ISSN 22773061

A Petri Net approach for representing Orthogonal Variability Models

Cristian Martinez, Silvio Gonnet, Horacio Leone
INGAR, Instituto de Desarrollo y Diseño, Santa Fe, Argentina
{ocmartinez,sgonnet,hleone}@santafe-conicet.gov.ar

ABSTRACT

The software product line (SPL) paradigm is used for developing software system products from a set of reusable artifacts, known as platform. The Orthogonal Variability Modeling (OVM) is a technique for representing and managing the variability and composition of those artifacts for deriving products in the SPL. Nevertheless, OVM does not support the formal analysis of the models. For example, the detection of dead artifacts (i.e., artifcats that cannot be included in any product) is an exhaustive activity which implies the verification of relationships between artifacs, artifacts parents, and so on. In this work, we introduce a Petri nets approach for representing and analyzing OVM models. The proposed net is built from elemental topologies that represents OVM concepts and relationships. Finally, we simulate the net and study their properties in order to avoid the product feasibility problems.

Keywords: software product line, orthogonal variability model, Petri nets

Council for Innovative Research
Peer Review Research Publishing System
Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY
Vol 9, No 1
editor@cirworld.com
www.cirworld.com, member.cirworld.com

1 INTRODUCTION

Software product line engineering (SPLE) has proven to be the methodology for developing a diversity of software-intensive systems at lower costs, in shorter time, and with higher quality using platforms and mass customization [1]. This is achieved through the management of commonalities and variability in the set of systems' artifacts.

SPLE has two central processes: domain engineering and application engineering.The former is responsible for establishing the reusable platform and thus for defining the variability and the commonality of the product line (PL). The platform consists of all types of software artifacts (requirements, design, realization, tests, etc.). Traceability links between these artifacts facilitate consistent and systematic reuse.The latter process is responsible for deriving product line applications from the platform established in domain engineering. It exploits the variability of the product line and ensures the correct binding of the variability according to the product' specific needs.

Variability can be defined either in a separate variability model or as an integral part of development artifacts. Many contributions have suggested the integration of variability in software development models and diagrams such as class diagrams, feature models and use case diagrams. Nevertheless, it has some disadvantages [2]: the variability spread across different artifacts become almost impossible to keep the information consistent; the single artifacts often leads ambiguous information; an increasing complexity of the software models by adding the variability definitions; a low integration since the concepts used in different kinds of development artifacts differ between them; and the influence of bias in the variability information which come from specific needs of analysis, design, realization, or test artifacts.

Pohl et al. [2] propose a separate model to define the variability of the software product line (SPL). They introduce an Orthogonal Variability Model (OVM) which provides a cross-sectional view of the variability across all software development artifacts. An OVM relates the variability defined to other software development models such as feature models, use case models, design models, component models, and test models.

The idea behind OVM is similar to feature model (FM), but OVM focuses on artifacts relationships whereas FM emphasizes the features decomposition. Therefore we examine some operations (or functions) of automated analysis of feature models (FMs) to be applied in automated analysis of OVM. Kang et al. [3] and Benavides et al. [4] have identified and discussed a complete sets of operations. For this work we only tackle three functions: *detection of dead* nodes, *finding a product*, and *obtaining all products*, however further issues can be covered with our proposal.

(i) Detection of dead nodes: a dead node represents a variation point or variant that never appears in any configuration of a SPL. These unviable nodes lead to inconsistency problems which cause an increase in complexity and a reduction of maintainability.

(ii) Finding a product: this function returns a product which configuration is feasible.

(iii) Obtaining all products: this function returns all possible products. This operation plays a central role during the product line evolution, since all products previously generated must be valid after the changes.

The function (i) refers to a consistency problem while (ii) and (iii) are grouped into satisfiability problem.

Even though these operations will avoid ambiguities and inconsistency, there is still a lack of automated support. Some researches have proposed the use of formal methods to study both FMs [5, 6, 7, 8] and orthogonal variability models [9, 10]. Each one of them use different formalism, e.g., propositional logic and constraint programming. These techniques allow the automation check throughout SPLE, both at early stages of development as well as during evolution.

In this work we will introduce a novel approach for automated support of OVM using Petri nets (PNs). The central idea is to represent the dependencies and constraints within in OVM, and next, analyze the properties of the resulting Petri net.

The remainder of this paper is structured as follows. In Section II we present an overview of OVM meta model and Petri net formalism. Section III describes our approach and the proposed topologies. The formalization of the model is given in Section IV. The approach is applied to an example in Section V. Finally, Section VI is dedicated to conclusions and future works, respectively.

2 OVM META MODEL AND PETRI NETS

In this section, we introduce the Orthogonal Varibility Model technique and Petri nets formalism.

2.1 OVM meta model

An OVM is a model that defines the variability of a software product line. It relates the variability defined to other software development models such as feature models, use case models, design models, component models, and test models [2].

The two central elements of OVM depicted in Fig. 1 are the variation point (VP) and variant (V). A VP documents a variable item (what vary) and a V documents the possible instances of a variable item (how a VP can vary). There are two types of relationships between variation points (VPs) and variants (Vs): variability dependency and constraint dependency.

A *variability dependency* represents that a VP offers a certain V and it is specialized into *mandatory* and *optional*. The former states that a V must beselected for an application if and only if the associated VP is part of the application. The latter defines that a V can (but does not need to) be a part of the application. A set of Vs that are related through an optional variability dependency (to the same VP) can be grouped into an *alternative choice* which are constrained by the cardinality maximum and minimum (min and max in Fig. 1).

A *constraint dependency* documents a restriction that exists between two Vs, a VP and a V, or two VPs; and it is either of the type *requires* or *excludes*. An *excludes* (*requires*) constraint specifies a mutual exclusion (implication) between two elements.

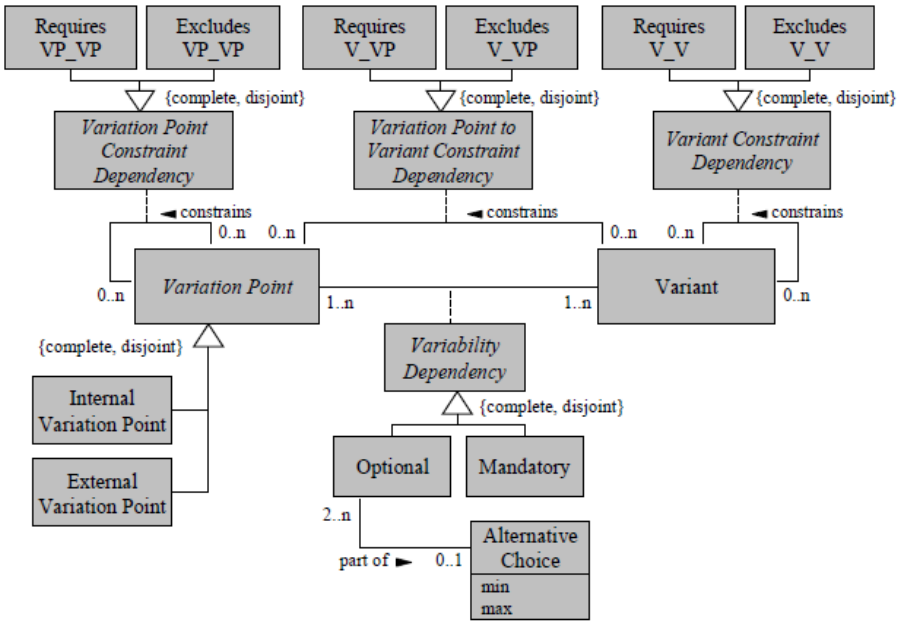


Fig 1: OVM meta model proposed in [2]

2.2 Petri net formalism

Petri nets are a well-known graphical and mathematical modeling tool [11, 12]. A Petri net (PN) is a directed graph consisting of two kinds of nodes, called transitions and places, where arcs are either from a transition to a place or from a place to a transition.In graphical representation, transition are drawn as boxes and places as circles.Arcs are labeled with their weights (positive integers, $\mathbb{N}$), where a *k*-weighted arc can be interpreted as the set of *k* parallel arcs. Labels for unity weight (*k*-1) are omitted.A marking (*M*) assigns to each place a nonnegative integer ($\mathbb{N}_0$). If a marking assigns to place *p* a nonnegative integer *k*, we say that*p* is marked with *k* tokens.

In this work we use the concept of conditions and events, where places represent conditions, and transitions represent events.A transition (event) has a certain number of input and output places representing the pre-conditions and postconditions of the event, respectively. The presence of a token in a place is interpreted as holding the truth of the condition associated with the place.

3 THE APPROACH

In this contribution we introduce a PN approach to represent and study OVMs. The OVM elements and the main activities of application engineering are dealt from an event/condition perspective. To be more precise, the selection of a variation point and the selection of a variant during the product derivation process are represented by the events (transitions), the variation points, variability and constraint dependencies are the pre-conditions (places), and the variants selected are the post-conditions (places).

The goal is to study the behavior of the PN and show the relationships between their markings ($\boldsymbol{M}$) and the valid configurations of the underlying OVM. The interesting $\boldsymbol{M}$ are those which no transitions are enabled (leaf nodes in the reachability graph), in other words, all decisions about the inclusion of variation points and variants have been taken. Although the following PNs belong to trivial OVMs, they can also be combined to support models with increasing complexity.

We briefly introduce the notation used throughout the paper. Given the place $\boldsymbol{p_1}$ represents the variant$\boldsymbol{V_1}$, $\boldsymbol{M}$ the marking, and $\boldsymbol{M(p_1)}$ the number of tokens in $\boldsymbol{p_1}$, whereas $\boldsymbol{M(p_1) = 1}$ depicts the consideration of the variant $\boldsymbol{V_1}$, $\boldsymbol{M(p_1) = 0}$ indicates the no inclusion of $\boldsymbol{V_1}$. The firing sequence $\boldsymbol{\sigma}$ is the chain of events (selections) to reach that marking.

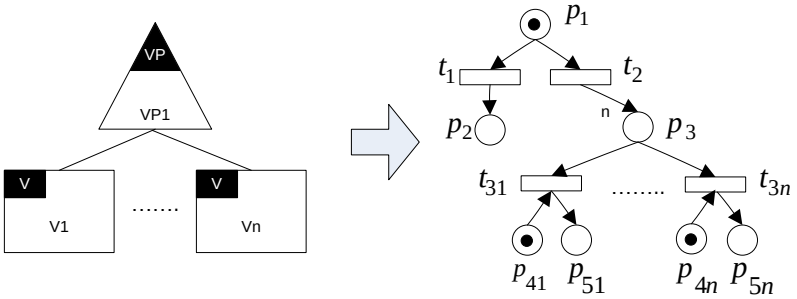
3.1 Variability dependency

A variability dependency is an association between a variation point and a variant, and it is specialized into*mandatory* and *optional* (Fig. 1).

3.1.1 Mandatory

A mandatory dependency states that the consideration of a VP implies the inclusion of the Vs associated to that VP.

In the OVM shown in Fig. 2 (a), the variants $V_1, V_2, \dots V_n$ are associated to VP_1 through mandatory dependencies. There are two configurations $\{\emptyset, (V_1, V_2, \dots V_n)\}$. The former $(\emptyset)$does not include the variation point. The latter configuration $(V_1, V_2, \dots V_n)$ considers VP_1 together with all variants.



(a) (b)

Fig 2: Mandatory dependency. In (a) VP_1 and their mandatory variants $V_1, V_2, \dots V_n$. In (b) the topology proposed.

In the PN illustrated in Fig. 2 (b) the place $p_1$represents VP_1, the transitions t_1 and t_2 the events no-selection and selection of VP_1 respectively. Each t_{3i} corresponds to the selection of the variant i. Places p_{4i} constraint the maximum number of selection of a variant (OVM allows up to 1 instance for each variant), and places p_{5i} indicates the selection of the variant i. Finally, p_3 enables the variant selection transitions.

The only one token in p_1 enables$t_1$and t_2, but exactly one can be fired.Ift_1 fires, no other transitions is enabled and VP_1 is not included. The marking $M(p_{51}) = M(p_{52}) = M(p_{5n}) = 0$ corresponds to the configuration∅.Otherwise, ift_2 fires,$n$tokens are put in p_3 and transitions $t_{31}, \dots t_{3n}$ are enabled.After firing the transitions, $M(p_{51}) = M(p_{52}) = M(p_{5n}) = 1$ which represent the configuration $(V_1, V_2, \dots V_n)$.

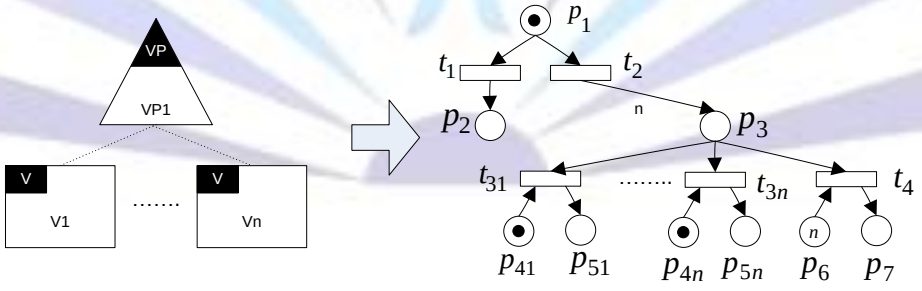
3.1.2Optional

In an optional dependency the consideration of a VP does not imply necessarily the inclusion of the V.

In the OVM shown in Fig. 3 (a), the variation point VP_1 is related through an optional variability dependency to the variants $V_1, V_2, \dots V_n$. The possible configurations are: ∅ and $VP_1 \cup \mathcal{P}(\{V_1, V_2, \dots V_n\})$ ($\mathcal{P}$ indicates the power set).

The topology (Fig. 3 (b)) is similar to the above PN (Fig. 2 (b)) but is extended with the addition of t_4, p_6, and p_7. $t_4$corresponds to the event no-selection of variant, and p_6 and p_7 indicate the maximum event occurrences and the effective occurrences respectively. In this case the number of tokens of p_6 is the amount of variants (n in Fig. 3 (b)).

As stated above after firing t_1 no other transitions is enabled. The resulting marking is $M(p_{51}) = M(p_{52}) = M(p_{5n}) = 0$ and no variants is included (configuration ∅). Otherwise, if t_2 is fired,$n$tokens are put in p_3, and transitions $t_{31}, \dots t_{3n}$ are enabled. After selecting the variants, the rest of tokens in p_3 will be consumed by t_4.



(a) (b)

Fig 3: Optional dependency. In (a) the VP_1 and their optional variants $V_1, V_2, \dots V_n$. In (b) the topology proposed.

3.1.3Alternative choice

An alternative choice groups a set of Vs and defines a range $[m..n]$ for the amount of Vs to be selected, with $m$and n the minimum and maximum cardinality respectively (min and max in Fig. 1).

The topology is similar to the previous PN (Fig. 3 (b)) but the initial marking of$p_6$is$n - m$. It means that at least m variants must be included. An alternative choice will be show in the case study in Section 5.

3.2 Constraint dependency

ISSN 22773061

A constraint dependency documents a restriction that exists between two Vs, between a VP and a V, or between two VPs. Each restriction is either of the type *requires* or *excludes*.

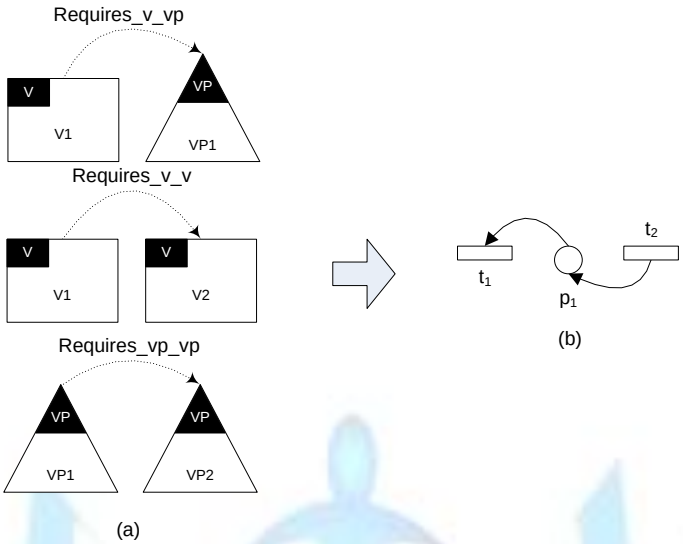


Fig 4: Requires dependency. In (a) the three types of requires constraints and their topology (b).

3.2.1Requires constraint

In a *requires* constraint the consideration of a variation point (or variant) implies the inclusion of another variation point (or variant).

Fig. 4 (a) illustrates the dependencies $Requires\_v\_vp$, $Requires\_v\_v$ and $Requires\_vp\_vp$ and (b) thetopology proposed. The transitions t_1 and t_2 represents the events selection of variation point (or variant) and the place $p_1$corresponds to the constraint. After firing t_1 (event unrestricted), t_2 (event restricted) is enabled.

3.2.2Excludes constraint

A *excludes* constraint indicates that two variation points, two variants or a variation point and a variant are mutually exclusive.

Fig. 5 (a) illustrates the dependencies $Excludes\_v\_vp$, $Excludes\_v\_v$ and $Excludes\_vp\_vp$ and (b) the topology proposed. The transitions t_1 and t_2 represents the events selection of variation point (or variant) and the place p_1 indicates the constraint. The token in p_1 enables t_1 and t_2 but only one can be fired.

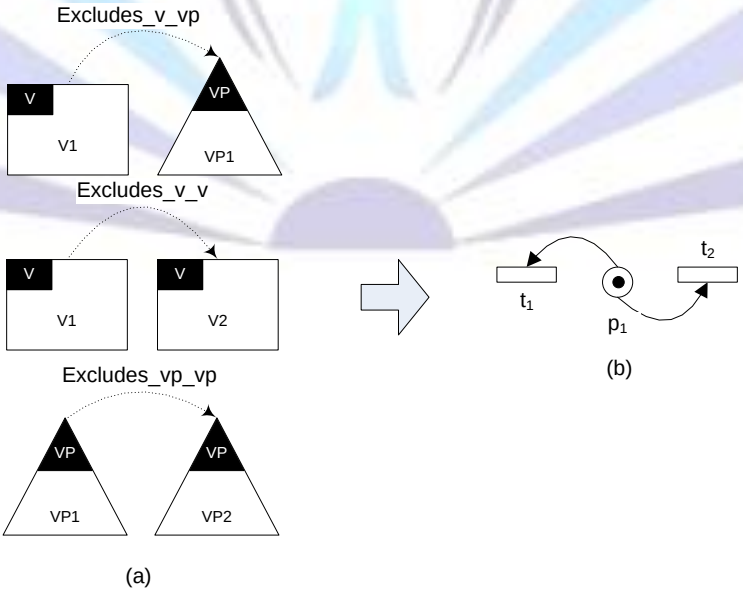


Fig 5: Excludes dependency. In (a) the three types of excludes constraints and their topology (b).

4 CONCEPTS AND PROPERTIES OF THE TOPOLOGY PROPOSED

In this section we provide PNs concepts and properties in terms of OVM.

4.1 Concepts

The meaning of some Petri net concepts helps to understand the relationship between the dynamic of the PN and the configurations allowed by the OVM.

4.1.1Marking

A marking is a*m*-vector where *m* is the total number of places. The *i*-th component of *M* (*M*(i)), is the number of tokens in place *i*. Each *M* depicts a specific product (or configuration) of the product line and tokens indicate which variation points and variants are included.

The interesting markings are those with no transition enabled denoted by dead-end. In these markings all decisions have been made and there are not unresolved variabilities.

4.1.2Token

The presence of a token in a place has several meanings according to the OVM concept associated. In the case of variation point (variant) a token states that the variation point (variant) is included. In a constraint dependency, the token ensures the accomplishment of mutually exclusive (inclusive) constraint disabling (enabling) a transition after firing another one. In the case of cardinality, the number of tokens in a place restrict the maximum or minimum of transition that can be fired, when the place is empty no more variants can be selected.

4.1.3Firing sequence

The firing sequence from the initial marking to a dead-end points out the sequence of transition (or events) to reach a marking. This sequence gives information about the selection of VP and selection of V necessary to set a configuration. Of course, not all sequences are possible since dependencies gives rules, e.g., the selection of a variation point is previous to the selection of their variants.

4.2 Properties

One of the major strength of Petri nets is their support for analysis of problems and properties associated with dynamic systems. We focus on those properties which are closely related with the functions identified in Section I. In the following we will explain how boundedness, reachability and liveness allow to deal with consistency and satisfability problems.

4.2.1Liveness

This property is related with the absence of deadlocks. There are five different levels of liveness, we focus on *L1-live* also called *potencially firable*. A PN is said to be *L1-live* if all transitions can be fired at least once in some firing sequence.

In the topologies proposed each variation point and variant is associated to some transition; therefore if exist at least one firing sequence for any VP and V, there is not any dead node. *L1-live* provides sound basis for function *(i) detection of dead nodes*.

4.2.2Reachability

A reachability graph (RG) of a Petri nets contains all possible reachable markings for a given initiaI marking. Nodes represent markings and its successors and each arc indicates the transitions which transforms one marking to another. This property helps to find all possible configurations of a product line and allows to deal with functions (ii) finding a product and (iii) obtaining all products.

4.2.3Boundedness

The reachability is closed related with the boundedness. A reachability graph of a PN has a finite number of states if and only if the PN is bounded. A net with this property allows to discover all possible products and no overflow occurs.

5 CASE STUDY: ELECTRONIC PAYMENT

The following case study describes partially the variability of a*electronic payment* (e-payment) for a software product line. Initially we introduce the OVM diagram which takes only the payment aspects, then build a Petri net for representing the OVM diagram and we finish studying their properties.

5.1 OVM diagram for electronic payment

The OVM diagram illustrated in Fig. 6 presents the variation points: $e-payment(VP_1)$, $cash\ machine$ (VP_2) and $security(VP_3)$. The variant $debit\ card$ (V_1) is mandatory for VP_1. The variants $https(V_2)$ and $ssl(V_3)$ are optional for VP_3 and defines an alternative choice with range [1..1]. The variants associated to VP_2 does not affect the rest of dependencies so they were not included in the OVM diagram. Finally the constraint dependencies shown the mutually exclusion between VP_2 and V_2 $(Excludes\_vp\_vp)$ and the implication between VP_1 and VP_3 $(Requires\_vp\_vp)$.

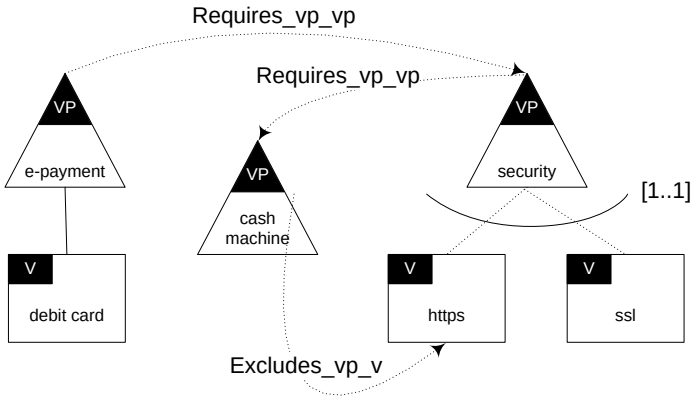


Fig 6: Case study: electronic payment

5.2 PN_{OVM} for electronic payment

The PN for electronic payment is shown in Fig. 7. The places p_1, p_6 and p_8 corresponds to the variation points VP_1, VP_2 and VP_3; p_5, p_{12} and p_{14} represents the variants V_1, V_2, and V_3; p_{15} is the constraint dependency $Excludes\_vp\_v$ (VP_2 - V_2); and p_{17}, p_{16} corresponds to the constraint dependencies $Requires\_vp\_vp$ (VP_1 - VP_3 and VP_3 - VP_2).

The interpretation of the places p_2, p_7, p_9, p_3, p_{10}, p_4, p_{11} and p_{13} is not directly observable from the OVM diagram and their meaning is related to the cardinalities and rules explained in Section III.

The transitions t_2, t_5 and t_6 (t_1, t_4 and t_7) represents the selection (no-selection) of variation point; and t_3, t_8 and t_9 corresponds to the selection of variant V_1, V_2 and V_3 respectively.

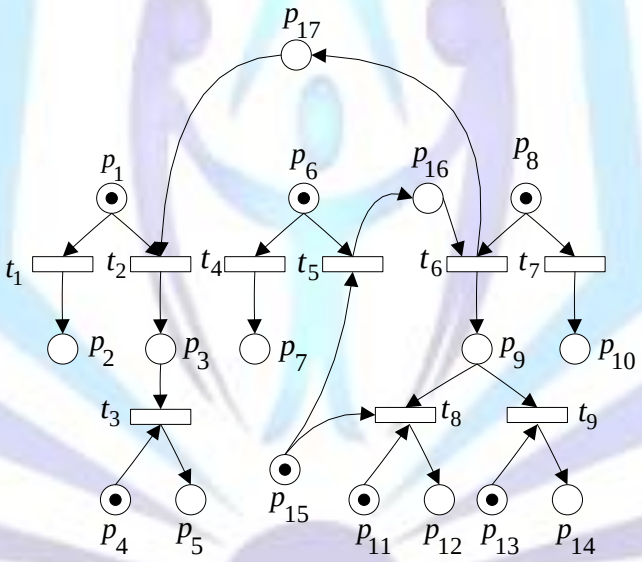


Fig 7: OVM for electronic payment

5.3 Analysis of PN_{OVM} electronic payment using the reachability graph

The Fig. 8 (a) illustrated the reachability graph of the PN_{OVM} *electronic payment*. For simplicity the nodes only described a subset of places, those which corresponds to variation points, variants and constraint dependencies.

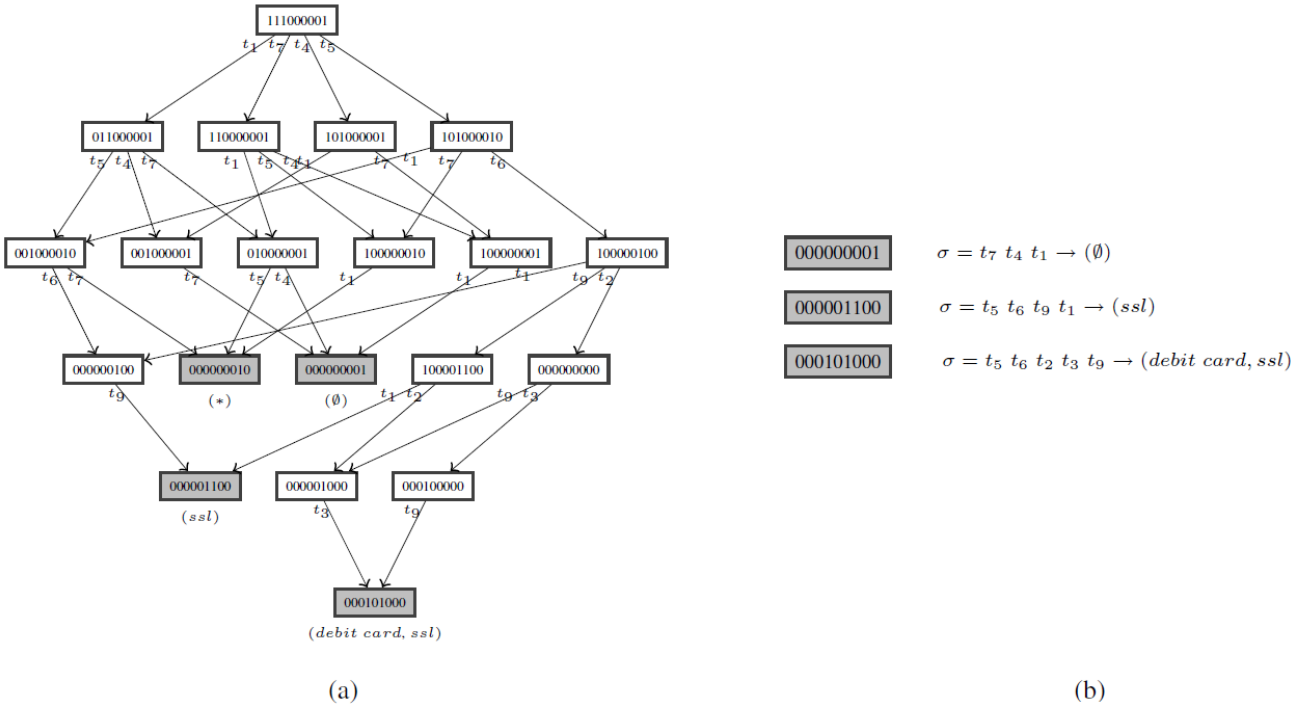


Fig 8:PN_{OVM} of e – payment. (a) shows the reachability graph, and (b) dead-ends and firing sequence.

The top node (111000001) is the initial marking and the terminal nodes (gray scale) are the four affordable configurations in the software product line, for example 000101000 indicates the configuration {*debit card, ssl*}. The reachability graph also provides information regarding the sequence of events to reach such configurations. The independence of certain events (those without any constraint dependency) results in several paths to the same dead-end. Fig. 8 (b) shows some firing sequence and the corresponding dead-end node.

By examining the graph we can get information about the consistency and satisfability of the underlying OVM. All transitions concerning variation points (t_2, $t_5$and t_6) can be fired in some sequence, then all variation points live. However the situation is certainly different in the case of variants, whereas the transitions t_3 and t_9 are firable, t_8 is nonfirable for any firing sequence, thus $https$will never be included during the product derivation.With respect to satisfability, the set of configuration allowed is $\{\emptyset, (ssl), (debit\ card, ssl)\}$ (shown in Fig. 8 (b)). Note that configuration 000000010 (*) is related to VP_2 which is not completely covered in this example.

6 CONCLUSIONS AND FUTURE WORK

Some general problems and operations of feature modeling identified in [3] and [4] can be observed in OVMs as well. In that sense, this paper introduced a Petri net approach for representing and studying OVMs during the development and evolution of a software product line. We defined set of elemental topologies of PNs which dealts the OVM concepts and rules from a event/condition perspective. Then, we focused on Petri net properties and shown that liveness, reachability and boundedness provide a sound basis for analyzing satisfiability and consistency functions mentioned above.Finally, the case study *electronic-payment* was developed using our approach and reported that the variant *debit card* will never be included in any product.

An important challenge is the size of the variability models. Benavides et al. [4] observe an ascendant tendency in the amount of features in last years, from 15 features used in 2004 up to 300 features in 2010. The increasing complexity plays a key role in the evaluation of techniques and tools. Future work will be address toward the use of reduction rules for Petri nets in order to facilitate the analysis by reducing the system model to a simpler one, while preserving the properties.

Another trend is to study other Petri net properties such reversibility and synchronic distance. The reversibility could help us to recover the initial marking given the possible configurations (or products). Synchronic distance is a metric closely related to a degree of mutual dependence between two events, and could provide qualitative information of product line variability.

A third issue is to extend the topologies proposed to feature modeling. Since FMs and OVMs share variability and dependency constraints (e.g., excludes, includes, and alternative choice) we will applied the same event/condition perspective of Petri nets for studying feature models.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the financial support received from CONICET,Universidad Tecnológica Nacional and Agencia Nacional de Promoción Científicay Tecnológica (PAE-PICT 02315).

REFERENCES

[1] Northrop, L., Clements, P. 2009. A framework for Product Line Practive. Version 5.0. Software Engineering Institute. Carnegie Mellon University. http://www.sei.cmu.edu/productlines/frame_report/index.html (ver. 06/24/2013).

[2] Pohl, K., Böckle, G., van der Linden F. 2005. Software Product Line Engineering: Foundations, Principles, and Techniques. Springer Heidelberg.

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

[4] Benavides, D., Segura, S., Ruiz-Cortés A. 2010. Automated analysis of feature models 20 years later: A literature review. Journal of Information Systems 35 (2010) 615-636.

[5] Batory, D. 2005. Feature models, grammars, and propositional formulas. In Software Product Lines Conference.

[6] Sun, J., Zhang, H., Li, Y., Wang H. 2005. Formal semantics and verification for feature modeling. In Proceedings of ICECSS05.

[7] Benavides, D., Ruiz-Cortés, A., Smith, B., O'Sullivan, B., Trinidad, P. 2006. Computational issues on the automated analyses of feature models using constraint programming. International Journal of Software Engineering andKnowledge Engineering.

[8] Benavides, D., Ruiz-Cortés, A., Trinidad, P. 2005. Using constraint programming to reason on feature models. In the 17th International Conference on Software Engineering and Knowledge Engineering.

[9] Metzger, A., Heymans, P., Pohl, K., Schobbens, P., Saval, G. 2007. Disambiguating the Documentation of Variabilityin Software Product Lines. In the 15thIEEE International Requirements Engineering Conference, pp. 243-253.

[10] Lauenroth, K., Pohl, K. 2008. Dynamic Consistency Checking of Domain Requirements in Product Line Engineering. In the 16thIEEE International Requirements Engineering Conference, pp. 193-202.

[11] Murata, T. 1989. Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, Vol. 77:4.

[12] Peterson, J. 1977. Petri Nets. ACM Computing Surveys, Vol. 9:3.