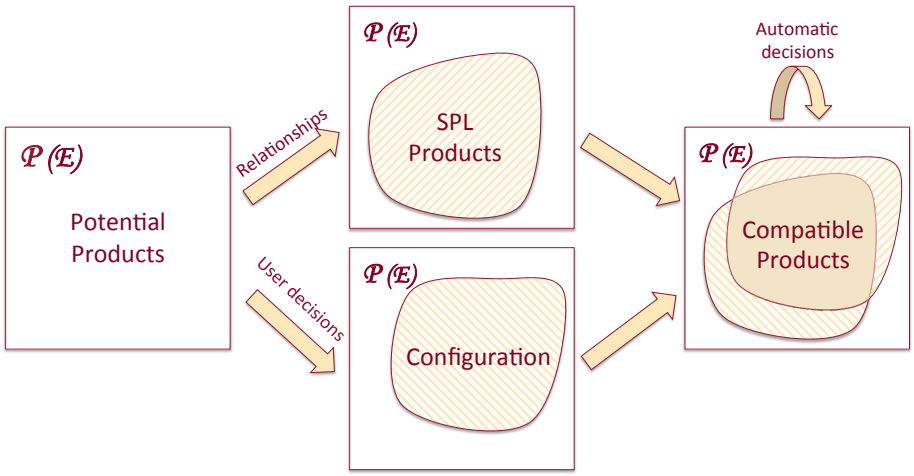


Fig. 2. A geometrical interpretation of basic concepts in CFMs

4 Abstract model for CFMs

In this Section we give a transformational semantics to CFMs in terms of set theory, relying on a new vision of products as an assignment of states to elements. In FMs, each product is described as a different subset of features. In CFMs, all the products share the same set of elements, denoted by a non-empty set $E = \{E_1, ..., E_n\}$ where each E_i is an element in that model, either features, cardinalities or attributes. A product is defined as an assignment of states to every element in E such that each element has a set of available states that depends on the kind of element. This way, features have selected or removed states to indicate their presence or absence in a product; a cardinality has a cardinal value as state to indicate the number of features that are in a selected state within a set relationship; attributes have a state which corresponds to a value that represents the quality or behaviour of their linked features.

To define which states every element can have, a CFM has a set of available state sets $AS_1, ..., AS_n$ such that AS_j is the set of available states for an element E_j. Let us consider an SPL with 6 elements $E = \{F_R, F_A, F_B, F_C, C_1, Mem\}$ as an example. F_R is the root feature; F_A, F_B and F_C are child features; C_1 is a cardinal affecting the 3 child features; Mem is an attribute linked to the root feature. The available states for each elements could be: $AS_{F_R} = \{sel\}$, $AS_{F_A} = AS_{F_B} = AS_{F_C} = \{sel, rem\}$, $AS_{C_1} = \{1, 2, 3\}$, $AS_{Mem} = \{128, 256\}$. This way, the root feature F_R must have a selected state, features F_A, F_B and F_C can be either in a selected or removed state, C_1 cardinality, which affects features F_A, F_B and F_C, must be either 1, 2 or 3 and the product can have a 128 or 256 kilobytes memory size. We define the set of potential products as follows:

Definition 1 (Set of potential products). *Let AS_j be the set of available states for the j^{th} element $e_j \in E$. The set of all the potential products in a CFM is defined by the cartesian product $D = AS_1 \times ... \times AS_n$. A* potential product *corresponds to any tuple $(s_1, ..., s_n) \in D$.*

In the previous example, the set of potential products D is comprised of 32 different potential products. $S_1 = (sel, sel, sel, sel, 2, 256)$ and $S_2 = (sel, rem, sel, rem, 1, 128)$ are examples of potential products. However, not all of them are *SPL products*, i.e. products that effectively can be built in the SPL. To define the set of all the SPL products, a first option that consists of enumerating all of them is immediately discarded when the number of SPL products shoots up. A set in general can be defined by a list of its elements or by a subset that satisfies a condition or constraint. CFMs use constraints to define the set of SPL products as a subset of the set D of potential products. A CFM has a set of *relationships* $R = \{R_1, ..., R_i\}$ that establishes the conditions a potential product must necessarily fulfill to be an SPL product. So the set of SPL products is defined as follows:

Definition 2 (Set of SPL products). *Let $R = \{R_1, ..., R_i\}$ be the set of relationships in a CFM and D its set of potential products. The set of SPL products defined by the CFM is:*

$$P = \{(s_1, ..., s_n) \in D | R_1 \wedge ... \wedge R_i\}$$

It might be the case that there exist no product satisfying all the relationships and therefore $P = \emptyset$. In this case, it is said that the CFM is *void* or *invalid*.

The kinds of constraint that we propose for CFMs are the same than those used for FMs. Table 2 shows a list of constraints that can be used to represent the relationships in a CFM. So for example, if `R` is a parent feature linked by a set relationship with three child features `A`, `B` and `C` and affected by cardinality C_1, the following constraint and set of SPL products can be defined:

$$P = \{(s_R, s_A, s_B, s_C, s_{C_1}, s_{Mem}) | set_3(s_R, s_{C_1}, s_A, s_B, s_C)\}$$

A CFM also stores a configuration which comprises all the decisions made by one or more users in a given moment. Decisions are collected in any order, supporting a parallel configuration process. These user decisions set a partition on the set of potential products D in a SPL: those that satisfy user decisions and those that do not. Following this criterion, we define a configuration as follows:

Definition 3 (Configuration). *Let $U = \{U_1, ..., U_j\}$ be a set of constraints defined on D, describing the user decisions in a CFM. A configuration C_U is defined as the set of potential products that satisfy all the user decisions:*

$$C_U = \{(s_1, ..., s_n) \in D | U_1 \wedge ... \wedge U_j\}$$

Table 2. Kinds of relationship constraints in a CFM

$mandatory(s_p, s_c)$	$\equiv s_p = s_c$
$optional(s_p, s_c)$	$\equiv s_c = sel \Rightarrow s_p = sel \wedge s_p = rem \Rightarrow s_c = rem$
$set_2(s_p, s_c, s_{c_1}, s_{c_2})^\dagger$	$\equiv s_p = sel \Leftrightarrow numSelChild_2(s_c, s_{c_1}, s_{c_2}) \wedge$ $s_p = rem \Rightarrow s_{c_1} = rem \wedge s_{c_2} = rem$
$set_3(s_p, s_c, s_{c_1}, s_{c_2}, s_{c_3})^\dagger$	$\equiv s_p = sel \Leftrightarrow numSelChild_3(s_c, s_{c_1}, s_{c_2}, s_{c_3}) \wedge$ $s_p = rem \Rightarrow s_{c_1} = rem \wedge s_{c_2} = rem \wedge s_{c_3} = rem$
...	...
$depends(s_1, s_2)$	$\equiv s_1 = sel \Rightarrow s_2 = sel$
$excludes(s_1, s_2)$	$\equiv \neg(s_1 = sel \wedge s_2 = sel)$

$^\dagger numSelChild_i(s_c, s_{c_1}, ..., s_{c_n})$ is a predicate that is true whenever the number of selected states in $s_{c_1}, ..., s_{c_n}$ coincides the cardinal s_c.

Users can make two kinds of decisions as shown in Table 3. A user can either make a *choose decision*, that assigns a state for one element, or a *discard decision* that avoids an element having a certain state. So for example, $U_1 = choose(s_B, sel)$ identifies a user selecting feature B; $U_2 = discard(s_{Mem}, 128)$ identifies a user refusing a 128Kb bandwidth. With these two kinds of decisions, users delimit the set of potential products by means of successive refinements.

In case that user decisions contradict each other, as two users choosing selected and removed states for the same feature for example, then the set $C_U = \emptyset$. This situation is known as a *contradictory configuration*.

Table 3. Kinds of decision constraints in a CFM

$choose(s_e, S) \equiv s_e = S$
$discard(s_e, S) \equiv s_e <> S$

From the configuration and the set of SPL products, it can be determined the set of *compatible products*, i.e. the subset of SPL products that satisfy the criteria established by user decisions. The set of compatible products is defined as follows:

Definition 4 (Set of compatible products). *Let P be the set of SPL products defined by the relationship constraints $R_1, ..., R_i$, and let C_U be a configuration defined by the user decision constraints $U_1, ..., U_j$. The set of compatible products (CP) is defined as:*

$$CP = P \cap C_U = \{(s_1, ..., s_n) \in D | R_1 \wedge ... \wedge R_i \wedge U_1 \wedge ... \wedge U_j\}$$

With the above definitions, we are able to represent a CFM in a compact manner as follows:

Definition 5 (Configurable Feature Model). *A CFM can be represented by a 4- tuple* (E, D, R, U) *such that* E *is the set of elements in the CFM,* D *is the set of potential products,* R *is the set of relationships and* U *is the set of user decisions.*

From the information contained in this tuple, the set of potential, SPL and compatible products can be deduced. Next we introduce three concepts that can be defined on top of CFMs: CFM states, element states and automatic decisions.

4.1 CFM states

From the set of compatible products (CP), it is possible to identify singular situations or *CFM states*:

- Initial state: a CFM is in its initial state when there exist no user decisions, i.e. $U = \emptyset$. In this case, the configuration coincides with the space of potential states ($C_U = D$) and therefore the set of compatible products coincides the set of products, i.e. $CP = P$.
- Final state: a CFM is in a final state when it only determines one compatible product, i.e. $|CP| = 1$. When a final state is reached, it can be affirmed that a product satisfying all the user decisions has been found. The final adjective comes from the inability of the CP set to evolve in a manner that the set of compatible products is reduced even more.
- Intermediate state: A state is intermediate if $U \neq \emptyset$ and $|CP| > 1$. This is the most common state in a CFM and it corresponds to a situation in which users may still made decisions.
- Invalid state: if $CP = \emptyset$ at any moment, a CFM is said to reach an invalid state. This state can be reached because the set of CFM products is empty ($P = \emptyset$), or the configuration is contradictory ($C_U = \emptyset$), or there exist no compatible product satisfying the user decisions ($P \cap C_U = \emptyset$).

A CFM state makes reference to the instant in which the configuration process is. This way, the configuration process starts with a CFM in its initial state. User decisions provoke the CFM state to change to an intermediate state. A final state can be reached if there exist only one product satisfying all the user decisions. In case a configuration defines no compatible product, an invalid state is reached.

4.2 Element states and automatic decisions

From a CFM, it is possible to extract relevant information about the elements and states on which users might still make decisions without reaching an invalid state. For this purpose, we define the concept of *element state* as the set of all the states an element has for every compatible product in CP.

Definition 6 (Element State). *Let E_i be an element in a CFM and CP the set of compatible products. The element state for that element E_i is defined as follows:*

$$State(E_i) = \bigcup_{s \in CP} \pi_i(s)$$

Being $\pi_i(s)$ the projection function that extracts the i^{th} element from a tuple s that represents a compatible product.

So for example, let us consider the following set of compatible products:

$$\begin{aligned} E &= \{F_R, F_A, F_B, F_C, C_1, Mem\} \\ CP &= \{P_1, P_2, P_3\} \\ P_1 &= \{sel, rem, sel, sel, 2, 256\} \\ P_2 &= \{sel, rem, rem, sel, 1, 128\} \\ P_3 &= \{sel, rem, sel, sel, 2, 128\} \end{aligned}$$

The following element states are obtained from these compatible products:

$$\begin{aligned} &State(F_R) = \{sel\},\ State(F_A) = \{rem\},\ State(F_B) = \{sel, rem\}, \\ &State(F_C) = \{sel\},\ State(C_1) = \{1, 2, \},\ State(Mem) = \{128, 256\} \end{aligned}$$

An element state is used to infer what we call the set of *automatic decisions* (A). An automatic decision is the one that if it is made by a user, the set of compatible products remains unaltered. This way, if $A = \{A_1, ..., A_k\}$ represents the set of automatic decisions, and being C_A the subset of all the potential states that satisfy such automatic decisions:

$$C_A = \{(s_1, ..., s_n) \in D | A_1 \wedge ... \wedge A_i\}$$

it is possible to affirm that

$$CP \cap C_A = CP \Rightarrow CP \subseteq C_A$$

This set of automatic decisions is used to assist users in the decision making process, avoiding as far as possible to reach for invalid states. Automatic decisions can be calculated from the element states. Let E_i be an element in a CFM having AS_i as available states and $State(E_i)$ as its corresponding element state. Depending on the element state, an automatic decision can or cannot be inferred as follows:

- $State(E_i) = AS_i$: in this case, the element is said to be in an *undecided* state. It means that no user has made any decision that affects it and no automatic decision can be made on it.
- $|State(E_i)| = 1$: in this case, it can be interpreted that an automatic decision has been made on E_i to choose the unique state in $State(E_i)$
- $|AS_i| > |State(E_i)| > 1$: in this case, it can be interpreted that all the states that are not in $State(E_i)$ have been discarded. It means that those states in the set $AS_i - State(E_i)$ can be automatically discarded.

– $State(E_i) = \emptyset$: this situation can only be given if $CP = \emptyset$ in which case it does not make sense to calculate any automatic decision since there is no compatible product.

Each automatic decisions obtained following the above criteria must be checked if it already exists as a user decision, in which case they are disposed. For the previous example, the following set of automatic decisions can be inferred:

$$A = \{choose(s_A, sel), choose(s_C, sel), discard(s_{C_1}, 3)\}$$

5 Configurable feature diagrams

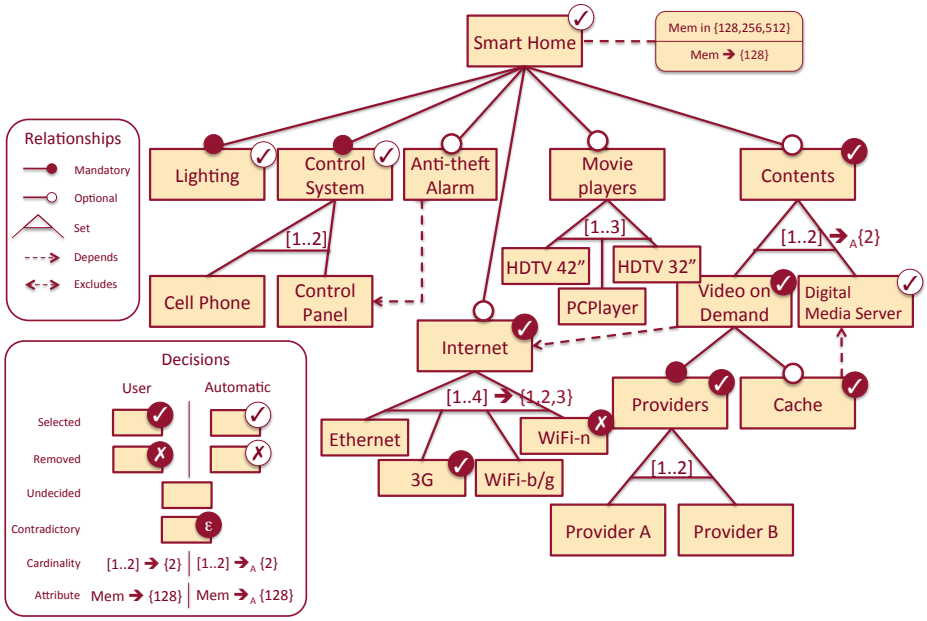


Fig. 3. An example of a Configurable Feature Diagram

Configurable feature diagrams are a graphical notation of CFMs based on feature diagrams [6]. The representation of elements and relationship constraints is the same that FMs: a tree-like structure where features are boxes linked by different kinds of lines that represent the relationships among features. Cardinalities are also drawn together with the corresponding set relationship.

User and automatic decisions are not explicitly depicted in the diagram. Instead of it, element states are drawn. If a feature state is $\{sel\}$, it is drawn as a tick (✓) within a circle in a corner of the feature box; if a feature state is

$\{rem\}$, a cross (✗) is used instead; if a feature is undecided, no specific mark is assigned; if two or more users have made contradictory decisions on a feature, it is marked as ε. It the feature state is inferred by user decisions, then a filled circle is used; if it can only be inferred by automatic decisions, a non-filled circle is used instead.

Decisions on cardinalities and attributes are drawn as $E_d \rightarrow E_s$ where E_d is the cardinality or attribute domain and E_s is the element state for that cardinality or attribute. For example a cardinality [1..3] whose state has been set to $\{2\}$ is drawn as $[1..3] \rightarrow \{2\}$. If state 2 is removed but no other state has been chosen, it is represented as $[1..3] \rightarrow \{1, 3\}$. Likewise, memory attribute (Mem) has three available states $AS_{Mem} = \{128, 256, 512\}$. If its state is set to 128 Mb, it is drawn as $Mem \rightarrow \{128\}$. If a user discards needing 512 Mb, then it is depicted as $Mem \rightarrow \{128, 256\}$. Just in case any state is inferred only from automatic decisions, $E_d \rightarrow_A E_s$ is used to remark that a state has changed due to automatic decisions. Figure 3 shows an example of a configurable feature diagram.

6 Configurable feature metamodel

A metamodel is a rigorous definition of a model that describes their main concepts, relationships and rules. We propose the Configurable Feature Metamodel (CFMM) as a rigorous definition of CFMs. Besides rigour, metamodels can be used as development artefacts in Model-driven Engineering (MDE). It enables to build analysis tools by the definition of transformations into other declarative languages which can be used to reason about them.

Obtaining a metamodel is a design exercise. So we firstly propose a list of objectives the design must satisfy according to the abstract model presented in Section 4:

Objective 1. Representing a set of elements, each of which has a set of available states.

Objective 2. Representing the set of SPL products by means of a set of relationship constraints.

Objective 3. Representing decisions as a set of constraints, distinguishing among user and automatic decisions.

Objective 4. Enabling the computation of element states according to user and automatic decisions.

Objective 5. Proposing an extensible metamodel that supports the future addition of new kinds of relationships and elements.

Figure 4 depicts our proposal of a CFMM using a UML class diagram that satisfies the above objectives. A CFM is represented by a `ConfigurableFeatureModel` instance. It is a container of `Element`, `Relationship` and `Configuration` instances. The elements in a CFM can be either `GenericFeature`, `Cardinality` or `Attribute` instances. There exists a class for each kind of relationship such as mandatory, optional, set, requires, excludes and attribute relationships. A

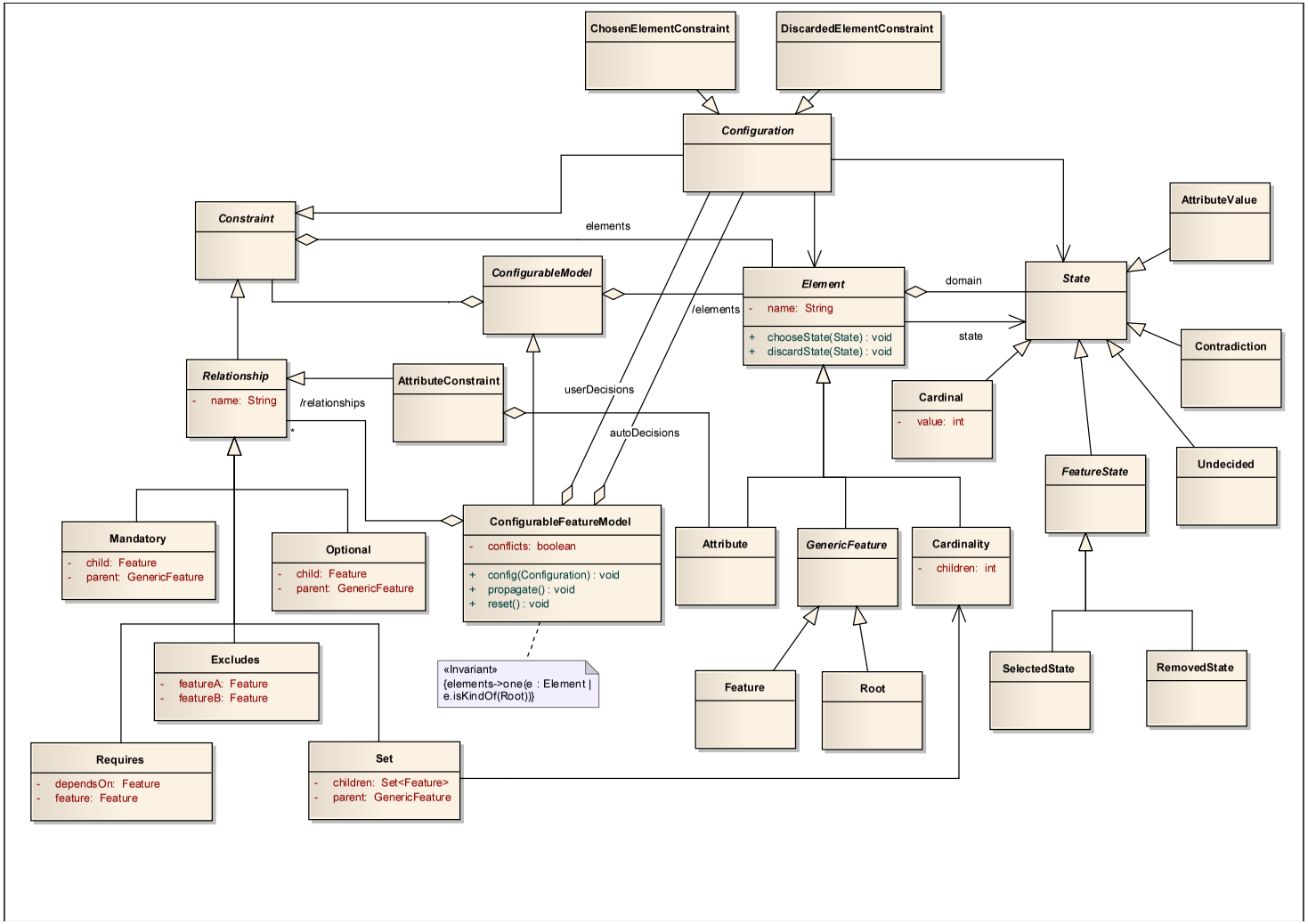


Fig. 4. Configurable Feature Metamodel in UML format

`GenericFeature` class is defined to distinguish between the root and remaining features. It is used to avoid the incorrect use of the root feature in relationships such as the root feature being the child of a relationship or an exclusion between a root and any other feature that provokes a dead feature.

The decisions can be defined in terms of an element selection (`ChosenElementConstraint` class) or discarding (`DiscardedElementConstraint` class). A CFM distinguishes between user and automatic decisions, storing them in two different aggregates. Decisions are introduced in the model by means of the model edition operations shown in the Figure within the `ConfigurableFeatureModel` class. Remaining operations, such as those to build a CFM from scratch or to modify elements and relationships have been left out of the Figure since they are not relevant for the purpose of this paper.

Every element has a domain and a current state. The domain is the set of available states. Each kind of element has its own element-specific available states: `SelectedState` and `RemovedState` for features, `Cardinal` for cardinalities and `AttributeValues` for attributes. An element state is immediately updated when a user decision is introduced into a CFM. Initially, every element state contains an `Undecided` state instance to indicate that no decision has been made on them. As decisions are added, the state of the affected elements changes to represent the decisions. In case a contradiction is found among decisions, such as a feature that is selected and removed at the same time, the `Contradiction` state instance is set as the state for the affected elements.

Last, there exist some generalisation points to increase the extensibility of the metamodel. A `ConfigurableModel` class is defined in case the constraint-element-state structure wants to be reused for other models; relationships and decisions are considered as two different kinds of `Constraint`. In case we want to add a new kind of constraint that is not covered by relationships and decisions, it can be done as an extension of this class.

7 Conclusions and further work

In this paper we have presented CFMs as a new kind of model that enables fully-configurable FMs. CFMs incorporate the concept of element states to describe the dynamic behaviour of CFMs which is a novel approach in the description of products in a SPL. CFMs distinguish among potential products as any combination of element states, SPL products as those potential products that satisfy the relationship contraints, and compatible products as the SPL products that satisfy the configuration constraints. They also allow to work with multiple users making decisions on a CFM at the same time and distinguishes between user and automatic decisions.

Configurable feature diagrams are proposed to graphically describe CFMs. They are inspired in feature diagrams where states are added to represent user and automatic decisions.

Besides the rigorous definitions given in this paper for CFMs, we define them in terms of an UML metamodel. Metamodels enable their use in MDE tools. The

CFMM takes extensibility as a major concern. The extension mechanisms allow the future exploration of the impact of continuous attributes which have been left out of this proposal. The metamodel presented in this paper will play a key role in the implementation of analysis operations that enable the extraction of relevant information from CFMs such as counting products, validating a configuration, detecting and explaining errors, explaining invalid configurations, etc. Our future work will consist in the proposal of a catalogue of analysis operations inspired in the AAFM [1] that introduces new operations where attributes and cardinalities are first-level artifacts.

References

[1] D. Benavides, S. Segura, and A. R. Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 9 2010.

[2] C. Cetina, P. Trinidad, V. . Pelechano, and A. Ruiz-Cortés. Customisation along lifecycle of autonomic homes. In *3rd International Workshop on Dynamic Software Product Line (DSPL09)*, 2009.

[3] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Nov. 1990.

[4] F. Lösch. *Optimization of variability in software product lines: a semi-automatic method for visualization, analysis, and restructuring of variability in software product lines.* PhD thesis, University of Stuttgart, 2008.

[5] R. Rabiser, P. Grunbacher, and D. Dhungana. Supporting product derivation by adapting and augmenting variability models. In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 141–150, Sept 2007.

[6] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps. Feature Diagrams: A Survey and A Formal Semantics. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, Minneapolis, Minnesota, USA, Sept. 2006.

[7] M. Steger, C. Tischer, B. Boss, A. Müller, O. Pertler, W. Stolz, and S. Ferber. Introducing pla at bosch gasoline systems: Experiences and practices. In R. L. Nord, editor, *Software Product Lines, Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings*, volume 3154 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2004.

[8] P. Trinidad, A. Ruiz-Cortés, and D. Benavides. *Automated Analysis of Stateful Feature Models*, chapter 30, pages 375–380. Springer Berlin Heidelberg, 2013.

[9] J. White, D. Benavides, D. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes. Automated diagnosis of feature model configurations. *Journal of Systems and Software*, 83(7):1094 – 1107, 2010.