An Ontology of Verification Criteria in the Product Line Domain

Camille Salinesi¹, Colette Rolland¹, Raúl Mazo¹,²
¹ CRI, Panthéon Sorbonne University, 90, rue de Tolbiac, 75013 Paris, France
² Ingeniería & Software, University of Antioquia, Medellín, Colombia
{Camille.salinesi, Colette.Rolland}@univ-paris1.fr, raulmazo@gmail.com

Abstract

Product Line (PL) based development is a promising approach to develop software intensive systems. The promises are multiple: reduced time to market, better reuse and therefore reduced development costs, etc. While PL modelling languages and PL configuration processes have generated many researches, many efforts remain to verify PL models (PLMs). Adequate PLM verification methods, techniques and tools are needed for better system quality. Indeed, any error in a PLM will inevitably spread to its configurations and will generate PL architecture stability issues, overbudget projects, etc. This paper presents a survey on PLM verification approaches. Based on this survey, we propose an ontology of verification criteria in which each criterion is formalized using first order logic. The survey can be used for three purposes: (a) to better understand the similarities and differences between existing PL model verification approaches; (b) to set a base ground for automated PL model verification tool, based on an off-the-shelf satisfiability solver; and (c) to identify gaps in the literature of model verification in the PL domain.

1. Introduction

Product Line Engineering is emerging as a viable and important development paradigm allowing companies to realize important improvements on time to market, cost, productivity, quality and flexibility [19]. In this approach a family of products is represented through a product line model, and each product through a configuration model. Configuration model can be specified through a ‘configuration’ process that consists in retrieving from the product line model the collection of features that specify the single product to build. Any error in a product line model will inevitably affect configuration models.

By verification of a PLM [2] we mean the formal process of determining whether or not a requirement specification satisfies a set of well defined criteria. These verification criteria ensure that the system is correct [9], which means there is no defect or error on it. Verification can be determined either by means of properties of the specification itself, or by means of a collection of properties of some other specification.

Many properties can be verified on a product line model, such as: consistency [10], [17]; correctness or Satisfiability of Constraints [2], [3], [18]; validity or richness [6]; suitability or utility [18] and usability or decidability [18]. Our particular interest is on correctness. Correctness of a product line model can be treated from two different points of view: structural correctness and semantic correctness. Structural correctness is about the correspondence between the model and the language in which the model is written. It also refers to the alignment between the model and a set of structural properties that must respect all model of the same type. Semantic correctness refers to: (i) richness and completeness of the PLM; (ii) consistency of PLM and of PLCMs with regard to PLM; and (iii) traceability, faithfulness, pertinence, modifiability and usability of PLMs.

Bibliographic research shows that many criteria, techniques and tools have been defined specifically to verify product line systems. The solutions developed over the last years have not been integrated into a coherent and flexible process of verification and validation (V&V). Thus, the two main motivations of this work are: (i) to unify definitions and criteria for verification of correctness in product line models; and (ii) to propose an ontology of operations and its techniques that allow verifying product line systems. Lacks in these two aspects are recognized by the community [13].

The remaining of this paper is structured as follows. Section 2 presents the feature models’ metamodel used to relate concepts appearing in criteria formalization. Section 3 presents the predicates needed to define the ontology of criteria presented in section 4. Section 5 develops each ontology’s leaf criterion. Section 6 concludes the paper and describes future works.
2. Feature models

In this paper we consider PLMs represented in the Family Oriented Requirements Engineering (FORE) notation [11], where a feature diagram is a directed acyclic graph. This notation provides two types of dependencies among features, namely: variation dependencies and transverse dependencies. Features are a clustering of individual requirement that describe a cohesive, identifiable unit of functionality [14] and are expressed by boxes labelled with the name of the feature. Variation dependencies are expressed by edges in which the direction is indicated by a circle at its end. If this circle is filled, then the dependency between the features is mandatory. If the circle is empty, the dependency is optional. Optional dependencies that originate from the same feature father can be grouped into a feature set or bundle. Each dependency can only be part of one bundle. A bundle has a cardinality that indicates the minimum and maximum number of features to be chosen from it. Visually, a bundle is shown by an arc connecting all the edges that are part of it. Transverse dependencies are expressed by two types of dotted arrow. If the arrow is simple, then the dependency between features is called requires. If the arrow is double, the dependency is called excludes. These concepts are related in the metamodel depicted in Figure 1.

Figure 1: Metamodel for product line models in FORE notation.

The central elements of the product line metamodel are Product Line Model and Product Line Configuration Model. A PLM is composed of three elements, the “Feature”, “Variant Dependency” and “Transverse Dependency”. PLCM are composed of “Features” and variant dependencies. The “Feature” class is specialised in two classes (“FatherFeature” and “ChildFeature”). One or several “ChildFeature” are associated by the relation “ChildOf” with a “FatherFeature” class. For more details see [7].

3. Predicates

Defining verification criteria as expressions in First Order Logic (FOL) has several advantages:

(1) FOL provides a uniform way of specifying the criteria. We consider that the formalised criteria are easy to adapt and reuse for other languages than FORE.

(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.

(5) In some cases, the logical expressions allow us not only to evaluate and detect an error but also to fix it. That is, we can use the second part of the logical expression (consequent) in order to fix the first part (antecedent).

In order to define criteria in FOL, a certain number of predicates [8] have to be defined:

- **type**: defines if a child feature A is affected by an optional or a mandatory-type relationship. e.g.: type (A, mandatory).
- **childOf**: identifies the relationships between a child feature B and its father A. e.g.: childOf (B, A), the root feature R does not have father, childOf (R, null).
- **max**: identifies the maximum number of features allowed to be selected in a cardinality relationship. e.g. max (Father Feature A, 4).
- **min**: Identifies the minimum number of features allowed to be selected in a cardinality relationship e.g. min (Father Feature A, 1).
- **common**: describes the commonality of specific feature. Features being a member of this group are called full-mandatory features [16]. e.g. common (A, yes) if the feature A is always selected in any configuration. If the feature is not common, the second slot in the predicate will become no, e.g. common (A, no).
- **include**: describes an inclusion relationship between two features (or group of features), that is, if a product contains feature A it also contains feature B, e.g. include (A, B).
- **exclude**: describes an exclusion relationship between two features (or group of features), that is, if a product contains Feature A, then it does not contain Feature B and vice-versa. e.g. exclude (A, B).
- **count**: count and returns the number of times that a feature A appears in the PLM. e.g. count (A).
- **relativePath**: returns true if feature A is an element in the path from the root of the PLM to B. The function returns false otherwise. e.g. relativePath (A, B).
4. Ontology of verification criteria

Mannion [6] was the first to connect First Order Logic (FOL) formulas to Feature Models (FMs) in a formal verification process. Zhang et al. [18] propose logical expressions to evaluate satisfiability, suitability and usability in PLMs detecting problems such as inconsistent constraints, conflicting or unnecessary binding resolution. Czarnecki and Pietroszek [3] propose an approach to verify feature-based model templates against OCL-based well-formed rules. Batory [1] use grammar and propositional formulas, in order to represent basic FMs using context-free grammars plus propositional logic enabling off-the-shelf satisfiability solvers to debug FM. Trinidad et al. [13] have defined a method to detect dead features based on finding all products and search for unused features. The idea is to automate error detection based on theory of diagnosis. This model mapped FM to diagnose-model and used Constraint Satisfaction Problems to analyze FM. Thang [12] defined a formal model in which each feature was separately encapsulated by a state transition model. The aim of the study is to improve consistency verification among features. Janota and Kiniry [5] use higher-order logic to reason feature models. Broek and Galvão [15] analyse feature models using generalised feature trees. Wang et al. [17] propose using description logic and Protégé-OWL to verify FMs. Elfaki et al. [4] propose to use FOL to verify FMs introducing expressions dealing with individuals and also sets of the features. Mazo and Salinesi [7] propose a method to verify some criteria in FMs and its configurations.

Ontology of Figure 2 incorporates the criteria to verify FMs founded in the above literature review and propose some innovative operations on certain criteria.

![Figure 2: Ontology of verification criteria for FMs](image)

In the ontology presented in Figure 2, the lefts are composed of collections of operations allowing verify the particular criterion. The ontology is developed as follow:

5. Correctness in variability models

1. PLM correctness

1.1. PLM Semantic Correctness

1.1.1. Obligatory criteria

The richness criterion implies that the PLM is not void. A feature model is void if it defines no product at all. This criterion is widely treated in literature [13], [15].

1.1.2. Optional criteria

Traceability [7], [20], not repeatability of features [16] and modifiability of each feature and each relationship [18], can be considered as optional operations allowing improve the PLM and its life-cycle management.

1.2. PLM Structural Correctness

1.2.1. Errors-free criteria

Root unicity: The feature model should have only one root element.

\[ \exists A: \text{common}(A, \text{yes}) \land \text{childOf}(A, \text{null}) \land (\text{count}(A) > 1) \Rightarrow \text{error} \]

Child-father unicity: A child feature should have one and only one father.

\[ \exists A, B, C: \text{childOf}(B, A) \Rightarrow \sim \text{childOf}(B, C) \]

![Figure 3: child feature B with two fathers A and C.](image)

Belong to a bundle: All features intervening in a cardinality bundle should be of optional type.

\[ \forall A, B: \text{min}(A, m) \land \text{max}(A, n) \land \text{childOf}(A, B) \Rightarrow \text{type}(B, \text{optional}) \]

![Figure 4: B1 is not an optional-type feature in this example. The error makes that features B2, B3 and B4 can never be selected.](image)

Well defined boundaries: the min value of the cardinality must be inferior that the set of features grouped in the cardinality bundle. The max value of the cardinality must be inferior or equal that the set of features grouped in the cardinality bundle.

\[ \forall A, B: \text{type}(B, \text{optional}) \land \text{childOf}(B, A) \land \text{min}(A, m) \Rightarrow \text{min}(A, m) < \text{sum}(A, B) \]

\[ \forall A, B: \text{type}(B, \text{optional}) \land \text{childOf}(B, A) \land \text{max}(A, n) \Rightarrow \text{max}(A, n) \leq \text{sum}(A, B) \]
A and B are path relative features (B can be relative relative to the other one).

Correct domain value of boundaries: in a defined cardinality bundle, m must be an Integer number and n must be an Integer number or have an indefinite value indicated by the symbol *. The value of n must be neither inferior to the value of m.

\[ \forall A : \min(A, m) \land \max(A, n) \Rightarrow (m \in \mathbb{Z}) \land (n \in \mathbb{Z} \cup \{\} \land (0 \leq m \leq n) \]

Z: Integer Numbers

Exclusion with a common feature [8], [13], [15], [16], [18]: in this case there are optional features in mutual exclusion with a common feature. In other words, 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.

This verification function also includes the case where the second feature is a relative-common feature with regard to the first one.

\[ \forall A, B : \text{common}(A, yes) \land \text{common}(B, no) \lor \text{relativePath}(A, B) \land \text{exclude}(A, B) \Rightarrow \text{deadFeature}(B) \]

Figure 7: In (a), the common feature A excludes the optional feature B, which becomes a dead feature. In (b), one of the features is path-relative to the other one.

Inclusion of a relative feature: in this case a feature A includes 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).

\[ \forall A, B, C : (\text{relativePath}(A, B) \lor (\text{relativePath}(C, B) \land \text{include}(A, C)) \lor (\text{include}(A, C) \land \text{exclude}(C, B)) \land \text{include}(A, B) \Rightarrow \text{error} \]

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

Figure 6: In this example, the limits of the bundle do not correspond to the correct values of a cardinality.

Figure 8: (a) the relative path is defined by Variant Dependency-type relationships. It is defined in (b) by a combination of Transversal Dependency and Variant Dependency-type relationships, and in (c), the relative path between A and B is composed of Transversal Dependency-type relationships.

1.2.2. Consistency criteria

Full-mandatory features requiring optional features [16]: in this case there are optional features being required by a common feature. In other words, an optional feature is included 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 [13].

\[ \forall A, B : \text{type}(A, mandatory) \land \text{type}(B, optional) \land \text{common}(A, yes) \land \text{require}(A, B) \Rightarrow \text{type}(B, mandatory) \land \text{common}(B, yes) \]

Figure 9: in this example, if a full-mandatory feature A requires one optional feature B, then B is not more optional and becomes a member of the commonality set.

Figure 10: represents the case where two common features are mutually excluded, that is inconsistent because both features have to be part of every potential product.

Exclusion between two common features: a mutual exclusion between two common features is inconsistent because both features have to be part of every potential product; therefore they must not exclude each other. This case is considered as an error in [13].

\[ \forall A, B : \text{common}(A, yes) \land \text{common}(B, yes) \land \text{exclude}(A, B) \Rightarrow \text{inconsistency} \]

Exclusion and requirement at the same time [8], [13], [16]: A mutual exclusion and a requirement between two features, simultaneously is considered as an inconsistency in PLMs. Two features cannot be mutually exclusive and required at the same time.

\[ \forall A, B : (\text{include}(A, B) \lor \text{include}(B, A)) \land \text{exclude}(A, B) \Rightarrow \text{inconsistency} \]
Inclusion among child features intervening in a cardinality bundle: inclusion between two alternative-child features that belongs to the same cardinality bundle can to cause an error in some cases. These types of inconsistencies occur when the number of inclusion-type relationships among the alternative-child features is equal or exceeds the cardinality’s max value.

\[ \forall A, B, C : \text{childOf}(B, A) \land \text{childOf}(C, A) \land (\text{count}(\text{include}(B, C)) \geq \max(A, n)) \Rightarrow \text{inconsistency}\]

![Figure 11](image)

**Figure 11:** in these examples we depict the two cases of inconsistencies due to mutual exclusion and requirement between two features at the same time.

Transitive include-type relationships [16]: a feature A includes a feature C, C includes B and A includes B. As B is already included by the transitive inclusion from A through C, the direct inclusion from A to B might be superfluous.

\[ \forall A, B, C : \text{include}(A, C) \land \text{include}(C, B) \land \text{include}(A, B) \Rightarrow \text{redundant}\]

![Figure 15](image)

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

Multiple include-type relationships from relative-path features [16]: 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.

\[ \forall A, B, C : \text{relativePath}(A, C) \land \text{include}(A, B) \land \text{include}(C, B) \Rightarrow \text{redundant}\]

![Figure 16](image)

**Figure 16:** in this example the implication from C to B is superfluous.

Exclusion in a cardinality-bundle: In a bundle with only two elements in which only one can be chosen, an exclude relationship between these two elements is redundant.

\[ \forall A, B, C : \text{type}(B, \text{optional}) \land \text{type}(C, \text{optional}) \land \text{childOf}(B, A) \land \text{childOf}(C, A) \land (\min(A, 0) \lor \min(A, 1)) \land \max(A, 1) \land \text{exclude}(B, C) \Rightarrow \text{redundant}\]

![Figure 17](image)

**Figure 17:** As in the cardinality max=1, this implies a mutual exclusion between the child features and the dependency is therefore superfluous.

A common feature include by another feature [16]: a common feature is implied by another feature. As the feature is already full-mandatory, the implication is superfluous.

\[ \forall A, B : \text{common}(A, \text{yes}) \land \text{require}(B, A) \Rightarrow \text{redundant}\]

![Figure 18](image)

**Figure 18:** B can or can not be a common feature, in any case, feature A is always selected and the require-type relationship is redundant.

1.2.3. Redundancy-free criteria

**Inclusion of a relative-father** [13], [16]: elements of the same relative path must not be related by include-type relationship. This case is not exactly an error, it is a redundancy.

\[ \forall A, B : \text{relativePath}(A, B) \land \text{include}(B, A) \Rightarrow \text{redundant}\]

![Figure 13](image)

**Figure 13:** in this example, B include A relationship is redundant.

**Cyclic include-type relationships:** a feature A includes a feature C, C includes B and B includes 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.

\[ \forall A, B, C : \text{include}(A, B) \land \text{include}(C, B) \land \text{include}(B, A) \Rightarrow \text{redundant}\]

![Figure 14](image)

**Figure 14:** if feature B is selected, then B includes A and A includes C, therefore the B includes A relationship is redundant because feature B is already selected.

2. PLCMs correctness

2.1. PLCM Semantic Correctness
2.1.1. Obligatory criteria
PLCM’s compliance to the corresponding PLM [7], [18]: in order to check this criterion it is necessary to verify the feature existence (every feature in a PLM must also be a member of the PLM) and the PLM’s constraint satisfaction (PLCMs’ structure must to be according to PLM’s structure and restrictions).

Bounder criterion [8]: in a configuration process, the number of selected features from a bundle must be superior to \( \min \) and inferior to \( \max \).
\[
\forall A, B : type(B, optional) \land childOf(B, A) \land select(B) \\
\min(A, m) \Rightarrow \text{sum}(A, (B)) \geq \min(A, m) \\
\forall A, B : type(B, optional) \land childOf(B, A) \land select(B) \\
\max(A, n) \Rightarrow \text{sum}(A, (B)) \leq \max(A, n)
\]

Figure 19: the number of selected child features is superior to the \( \max \) value. The PLCM corresponds only to shared features and its variant relationships.

2.1.2. Optional criteria
Traceability [7], modifiable features and feasible PLCMs on real cases are advisable and treated in [20].

2.2. PLCM Structural Correctness
Root unicity and Child-father unicity: these criteria are already presented in section 1.2.1.

6. Conclusions and Future work

Verification of variability models in the product line domain is an important task in domain and application engineering. With the growth of the number of features in PLMs manual checking of structural correctness and PLCMs validity are very laborious and error-prone. By this reason, many approaches are proposed to fix these lacks. Our ontology of verification criteria allows better understand the similarities and differences between existing FMs verification approaches. It can be extended with other criteria. We have also formalised most of the criteria as an attempt to set a base ground for automated verification of FMs based on an off-the-shelf satisfiability solver. This paper is also useful to identify gaps in the literature of model verification in the PL domain and allows defining new axes of research.

Although we have presented a complete literature review on verification of FMs, there are others PL modeling formalisms that have not been considered in this paper and that are envisaged for future works.

7. References