Criteria for the verification of feature models

Camille Salinesi[*] — Raúl Mazo[*,] — Daniel Diaz[*]**

** CRI, Université Paris 1 - Panthéon Sorbonne*
90, rue de Tolbiac
75013 Paris, France

{Camille.Salinesi, Daniel.Diaz}@univ-paris1.fr

*** Ingeniería de Sistemas, Universidad de Antioquia*
Medellín, Colombia

raulmazo@gmail.com

RÉSUMÉ. L'Ingénierie de lignes de produits est une approche pour le développement de systèmes intensifs. L'expérience a montré les bénéfices de cette approche dans la réduction du temps pour la misse en marché, la réutilisation et la réduction du coût de développement. Les langages de modélisation, en particulier pour la création de modèles de caractéristiques et processus de configurations sont actuellement supportés par quelques outils existants sur le marché. Néanmoins, il manque des travaux de recherche sur les méthodes, techniques et outils de vérification de modèles de caractéristiques. Aussi, il est crucial que la vérification soit faite avec de bons critères car toute erreur dans le modèle de ligne de produits se propage sur les modèles de produits dérivés de la ligne et génère des problèmes d'instabilité dans l'architecture. Cet article présente un travail original concernant une revue de la littérature sur les critères de verification de modèles de lignes de produits. Les critères sont (i) classifiés par rapport aux buts qu'ils représentent et (ii) formalisés d'une manière consistante en utilisant la logique de première ordre.

ABSTRACT. Product Line (PL) based development is a promising approach to develop software intensive systems. Experience already report multiple benefits, such as reduced time to market, better reuse, and reduced development costs. PL modelling languages, in particular to create feature models (FMs), and PL configuration processes are now supported by market tools. Although there is a wealth of research works on the theme of FM verification, there is to our knowledge no comprehensive method, technique or tool. However, it is crucial that when verifying a FM, the right criteria are considered: any error in a FM will inevitably spread to the configured software and generate PL architecture stability issues, with a serious risk of undermining the expected benefits. Dealing with key issues such as selecting the 'right' set of verification criteria or defining a small core of criteria from which all other could be derived calls for a consistent definition of all the criteria. This paper presents an original literature survey of FM verification criteria in which all the criteria are (i) classified according to their purpose and (ii) formalized consistently using first order logic.

MOTS-CLÉS : Modèles de lignes de produits, vérification, logique de premier ordre.

KEYWORDS: Product line model, verification, first order logic.

1. Introduction

Product line engineering is an emerging reuse based development approach that is already known for allowing companies realize important improvements on time to market, cost, productivity, quality and flexibility (SEI, 2010). In this approach a family of products is specified using a Product Line Model (PLM), and each product is specified by a product model that reuses elements from the PLM. Specifying PLMs is called domain engineering, while specifying configuration models is referred to as application engineering. The transition from domain to application is achieved through a 'configuration' process that somehow consists in adapting a PLM to specify a product that satisfies some requirement. Domain engineering is particularly challenging because PLMs handle variability to imply (sometimes large) collections of product models. One example of this difficulty is during the optimization of a PLM. In this activity, goal functions involve multiple products which, although they are implicit, need to be optimized too (Benavides *et al.,* 2006).

This paper is interested in the problem of verifying PLMs. The difficulty in PLM verification results from the fact that the semantics of the model is represented by the set of implicit product models that can be generated from it. Any error in a PLM can affect product models, or the ability to specify the right products from it. For example, the introduction of inadequate dependencies in the PLM can create inconsistencies that forbid the configuration of products that should on the contrary be permitted. Another example is when the PLM is poorly constrained and product configurations that should not exist are still represented in the PLM.

One way to verify PLM is through manual checking. However, manual checking is laborious and error-prone, especially in large and complex models. We therefore believe that PLM verification should aim at avoiding errors both in PLMs and in the resulting product models. By verification of a PLM (Bjorner, 2006) we mean the formal process of determining whether or not a PLM satisfies a set of well defined criteria. Literature review shows that several methods, techniques and tools have already been proposed for the verification of PLMs, especially feature-oriented modelling notations (Benavides *et al.,* 2006), (von der Massen *et al.,* 2004). One observation is that although PLM verification criteria are often implemented using a SAT tool, they are not systematically specified. Another observation is that while there has been a focus of some approaches on the detection of so called 'dead features' and 'full-mandatory features' in Feature Models (FM), other criteria have also been proposed; twelve criteria are for instance identified in (von der Massen *et al.,* 2004). We collected a list of 15 verification criteria, formalized them in a consistent way by means of First Order Logic (FOL) expressions, and classified them according to their purpose. We have chosen FOL as our verification criteria representation formalism because: (1) FOL provides a uniform way of specifying the criteria. We consider that the formalized criteria are easy to adapt and reuse for other languages than cardinality-based feature models. (2) Criteria are specified in a natural way and therefore formulate the invariants that shall be respected. (3) The collection of criteria can be augmented without altering existing criteria. (4) They can be automatically implemented using an off-the-shelf satisfiability solver tool.

The remainder of this paper is structured as follows. Section 2 presents a formal notation of feature based PL modelling languages. Section 3 presents our classification of all the criteria identified in literature and also defined by us. Each criterion classified in section 3 is formally specified in section 4 using first order logic and respecting the notation presented in section 2. Section 5 present some related works and discusses which criteria can be used for which feature-oriented modelling dialect, other PLM languages, and other variability models. Section 6 concludes the paper and describes future works.

2. Reference feature meta model

There are a very large number of features notations to model product lines (Czarnecki *et al.*, 2005), (Gurp *et al.,* 2001), (Streitferdt, 2003). The most well known feature notation is FODA (Kang *et al.*, 1990) the others are improvements to FODA notation. As each has specific characteristics, we have decided to consider the three most known dialects in this paper (a) the cardinality-based feature notation that was proposed in Czarnecki *et al* (Czarnecki *et al.*, 2005), (b) FORE (Family Oriented Requirements Engineering) (Streitferdt, 2003) and (c) Bosch's notation (Gurp *et al.*, 2001).

Figure 1 presents a meta-model of the 3 FODA dialects that we consider. The meta-model shows that a PLM is composed of features (some with cardinalities) and relationships between a source and a target feature. Two types of cardinalities are represented in the meta-model, feature and group cardinalities. A feature Cardinality indicates the number of times a single feature can appear in a product is a composition of several optional relationships sharing the same father. The Group Cardinality indicates the minimum and maximum number of features that can be chosen together in a single product. The aim of the meta-model is to define all concepts that will be used in predicates that we use in the formalization of each verification criterion.

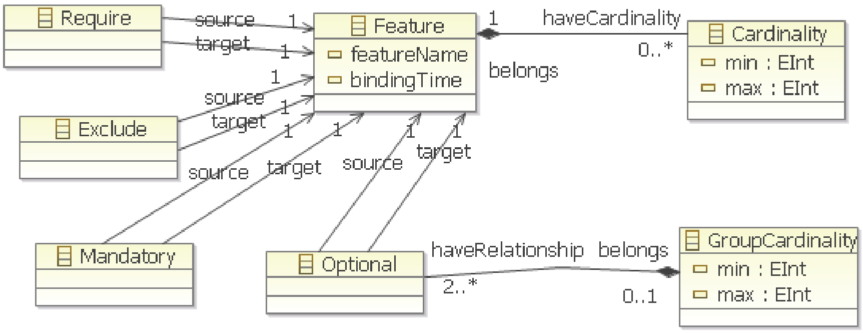
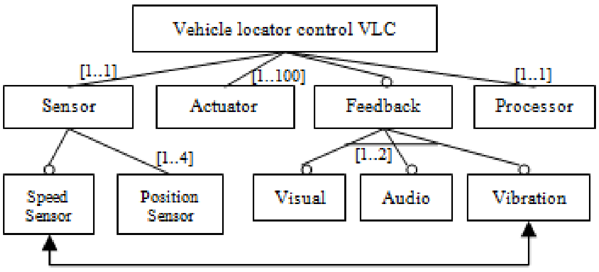


Figure 1. *Meta-model for cardinality-based feature models.*

Figure 2 provides an example of PLM specified using the cardinality-based feature notation depicted in *Figure 1*. Model of *Figure 2* is a directed acyclic graph based on a tree where nodes represent features and edges represent variation dependencies. Features specified in the graph can for instance, describe a cohesive, identifiable unit of system functionality (Turner *et al.*, 1999). In this model, optional dependencies are represented with an empty circle at the end (For example Speed Sensor, Feed Back, Visual, Audio and Vibration). Two other kinds of transverse dependencies can be set between any feature in the tree to specify exclusion and requirement constraints (For example Speed Sensor *excludes* Vibration). Visually, a feature set is shown by an arc connecting all the edges that are part of it. In the example of *Figure 1*, features Visual, Audio and Vibration are a feature set whit [1..2] as group cardinality.



Supplementary constraint:
If Processor's performance < 1.5 GHz then only two sensors could be chosen at maximum.

Figure 2. *Extract of a VLC product line model using cardinality–based feature notation.*

3. Classification of verification criteria for feature models

A PLM can have many anomalies. We have conducted a large survey based on literature review (Batory, 2005), (Benavides *et al.*, 2005), (Czarnecki *et al.*, 2005), (Czarnecki *et al.*, 2006), (Elfaki *et al.*, 2009), (Janota *et al.*, 2007), (Salinesi *et al.*, 2009), (Trinidad *et al.*, 2008), (van den Broek *et al.*, 2009), (von der Massen *et al.*, 2004), (Wang *et al.*, 2005), (Zhang *et al.*, 2004). Our survey showed us that:

(a) Each anomaly can be searched for using a given criterion. The literature review showed that some verification criteria are more related to expected qualities of the PLMs (for example expressiveness), while on the other hand there are some errors for which no criterion exists at all in the literature. Redundancy is an example of error for which no verification criterion exists (at least to our knowledge).

(b) Certain criteria are related to semantic anomalies detection in the PLM (for example, the no existence of dead features in the PLM, a dead feature is a feature that can never be chosen), others are related to inconsistencies detection (for example, the no existence of full-mandatory features requiring optional features. It is inconsistent because the optional feature required by the full-mandatory becomes full-mandatory also) while others are related to redundancies detection (for example, the no existence of child features requiring a relative father. It is redundant because if the child feature is selected means that all its ancestors have been selected also, then the requiring relationship is redundant).

(c) While certain criteria are oriented to verify the ability of PLM to generate all the possible products and only these ones, others are interested in quality of PLMs, independently of their semantics (i.e., the collection of possible products). The later criteria make a difference between two PLMs that generate the same products, but where one does not verify some desirable properties, such as for example the absence of any redundancy.

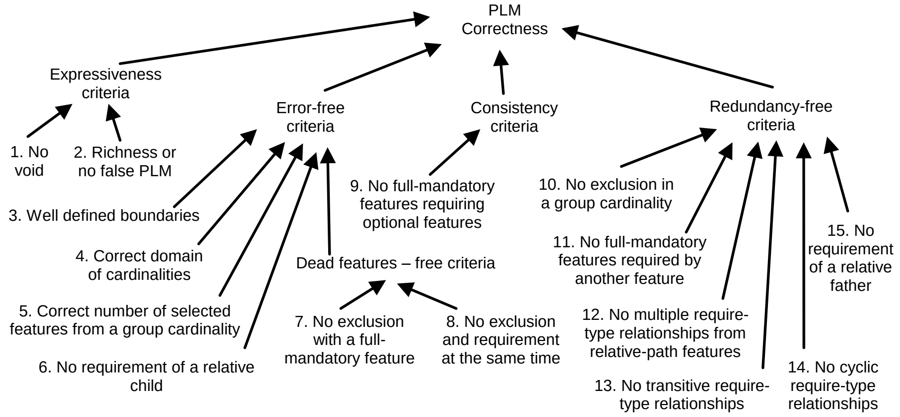

PLM Correctness
Expressiveness criteria
Error-free criteria
Consistency criteria
Redundancy-free criteria
1. No void
2. Richness or no false PLM
3. Well defined boundaries
4. Correct domain of cardinalities
5. Correct number of selected features from a group cardinality
6. No requirement of a relative child
Dead features – free criteria
7. No exclusion with a full-mandatory feature
8. No exclusion and requirement at the same time
9. No full-mandatory features requiring optional features
10. No exclusion in a group cardinality
11. No full-mandatory features required by another feature
12. No multiple require-type relationships from relative-path features
13. No transitive require-type relationships
14. No cyclic require-type relationships
15. No requirement of a relative father

Figure 3. *Classification of FMs verification criteria*.

A last remark is that not all criteria have the same level of importance: as already mentioned, some impact the semantics of PLMs, others can be used to improve PLMs without altering their semantics. We propose a classification, shown in *Figure 3*, which can be used to select the criteria that one wants to use to verify a PLM. The leaves of the classification correspond to operational criteria, *i.e.*, for which verification is unique, which can be operationalized using FOL.

Our classification, in *Figure 3*, is structured based on these considerations. The leaves are operational criteria, *i.e.*, for which there exists a unique verification, thus potentially predicative simple formalization.

4. Formalizing criteria

One thing is to identify criteria and define them in English. However, systematic and reliable verification calls for further formalization. We have chosen to formalize feature model verification criteria using first order logic because it provides a uniform way of specifying the criteria, independently of the model formalism. This section provides the formalization of criteria for verification of feature models, that is, the formalization of requirements of any future tool that intends to automate the verification of feature models. Prior to formalizing the criteria in FOL, a certain number of predicates (Osman *et al.*, 2008) must be defined:

- *optional*: identifies the relationships between a *target* feature B and it *source* A, which is specified optional(A, B).
- *mandatory*: identifies the relationships between a *target* feature B and it *source* A, which is specified mandatory(A, B).
- *max*: identifies the maximum number of features allowed to be selected in a *cardinality* relationship. *For example* max (Father Feature A, 4) indicates that the feature set which father is A has a 4 cardinality.
- *min*: Identifies the minimum number of features allowed to be selected in a *cardinality* relationship*, as in* min (Father Feature A, 1).
- *common*: this predicate has two attributes, the first one identifies a feature and the second one determines if this feature is full-mandatory or not (von der Massen *et al.*, 2004). *For example,* common (A, yes) indicates that the feature A is always selected in any configuration.
- *require*: describes an inclusion dependency between two features. For example, the constraint "if a product contains feature A it should also contain feature B" is specified: require (A, B).
- *exclude*: describes an exclusion dependency between two features (or group of features), that is the constraint "if a product contains Feature A, then it shall not contain Feature B and vice-versa" is specified: exclude (A, B).
- *count*: counts and returns the number of times that a feature A appears in the PLM. *e.g.* count (A).
- *relativePath*: returns true if a feature A is an element in the path from the root of the PLM to another feature B. This is specified relativePath (A,B).
- *featureSet*: is the collection of features that belong to a group cardinality.
- *find*: returns true if a certain number of products can be derived from a PLM, "false" elsewhere. For example find(M, 2) is true if the PLM allows to derive at least 2 products.

In the next sub-sections criteria are grouped by family, as is showed in *Figure 3*. For each criterion we present (a) an explanation and literature review; (b) the formal definition; (c) one (or several) graphical examples of errors that it allows to identify;

and eventually (d) a comment about how to implement the criterion with a constraint solver.

PLM Correctness

4.1. *Expressiveness criteria*

1. No void (Metzger *et al.*, 2007), (Trinidad *et al.*, 2008), (van den Broek *et al.*, 2009): a feature model is void if it defines no product at all. Some implement this feature by calculating the number of products that can be derived from a PLM (van den Broek *et al.*, 2009). If the number of products that can be derived from the PLM is equal to zero, the PLM is void. As this calculation is computationally difficult, actually it is sometimes even impossible (Trinidad *et al.*, 2008), we propose to formalize it the other way round, i.e. by determining if there is at least one configuration that can be generated. If the PLM is valid, a constraint solver will find the first configuration quickly, and the process can be stopped.

$$\neg find(M,1) \Rightarrow void$$

2. Richness or no false PLM: a PLM from which only one valid product can be configured is by definition invalid. In (Metzger *et al.*, 2007) and (Trinidad *et al.*, 2008), authors propose to check this criterion using functions that return all the products that can be configured from the PLM. These functions are automated by using off-the-shelf solvers, but they are computationally expensive in very large PLMs. There is however no need to look for all possible configurations to demonstrate that a PLM can be configured in at least two products. Thus, we propose to search the first two configurations to decide if the PLM is correct with respect to the false PLM criterion. This is formalized in FOL as follows:

$$\neg find(M,2) \Rightarrow false\_PLM$$

4.2 Error-free criteria

3. Well defined boundaries (Czarnecki *et al.*, 2005): the *min* value of the cardinality must be inferior to the number of features grouped in a group cardinality. The *max* value of the cardinality must be inferior or equal to the number of features grouped in a group cardinality.

$$\forall A, B : optional(A,B) \wedge \min(A,m) \Rightarrow \min(A,m) < sum(A,(B))$$

$$\forall A, B : optional(A,B) \wedge \max(A,n) \Rightarrow \max(A,n) \leq sum(A,(B))$$

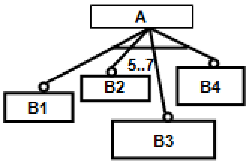


Figure 4. In this example, sum (*A*, (*B1*,*B2*,*B3*,*B4*)) = 4. Thus, the error is identified because 5 (*min*) is not inferior to 4, and 7 (*max*) is not inferior or equal to 4.

4. Correct domain of cardinalities (Czarnecki *et al.*, 2005): in a cardinality [*m*..*n*], *m* must be an Integer number and *n* must be either an Integer number or an indefinite value indicated by the symbol *. The value of *m* must be inferior to the value of *n*.

$$\forall A : \min(A, m) \wedge \max(A, n) \Rightarrow (m \in Z) \wedge (n \in Z \cup \{*\}) \wedge (0 \leq m \leq n)$$

$$Z : Integer\ numbers$$

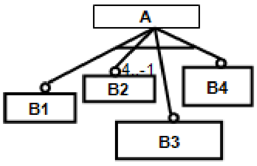


Figure 5. In this example, the limits of the group cardinality are not correct values, for instance: they are not ordered in an incremental manner and they contain a negative value.

5. Correct number of selected features from a group cardinality (Czarnecki *et al.*, 2005), (Osman *et al.*, 2008): in a configuration process, the number of selected features from a group cardinality must be superior to *min* and inferior to *max*. This criterion is applicable in PLCMs derived from cardinality-based PLMs.

$$\forall A, B : optional(A, B) \wedge select(B) \wedge \min(A, m) \Rightarrow sum(A, (B)) \geq \min(A, m)$$

$$\forall A, B : optional(A, B) \wedge select(B) \wedge \max(A, n) \Rightarrow sum(A, (B)) \leq \max(A, n)$$

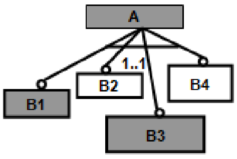


Figure 6. The number of selected child features is superior to the *max* value. Therefore, the resulting configuration (shaded features) does not correspond to the PLM.

6. No inclusion of a relative child: in this case a feature *A* require a feature *B* and at the same time *A* and *B* are related by combinations of relationships. For example, *A* and *B* are path relative features (*B* can be relative-full-mandatory to *A* or not).

$$\forall A, B, C : (relativePath(A, B) \vee (relativePath(C, B) \wedge require(A, C)) \vee$$

$$(require(A, C) \wedge exclude(C, B)) \wedge require(A, B) \Rightarrow error$$

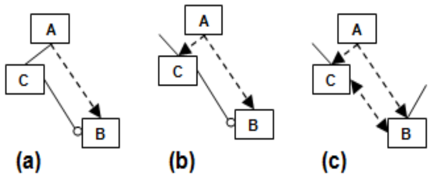


Figure 7. In (a), the relative path is defined by a *mandatory* and an *optional*-type relationships. It is defined in (b) by a *require* and an *optional*-type relationships, and in (c), the relative path between *A* and *B* is composed of a *require* and an *exclude*-type relationships.

4.2.1 Dead features-free criteria

7. No exclusion with a full-mandatory feature (Osman *et al.*, 2008), (Trinidad *et al.*, 2008), (van den Broek *et al.*, 2009), (von der Massen *et al.*, 2004), (Zhang *et al.*, 2004), (Metzger *et al.*, 2007): we can have two cases, in the first one, one of the features is optional and in the second one, two features are mandatory. In the first case, an optional feature is mutual exclusive to a full-mandatory feature.

Consequently, the optional feature can never be chosen in a configuration process and is considered as a dead feature. In the second case, a mutual exclusion between two full-mandatory features makes that both features become dead features. This verification function also includes the case where *A* is a path-relative feature with regard to a feature *B* (in this case, *B* can be either optional or mandatory, see *Figure 8b*).

$$\forall A, B : common(A, yes) \wedge (common(B, no) \vee relativePath(A, B))$$
$$\wedge\, exclude(A, B) \Rightarrow deadFeature(B)$$

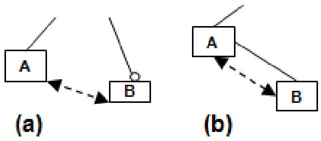


Figure 8. In (a), the full-mandatory feature *A* excludes the optional feature *B*, this latest one become a dead feature. In (b), a mutual exclusion between two mandatory features makes that both features became dead features.

8. No exclusion and requirement at the same time (Elfaki *et al.*, 2009), (Osman *et al.*, 2008), (Trinidad *et al.*, 2008), (von der Massen *et al.*, 2004): a mutual exclusion and a requirement between two features, simultaneously, make that the feature that requires the second one becomes a dead feature. Thus, two features cannot be mutual exclusive and required at the same time.

$$\forall A, B : (require(A, B) \vee require(B, A)) \wedge exclude(A, B) \Rightarrow deadFeature(A)$$

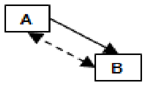


Figure 9. In this case, feature *A* can never be selected due to mutual exclusion and requirement with feature *B* at the same time.

4.3 Consistency criteria

9. No full-mandatory features requiring optional features (Trinidad *et al.*, 2008), (von der Massen *et al.*, 2004): in this case there are optional features being required by a full-mandatory feature. Consequently the optional feature is not optional anymore but becomes a full-mandatory feature as well. This case is treated as an error in (Trinidad *et al.*, 2008).

$$\forall A, B : optional(\_, B) \wedge common(A, yes) \wedge require(A, B) \Rightarrow inconsistency$$

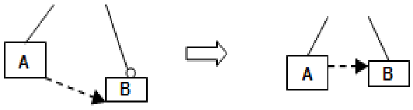


Figure 10. In this example, if a full-mandatory feature *A* requires one optional feature *B*, then *B* is not more optional and becomes a full-mandatory feature as well.

4.4 Redundancy-free criteria

10. No exclusion in a group cardinality: In a cardinality set with only two elements in which only one can be chosen, an exclude relationship between these two elements is redundant.

$$\forall A, B, C : optional(A, B) \wedge optional(A, C) \wedge$$
$$(\min(A,0) \vee \min(A,1)) \wedge \max(A,1) \wedge exclude(B, C) \Rightarrow redundant$$

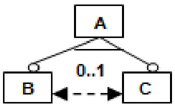


Figure 12. As in the cardinality *max*=1, this implies a mutual exclusion between the child features and the dependency is therefore superfluous.

11. No full-mandatory feature required by another feature (von der Massen *et al.*, 2004): a full-mandatory feature is implied by another feature. As the first feature is already full-mandatory, the implication is superfluous.

$$\forall A, B : common(A, yes) \wedge require(B, A) \Rightarrow redundant$$

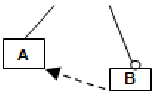


Figure 13. *B* can or can not be a full-mandatory feature, in any case, feature *A* is always selected and the require-type relationship is redundant.

12. No multiple require-type relationships from relative-path features (von der Massen *et al.*, 2004): A feature *B* is included by multiple features *A*,*C*… whereas *A* and *C* are relative-path features. The implication from *C* to *B* is then superfluous.

$$\forall A, B, C : relativePath(A, C) \wedge require(A, B) \wedge require(C, B) \Rightarrow redundant$$

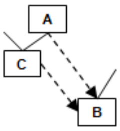


Figure 14. In this example the implication from *C* to *B* is superfluous.

13. No transitive include-type relationships (von der Massen *et al.*, 2004): a feature *A* requires a feature *C*, *C* requires *B* and *A* requires *B*. As *B* is already required by the transitive inclusion from *A* through *C*, the direct requirement from *A* to *B* might be superfluous. The formal description of this criterion only reflects the situation of three features, but it can be extended to more than three, following the systematic construction defined in the next formula.

$$\forall A, B, C : require(A, C) \wedge require(C, B) \wedge require(A, B) \Rightarrow redundant$$

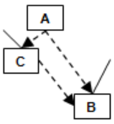


Figure 15. This example shows a superfluous inclusion from *A* to *B* since is already include from *A* through *C*.

14. No cyclic require-type relationships: a feature *A* includes a feature *C*, *C* requires *B* and *B* requires *A*. The cycle can be started in any feature. In any case, the latest include-type relationship is redundant since the triggered feature must be already selected. The formal description of cyclic include-type relationships only reflects the situation of three features, but it can be extended to more than three, following the systematic construction defined in the next formula.

$$\forall A, B, C : require(A, B) \wedge require(C, B) \wedge require(B, A) \Rightarrow redundant$$

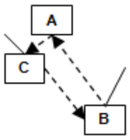


Figure 16. If feature *B* is selected, then *B* requires *A* and *A* requires *C*, therefore the *B* requires *A* relationship is redundant because feature *B* is already selected.

15. No requirement of a relative father (Trinidad *et al.*, 2008), (von der Massen *et al.*, 2004): elements of the same relative path must not be related by require-type relationships. This case is not exactly an error, it is a redundancy.

$$\forall A, B : relativePath(A, B) \wedge require(B, A) \Rightarrow redundancy$$

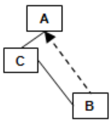


Figure 17. In this example, *B* requires *A* relationship is redundant.

5. Related works and discussion

Zhang *et al.* (Zhang *et al.*, 2004) have proposed logical expressions to verify three criteria in different binding times: (i) satisfiability, to "ensure that there is no inconsistency in tailoring and binding actions"; (ii) usability, to "ensure that every feature not yet selected has the possibility of being bound in some future binding time"; and (iii) suitability, to "ensure that every feature not yet selected has the possibility of being removed in some future binding time". Zhang *et al.* hold that these FOL verification criteria "can be automated by using model checking, such as SMV[1]". Czarnecki and Pietroszek (Czarnecki *et al.*, 2006)'s approach support the verification of feature-based models against templates using OCL-based well-formed rules. They "give an automatic verification procedure which can establish that no ill-formed template instances will be produced given a correct configuration of the template's feature model", that is, their work is centered in verification correctness of the instances of a PLM and not in the model itself. Batory (Batory, 2005) use grammar and propositional formulas, in order to represent basic FMs using context–free grammars plus propositional logic enabling logic truth maintenance systems and SAT solvers to identify contradictory (or inconsistency) predicates in a FODA model and to "verify that a given combination of features defines a product". Batory's approach is validated in the Guidsl tool (Batory, 2005), which also assign a unique number to each PLM graph vertex, computes the connected components of an undirected graph, computes the strongly connected components of a directed graph, determines if there are cycles in a PLM graph, computes a minimum spanning tree and computes the shortest path from a source

[1] http://www-2.cs.cmu.edu/~modelcheck/smv.html

vertex to all other vertices. Boolean equations are also used by Benavides *et al.* (Benavides *et al.*, 2005) in order to analyze FODA models. Their analysis consists in finding just one solution (with no preference as to which one), finding all solutions and finding an optimal solution by means of an objective function defined in terms of one or more variables. They have developed a tool[2] that uses the constraint satisfaction problem solver OPL Studio. Trinidad *et al.* (Trinidad *et al.*, 2008) has defined a method to detect dead features and full mandatory features based on theory of diagnosis (Trinidad *et al.*, 2008), the verification criteria that they cover are cited in *Table 1*. Janota and Kiniry (Janota *et al.*, 2007) use higher-order logic (HOL) to reason about feature models, in particular, they propose HOL expressions for root selectivity, existence of a path of selected features from the root to a feature that has been selected, and cardinality satisfaction of a selected feature. They also offer some lemmas formalized in HOL: (i) "If a group *g* has exactly the admissible cardinality 1, and contains exactly one member *m*, then in any valid configuration that selects the owner of that group, *m* is selected as well"; and (ii) "Whenever a new feature tree *ft2* is obtained from an existing feature tree *ft1* by removing some admissible cardinalities of a certain group *g*, the feature tree *ft2* is a specialization of the original tree *ft1*" implemented in the Mobius[3] program verification environment. Broek and Galvão (van den Broek *et al.*, 2009) analyze FODA models using generalized feature trees, in particular, they propose functions to detect existence of products, dead features, products which contain a given set of features, minimal set of conflicting constraints, to calculate the number and the list of all products, and to generate explanation of dead features. Wang *et al.* (Wang *et al.*, 2005) proposed to use description logic and Protégé-OWL to verify consistency of configuration models against its PLM. Their process consist in transform the FODA model into OWL, then, load the resulted ontology into the OWL reasoner FaCT++[4] and check its consistency. Elfaki *et al.* (Elfaki *et al.,* 2009) propose to use FOL to detect dead features, inconsistencies due to contradictions between include and exclude relationships, and to propose inconsistency-prevention in FMs. Their innovative work is the proposition of expressions dealing with individuals and also sets of the features, the verification criteria that they cover are indicated in *Table 1*.

Table 1 resumes our literature review of feature models verification criteria, classified in *Figure 3* and shows how to select a particular criterion according to the modelling formalism in use. This literature review shows that verification criteria are not systematically presented and treated across the literature. Also, that almost all research efforts are centered in verification of FODA-like models, neglecting the other formalisms. Finally, it is also showed in *Table 1* that some of the verification criteria presented in this paper have never been systematically tried and formalized, as far as we know. Perhaps, because there are different levels of importance and the research community has given more importance to some criteria that to others or because some are more difficult to identify that others.

[2] http://www.tdgseville.info/topics/spl

[3] http://mobius.inria.fr/twiki/bin/view/Mobius

[4] http://owl.man.ac.uk/factplusplus

Table 1. *Literature overview of verification criteria that have been applied at explicitly one or more PLM formalisms. Contributions highlighted in bold were not automated by their authors.*

Languages Criteria	FODA	FOPLE	FORM	Czarnecki's app	FORE	Bosch's approach
1. No void	(Trinidad *et al.*, 2008), (van den Broek *et al.*, 2009)					
2. Richness or no false PLM						
3. Well defined boundaries	N/A	N/A	N/A	**(Czarnecki *et al.*, 2005)**	**(Czarnecki *et al.*, 2005)**	N/A
4. Correct domain value of boundaries	N/A	N/A	N/A	**(Czarnecki *et al.*, 2005)**	**(Czarnecki *et al.*, 2005)**	N/A
5. Correct number of selected features from a feature set				**(Czarnecki *et al.*, 2005)**	**(Czarnecki *et al.*, 2005)**	
6. No inclusion of a relative child						
7. No exclusion whit a full mandatory feature	(Elfaki *et al.*, 2009), (van den Broek *et al.*, 2009), (Trinidad *et al.*, 2008)	(Elfaki *et al.*, 2009)	(Elfaki *et al.*, 2009)	(Elfaki *et al.*, 2009)	(Elfaki *et al.*, 2009)	(Elfaki *et al.*, 2009)
8. No exclusion and requirement at the same time	(Elfaki *et al.*, 2009), (Osman *et al.*, 2008), (Trinidad *et al.*, 2008), (van den Broek *et al.*, 2009), **(von der Massen *et al.*, 2004)**	(Elfaki *et al.*, 2009), (Osman *et al.*, 2008)	(Elfaki *et al.*, 2009), (Osman *et al.*, 2008)	(Elfaki *et al.*, 2009), (Osman *et al.*, 2008)	(Elfaki *et al.*, 2009), (Osman *et al.*, 2008)	(Elfaki *et al.*, 2009), (Osman *et al.*, 2008)
9. No full-mandatory features requiring optional features	(Trinidad *et al.*, 2008), **(von der Massen *et al.*, 2004)**					
10. No exclusion in a group cardinality						
11. No full-mandatory features included by another feature	**(von der Massen *et al.*, 2004)**					

12. No multiple include-type relationships from relative-path features	**(von der Massen *et al.*, 2004)**					
13. No transitive include-type relationships	**(von der Massen *et al.*, 2004)**					
14. No cyclic include-type relationships	(Batory, 2005)					
15. No inclusion of a relative father	(Trinidad *et al.*, 2008), **(von der Massen *et al.*, 2004)**					

Discussion

Verification of PLMs is an important task in domain and application engineering. With the growth of the number of features in PLMs, manual checking becomes very laborious and error-prone. In spite of the fact that many approaches are proposed to fix these lacks, even a complete and formalism-independent method of verification is necessary. However, our classification of verification criteria allows better understands the similarities and differences between existing FMs verification approaches and to enrich the verification criteria of PLMs. Besides, it can be extended with other criteria. We have also formalized the criteria as an attempt to set a base ground for automated verification of FMs based on an off-the-shelf solver. Thus, these formalized criteria are the requirements of any future tool that intends to automate the verification of feature models.

Our approach has been validated using the Stago's and Baxter product line models. Diagnostica Stago, Inc. is a French industry offering a set of hemostasis instrumentation and optimized reagent kits for research as well as for routine analysis. On the other hand, Baxter International Inc. develops, manufactures and markets products for people with hemophilia, immune disorders, infectious diseases, kidney disease, trauma, and other chronic and acute medical conditions. By confidentially reasons we cannot present neither Stago's nor Baxter's product line model, in which we have indentified 85% of its anomalies using our list or verification criteria, classified in *Figure 2*. Criteria to identify the rest of anomalies, like: the existence of a path from the root to a feature that has been selected or the root selectivity, are not yet included in our classification. The improvement of our criteria classification is part of our future work.

6. Conclusions and future work

Verification of PLMs is one of the most important challenges in product line engineering. This error-prone activity has been centered, by the scientific community, in some verification criteria for FODA-like models. Even if FODA is one of the most used and accepted formalism to model product line systems, it is not the only one language and it cannot be used in all different views of a system. In this paper we explore some errors that not had been explored before, and arrange our minimal collection of verification criteria in a classification that consider the nature or the error-type that can be identified. Researchers and engineers can use our FOL formalization and exhaustive explanation of each criterion to guide a PLM verification process, as we have made in two real cases. We are conscious that even though it is not possible to state that a collection of verification criteria ensures the correctness of a cardinality-based FM, they improve the quality of the PLM. Besides, our analysis of verification criteria based on common errors in different types of FMs, can be used as base-line further verification analysis and automation of PLMs. At present we are working on the development of a prototype that allows automating the verification process of feature-based models. A first version of the prototype has been presented in (Salinesi *et al.*, 2009). Our aim is to create a formalism-independent framework, in which, every PLM may be verified by means of our extensible classification of criteria. Thus, although we have presented a complete and up-to date literature review on verification of FMs (*Table 1*), there are others PL modelling formalisms that have not been considered in this paper and that are envisaged for future works.

7. References

Batory D., Feature Models, Grammars, and Propositional Formulas. *In 9th International Software Product Lines Conference (SPLC05)*, Rennes, France, 2005.

Benavides D., Ruiz-Cortés A., Trinidad P., Automated reasoning on feature models. *In Proc. of CAiSE'05*, 2005, pp. 491–503.

Benavides D., Ruiz–Cortés A., Trinidad P., Segura S., A Survey on the Automated Analysis of Feature Models. *XV Jornadas de Ingenieria del Software y Bases de Datos JISBD'06. José Riquelme - Pere Botella (Eds).* CIMNE, Barcelona, 2006.

Bjorner D., Software Engineering 3: Domains, requirements and Software Design. *Springer-Verlag* 2006

Czarnecki K., Helsen S., Eisenecker U., Formalizing cardinality-based feature models and their specialization. *Software Process Improvement and Practice*, 10(1):7– 29, 2005

Czarnecki K., Pietroszek K., Verifying Feature-Based Model Templates Against Well-Formedness OCL Constraints. *5th Int. Conference on Generative Programming and Component Engineering*, 2006.

Elfaki A., Phon-Amnuaisuk S., Kuan Ho C., Using First Order Logic to Validate Feature Model. *Third Int. Workshop VaMoS*, 2009

Fantechi A., Gnesi S., Lami G., Nesti E., A Methodology for the Derivation and Verification of Use Cases for Product Lines. *In proceedings of SPLC2004*, Boston, 2004.

Gurp J., Bosch J., Svahnberg M., On the Notion of Variability in Software Product Lines. In the *Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2001.

Janota M., Kiniry J., Reasoning about Feature Models in Higher-Order Logic. *In 11th Int. Software Product Line Conference (SPLC07)*, 2007.

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

Metzger A., Heymans P., Pohl, K., Schobbens P., Saval G., Disambiguating the Documentation of Variability in Software Product Lines. *In Proc. of RE'07*, 2007.

Osman A., Phon-Amnuaisuk S., Kuan Ho C., Knowledge Based Method to Validate Feature Models. *12th Int. Software Product Line Conference,* 2008.

Salinesi C., Rolland C., Mazo R., VMWare: Tool Support for Automatic Verification of Structural and Semantic Correctness in Product Line Models. *Third International Workshop VaMoS*, Spain, 2009.

SEI, Software Engineering Institute, Carnegie Mellon University, last consulted 11-04-2010 http://www.sei.cmu.edu/productlines/

Streitferdt D., Family-Oriented Requirements Engineering. PhD Thesis, *Technical University Ilmenau*, 2003.

Trinidad P., Benavides D., Durán A., Ruiz-Cortés A., Toro M., Automated error analysis for the agilization of feature modeling. *Journal of Systems & Software – Elsevier*, 2008.

Turner C., Fuggetta A., Lavazza L., Wolf A., A conceptual basis for feature engineering. *J. System Software 49*, 1999.

van den Broek P., Galvão I., Analysis of Feature Models using Generalised Feature Trees. *Third International Workshop VaMoS*, Sevilla-Spain, 2009.

von der Massen T., Lichter H., Deficiencies in feature models- Towards Tool Support, *Workshop on Software Variability Management for Product Derivation*, 2004.

Wang H., Fang Li Y., Sun J., Zhang H., Verify Feature Models using Protégé-OWL. *ACM'05*, Chiba, Japan, 2005.

Zhang W., Zhao H., Mei H., A propositional Logic-based Method for Verification of Feature Models. *6th Int. Conference on Formal Engineering Methods ICEFEM*, 2004.