VMWare: Tool Support for Automatic Verification of Structural and Semantic Correctness in Product Line Models

Camille Salinesi[1], Colette Rolland[1], Raúl Mazo[1,2]
[1] *CRI, Université Paris 1 – Sorbonne, 90, rue de Tolbiac, 75013 Paris, France*
[2] *Ingeniería & Software, Universidad de Antioquia, Medellín, Colombia*
{Camille.salinesi, Colette.Rolland}@univ-paris1.fr, raulmazo@gmail.com

Abstract

The verification of variability models is recognized as one of the key challenges for automated development of product lines. Some computational tools have been proposed to verify product line models and product line configurations models. VMWare is a tool integrating different criteria to verify structural and semantic correctness of models derived from the FORE metamodel. Our tool gives the possibility of (i) build feature-based product line models and product line configuration models, (ii) verify their structural and semantic correctness in a completely automated manner and (iii) import/export them in XMI files.

1. Introduction

Feature Modelling is a mechanism to represent requirements in the context of Software Product Lines (SPL). A Feature Model (FM) defines features and their usage constraints in product-lines (PL). Their main purposes are: (i) to capture feature commonalities and variabilities; (ii) to represent dependencies between features; and (iii) to determine combinations of features that are allowed and disallowed in the product line. A feature is a product characteristic that some stakeholders (e.g. users, sellers, engineers, customers) consider important to include in the description of the product line.

Automated analysis of FMs is recognized in the literature as an important challenge in PL engineering and is considered as an open issue by many SPL researchers [1], [2], [4], [10]. Verification of FMs is important for industry because any error in a Product Line Model (PLM) will inevitably affect the configuration models (PLCMs) and thereafter final products. By verification of FMs we mean the formal process of determining whether or not they satisfy well defined verification criteria. Verification criteria can be determined either by means of properties of the specification itself, or by means of a collection of properties of some other specification. FMs correctness includes structural correctness and semantic correctness.

This paper presents a prototype tool for PLMs and PLCMs construction and verification. The tool is based on a framework for the automated analysis of feature models. Broadly speaking, it allows: (i) creating PLMs and PLCMs; (ii) verifying structural correctness criteria of PLMs and PLCMs; (iii) verifying semantic correctness of PLMs; and (iv) verifying PLCMs in regard to PLMs. The implementation is based on a three-layer architecture and uses XMI files as a mechanism to exchange the FMs with other tools.

The remainder of the paper is structured as follows. Section 2 gives a brief overview of feature modeling and of the verification process. Section 3 describes the functionality and provides some implementations details of the framework. Section 4 concludes the paper and describes future works.

2. Feature Modeling and Verification

Feature modeling is the activity of identifying externally visible characteristics of products in a domain and organizing them into a feature model. The notation considered in this paper is FORE notation (Feature Oriented Requirements Engineering) [3].

The characteristics of the FORE notation are:

- a feature diagram is a Directed Acyclic Graph (DAG);
- a feature is represented by a node of this graph;
- relationships between features are represented by links. There are two types of relationship, namely variant dependency and transverse dependency;
- *variant dependencies* can be mandatory or optional. The collection of features related by variant dependencies take the form of a tree;

- *transverse dependencies* can be of two kinds: the *excluding* one or the *requiring* one;
- *optional* relationships with the same father can be grouped into a bundle. A relation can be member of one and only one bundle;
- a bundle has a cardinality that indicates the minimal and maximal number of features that can be chosen. The meaningful cardinalities are: 0..1, 1, 0..*N*, 1..*N*, *N*, *p*, 0..*p*, 1..*p*, *p*..*N*, *m*..*p*, 0..* and 1..*;
- graphically, a bundle of variant dependencies is represented by an arch that related all the implicated relations;

The FORE notation fits the construction of PLMs, while eliminating many ambiguities. However, there are no well established guidelines to identify structural and semantic errors in FORE models.

The FM verification process that we propose can be summarized in *Figure 1*. The process is structured around two cycles, the first one corresponds to PLMs verification and the second one corresponds to PLCMs verification.

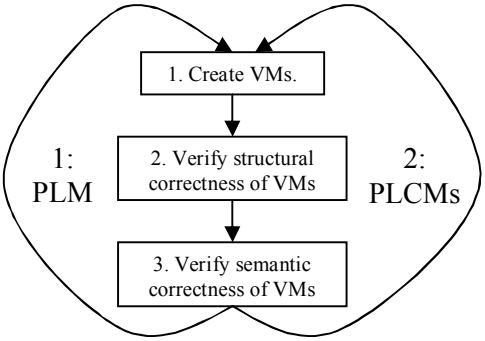

Figure 1. FORE-based PLMs and PLCMs correctness verification process.

2.1. Verify the structural correctness criteria of the Feature Model

Structural correctness concerns: (i) the correspondence between the model and the language in which the model is written; and (ii) the alignment between the model and a set of structural properties that any model of the same type must respect.

The purpose of the VMWare tool is to automatically verify FORE-based models according to a collection of well defined criteria [11]. To achieve this, we have divided the collection of criteria into three groups: (i) general criteria that every FORE-based FM shall respect; (ii) criteria specific to PLMs; and (iii) criteria specific to PLCMs.

In order to build a complete and consistent list of criteria, we undertake a state of the art of computational tools for construction of variability models supporting their automated verification. A summary of the criteria supported by the analysed tools are presented in Table 1.

Table 1. Structural and semantic correctness criteria (not) implemented in related tools.

Tool		**VMWare**	**Feature Plugin** [5] / **DECIMAL** [8]	**XFeature**[1] [13] / **Pure::variants**[2] [7]	**Requiline**[3] [12]	**FAMA** [9]
Modeling Formalism		FORE Cons-traints	FOD A* / Class	FOR E	FOR M	FOR E
PLCM Verification		Y	Y	Y	Y	Y
PLM Verification		Y	N	Y	Y	Y
Criteria						
Structural	Root uniqueness	Y	N	Y	Y	N
	Child-father uniqueness	Y	Y	Y	Y	N
	Ordered cardinality	Y	N	N	N	N
	Applicable cardinality	Y	N	N	N	N
	Optional features and include dependencies coherence	Y	N	N	Y	Y
	Mandatory features and exclude dependencies coherence	Y	N	N	Y	Y
	Well limited cardinalities	Y	N	Y	N	N
	Consistency between transversal dependencies and cardinalities	Y	N	N	N	N
	No dead features	Y	N	N	N	Y
	DAG Structure	Y	?	Y	Y	N
Semantic	Richness – No void feature models	N	?	N	?	Y
	PLCM's compliance to the corresponding PLM	Y	Y	Y	Y	N
	Traceability	P	?	P	Y	?
	Uniqueness	N	?	?	N	N
	Pertinence	N	?	?	Y	N
	Modifiability	N	?	?	Y	?

Legend: Y = Yes, N = No, P = Partially, ? = unavailable information

* FODA with cardinality-based feature modeling.

[1]http://www.pnp-software.com/XFeature/

[2]http://www.software-acumen.com/purevariants/feature-models

[3]http://www-lufgi3.informatik.rwth-aachen.de/TOOLS/requiline

The criteria that we have chose for VMWare are defined bellow.

General criteria

1 *Root uniqueness*: The PLM should have only one root element.
2 *Child-father uniqueness*: A child feature should have one and only one father.
3 *Tree structure*: Variability structure of PLM, as well as PLCMs should be represented as connected and acyclic graphs.

PLM criteria

1 *Ordered cardinality*: All features grouped by a cardinality should be ordered in a consecutive manner.
2 *Applicable cardinality*: All features intervening in a cardinality should be optional.
3 *Optional features and include dependencies coherence*: This state of structural correctness criteria is respected when a feature is not at same time: mandatory and exclude dependent.
4 *Mandatory features and exclude dependencies coherence*: The state of structural correctness criteria is respected when a feature is not simultaneously: optional and require dependent.
5 *Well limited cardinality*: The state of structural correctness is respected when: (i) superior limit >= ||bundle||; and (ii) there are no cardinalities where both boundaries have 0 value (e.g. "0,0"), or the superior limit is lower than the inferior one, or where the inferior limit is a negative number.
6 *Consistency between transversal dependencies and cardinalities*: This criterion is determined by three conditions: (i) cardinality of bundle should be well formed; (ii) if a feature is involved in a bundle, then this feature cannot be related by a transverse relationship with other feature of the same bundle; and (iii) the same feature must not belong to two different bundles.
7 *No dead features*: It should be possible to include every feature in a PLM in at least one PLCM.
8 *DAG structure*: In a PLM it is forbidden to find a collection of features forming a cycle by means of *Transversal Dependencies* and/or *Variant Dependencies*. In order to evaluate this criterion, variability dependencies are enriched with a direction from the father to child. Transversal dependencies preserve its original directions. Thus, errors like exclusion (inclusion) of an ancestor and vice versa are identified.

Each of these criteria has been formally specified using first order logic predicates [11]. This allows implementing verification systematically using a SAT-like solver. For example, criterion ***child-father uniqueness*** was formally defined as follow:

$$\forall(feature\ P_i, feature\ C) \in PLM.\ FatherFeature(P_i) \wedge ChildFeature(C) \wedge (C\ childOf\ P_i) \wedge ((C\ Mandatory\ P_i) \oplus (C\ optional\ P_i)) \Rightarrow \left|(C \bullet P_i) \oplus (C \circ P_i)\right| = 1$$

Where "●" represents a mandatory and "○" an optional relationship between father and child features.

3. Implementation

The technologies used in the development process of our tool are:
(i) The Microsoft .NET Framework v2.0.50727 provided the general libraries.
(ii) Its source code was written using Microsoft Visual Studio 2005.
(iii) XmlExplorer Controls V1.0.0.0 was used in order to handle XML files, to record models and to handle interoperability with other CASE tools.

Functionalities

VMWare tool allows creating three types of specifications:
(i) Product line models using the FORE notation.
(ii) Product configuration models, in the adequate subset of FORE as described earlier.
(iii) Textual product line constraint specifications.

Our goal is to support the specification of other kinds of models such as goal models, aspect models, etc. A project is a set of several models, one by default. It includes the following functionalities:
1. Create a PLM.
2. Export and import PLM and PLCMs using an XMI file. This functionality allows communicating models from and to other applications.
3. Verify structural and semantic (partially) correctness of product line models.
4. Create and verify PLCMs, compared to a PLM. The set of verified criteria on PLCMs are: root uniqueness, child-father uniqueness, feature existence and PLM's constraint satisfaction.

Example

In VMWare, users can create or open either a project or a specific model. The "verification" menu offers to users the functions that allow choosing the different verification criteria. *Figure 2* gives an example of the feedback provided by the tool after the verification of the structural correctness of a FM. In *Figure 2*, *Feature*

1 and *Feature 2* are mandatory features that are linked by an *excludes*-type relationship.

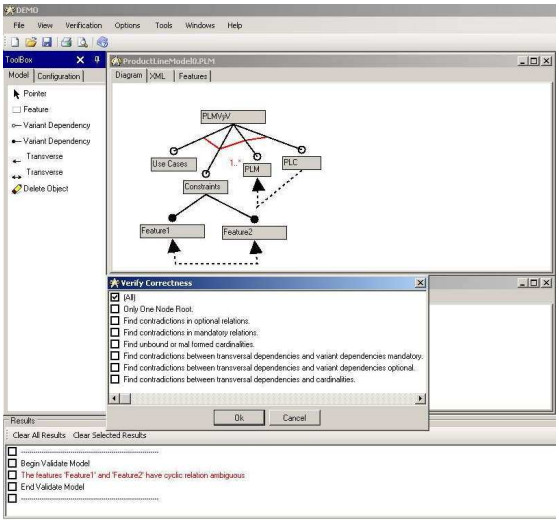

Figure 2. Identification of structural error in a PLM.

In order to verify semantic correctness of a PLM, it is necessary to check: (i) PLCMs' compliance to the corresponding PLM; and (ii) PLM's richness and traceability, uniqueness, pertinence, modifiability and usability of each feature. In order to check PLCMs' compliance, it is necessary to verify the *Feature existence* (every feature in a PLCM 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). At this moment, we are working in formal definition of these criteria; they are not implemented in our tool yet.

4. Conclusions and Future Works

Our goal is to develop a generic method that would automatically help verify any kind of specification based on one or several VMs. We believe that the semantic verification criteria can be defined in a generic level at which any model can be checked. We are currently experimenting the use of constraint languages [6] on top of which these generic semantic verification criteria would be specified. The semantic verification process shall consist in a transformation of the verified model into a constraints program, and in a semantic verification of the constraints program. So far structural verification is concerned, we hope to be able to instantiate meta model-specific verification criteria from an ontology on generic criteria associated to an ontology on general meta-model concepts.

VMWare is not a mature tool yet, and many improvements remain, such us: (i) to support the definition and verification of VMs; (ii) to implant the multi-model verification criteria to validate consistency between PLM and PLCMs as well as between multiple PLMs; (iii) to implant other semantic correctness properties to verify and validate, like traceability, uniqueness, pertinence and modifiability of features and its relationships; and (iv) to support incremental verification.

References

[1] D. Batory, "Feature models, grammars, and propositional formulas", Software Product Lines Conference, LNCS 3714, pages 7–20, 2005.

[2] D. Batory, D. Benavides, and A. Ruiz-Cortés, "Automated analysis of feature models: Challenges ahead", Communications of the ACM, December, 2006.

[3] D. Streitferdt, "Family-Oriented Requirements Engineering", PhD Thesis, Technical University Ilmenau, 2003.

[4] Kim Lauenroth, Klaus Pohl, "Towards Automated Consistency Checks of Product Line Requirements Specifications", ACM/IEEE Intl. Conference on Automated Software Engineering, 2007, pp. 373-376.

[5] M. Antkiewicz, K. Czarnecki, "FeaturePlugin: feature modeling plug-in for Eclipse", OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop, pp. 67-72.

[6] O. Djebbi, and C. Salinesi, "Towards an Automatic PL Requirements Configuration through Constraints Reasoning", Int. Workshop on Variability Modelling of Software-intensive Systems (VaMoS), Essen, Germany, January 2008.

[7] O. Spinczyk, D. Beuche, "Modeling and Building Software Product Lines with Eclipse", International Conference on Object-Oriented Programming, Systems, Languages, and Applications, 2004.

[8] P. Padmanabhan, R. Lutz, "Tool-Supported Verification of Product Line Requirements", Automated Software Engineering, Vol. 12, No. 4, 2005, pp. 447-465.

[9] P. Trinidad, A. Ruiz-Cortés, D. Benavides, S. Segura, A. Jimenez, "FAMA Framework", 12th Int. Software Product Line Conference (SPLC), 2008.

[10] Klaus Pohl, Gunter Bockle, Frank van der Linden, "Software Product Line Engineering: Foundations, Principles and Techniques", Springer, July 2005.

[11] Raul Mazo, Camille Salinesi, "Methods, techniques and tools for product line model verification. Research report", Centre de Recherche en Informatique CRI, Université Paris 1 Panthéon Sorbonne, 2008. In: http://halshs.archives-ouvertes.fr/docs/00/32/36/75/PDF/Methods_Techniques_and_Tools_for_PLM_Verification.pdf

[12] T. von der Maßen, H. Lichter, "RequiLine - A Requirements Engineering Tool for Software Product Lines", Software Product-Family Engineering, Springer LNCS 3014, 2004.

[13] V. Cechticky, A.Pasetti, O. Rohlik, and W. Schaufelberger, "Xml-based feature modelling", LNCS, Software Reuse: Methods, Techniques and Tools: 8th ICSR 2004. Proceedings, 3107:101–114, 2004.