A Petri Net Variability Model for Software Product Lines

Cristian Martinez, Nicolás Díaz, Silvio Gonnet, Horacio Leone

INGAR (UTN - Conicet)
Facultad Regional Santa Fe, Universidad Tecnológica Nacional
{ocmartinez, sgonnet, hleone}@santafe-conicet.gov.ar, nicoe-diaz@gmail.com

Abstract. Variability is defined as the possibility that a system has to be extended, changed, localized or configured in order to be used in a particular context. Variability specification in a software product line (SPL) is a main activity where product families are specified in terms of variants and dependencies. One way of defining the variability of a SPL is through a feature model (FM). However the product families obtained can present feasibility problems, for instance, inclusion rules that can result contradictory which is translated in a set of features impossible to be incorporated into any product. Such inconveniences may come from the initial feature model developed as well from modifications introduced to satisfy new demands. In this paper a tool based on Petri nets is proposed in order to represent and analyse FMs as well as detecting the problems mentioned before.

Keywords: Features, Variability, Petri Nets

1 Introduction

A software product line (SPL) is a set of software systems which share and manage a number of features and are developed from a common set of core assets in a prescribed way [1]. One of its main advantages is the cost reduction which is achieved by reusing components and artefacts among different products. Two main processes take part in the SPL engineering: the domain engineering and the applications engineering [2]. In terms of product line variability, the first one defines it and the second one exploits it selecting the features along the product derivation process. In particular the domain engineering is responsible for defining a basic set of common features and defining the variability of the resulting applications of the SPL. Moreover, the engineering application handles applications derivation starting from the artefacts identified in the previous process. In this process are set the features of the final product.

In order to support these activities is necessary to define the variability of the SPL. There are two approaches to define it: as an integral part of the development artefacts or as a separate model. Pohl et al. [3] state several advantages in favour of the second alternative. Within the group of the second alternatives, the most used method is the Feature-Oriented Domain Analysis (FODA) [4]. FODA focuses on the representation

of the domain's features, and structural relationships between them. FODA proposes a features modelling language (FM), which is then extended by other authors [5].

Since the domain engineering is not only an early stage but rather an ongoing process, the underlying FM must evolve in order to support new requirements. This adaptation to the new requirements results in the addition, elimination or modification of different elements. However, consider that any changes may result in an unviability problem. An unviable configuration indicates that no product can have the feature set of such configuration. This can affect the derivation of future products, invalidate already established and generate confusing and contradictory models.

The simple inspection of a FM [4, 5] in order to detect unviability is a very limited strategy. The complexity of this task is evident since just a few dozen features are necessary for obtaining FMs complex and difficult to examine. Benavides et al. [6] highlight the upward trend over recent years in the number of features available in SPLs. In 2004 models were described in terms of around 15 features, but in 2010 this number was on the order of the 300. This scenario requires analysis tools that provide support in the analysis of SPLs.

Kang et al. [4] and Benavides et al. [6] have identified a complete set of operations that must be performed during the analysis of FMs. Several suggestions have addressed the study of these partial operations [7, 8, 9] through formal methods. These contributions use different formalisms such as propositional logic or constraint programming, and enable a SPL verification. However, none of these contributions provides information about the product derivation process: selection of features to set up a final configuration. Thus, the main objective of this paper is to introduce an approach based on Petri nets (PN) [10] for the representation and evaluation of FM as dynamic systems. In this proposal we consider three important functions related to the problem of unviability exposed before: (1) dead nodes detection, (2) obtaining a product, and (3) obtaining all products.

- Dead nodes detection: a dead node represents a feature that never appears in the SPL configuration. The inconsistency introduced by these unviable nodes adds complexity to the SPL and makes it difficult to maintain.
- Obtaining a product: this allows us to obtain a product whose configuration is feasible.
- Obtaining all products: this function allows us to identify all possible products. It is of critical importance in the evolution of a SPL, since all previously generated products must be valid after the changes.

The first function addresses a consistency issue in SPLs, on the other hand the other two functions are related with satisfiability problems. Taking this as a starting point, we represent FM in terms of PN. From the obtained network is possible to analyse its properties providing support to the three functions recently introduced. Unlike the proposals [7, 8, 9] and the ones discussed in [6], our work is focused on the activities and their precedence. For each valid configuration is possible to find the sequence of decisions regarding the inclusion and exclusion of features. In addition, the proposed model has the simulation and analysis ability characteristic of PNs.

A synthesis of the elements of a feature model together with the PN formalism is included in the following section. In Section III, the proposed topologies used to build a PN from the basic elements of a FM are described. After that, in Section IV, those topologies are formalized and defined. Section V and VI presents and analyses a case of study. Finally, Section VII discusses conclusions and further research.

2 Feature models and Petri nets

2.1 Feature models

A feature model represents the information of all the possible products in a SPL through modelling the features of its domain and the structural relations between them [4]. In a FM, features are connected in a tree hierarchy, where a parent feature (or composed feature) is connected with its child features (or sub-features) through the following variability dependency relationships:

- *Mandatory*. Relationship between a parent feature and a child feature which indicates that the child feature appears in every product where its parent feature is included.
- *Optional*. Relationship between a parent feature and a child feature which indicates that the child feature may or may not be included in the products where its parent feature is included.
- *Alternative or group cardinality*. Relationship between a parent feature and a group of child features. The cardinality of the group is represented by an interval [*min..max*] which limits the number of child features that can be included in a product when the parent feature is included. The min and max values represents the minimum and maximum amount of features that may be included. OR and XOR relations are particular cases where cardinality assumes the values [1..max] and [1..1] respectively.

In a FM is also possible to define restrictions between features through the hierarchy tree:

- *Require*. Relationship that conditions the inclusion of a feature by selecting another feature.
- *Exclude*. Relationship that indicates that two features are mutually exclusive.

2.2 Petri Nets formalism

PNs are a mathematical modelling tool that allows studying events dynamics together with pre-conditions and post-conditions [10]. A PN is a directed graph with two node types: transitions and places. A transition (event) is associated to a set of input and output places through arcs, representing the preconditions and post conditions of the events respectively. In Table 1 there is a formal definition of the previous.

Table 1. Formal Petri net definition.

A Petri net is a 5-tuple (P, T, A, W, M_0) where: $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places, $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions, $A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relationship), $W: A \rightarrow \mathbb{N}$ is a weight function, $M_0: L \rightarrow \mathbb{N}_0$ is the initial marking, $P \cap T = \varnothing$ and $P \cup T \neq \varnothing$.

3 Representing variability using Petri nets

In this section the topologies proposed for representing the elements of a FM using PNs are defined. The elements of a FM together with the application engineering core activities can be considered depending on the "event/condition" point of view of PN modelling. The *selections of features* during the product derivation are related to *events* in the application engineering process and they are represented as transitions in a PN. Features, variability dependencies and restriction dependencies are preconditions for these events and are represented as places in a PN. Finally the included features are the result of events occurrence, this is, post-conditions also are represented as places. In Table 2 are resumed the proposed relations between FM elements, application engineering process activities, and the elements of a PN. In parenthesis there are included the generic names assigned for each element.

Table 2. Relation between FM and Petri Net elements.

FM elements	**Petri Net elements**
Feature (f_i)	1 Place (*PfiSelected*) and 1 Arc
Feature selection (f_i)	1 Transition (*TSelectionfi*)
Root feature (f_0)	2 Places (*P0, Pf0NotSelected*), 3 Arcs and 1 Transition (*TNoSelectionf0*)
Mandatory (f_1 —● f_2)	1 Place (*Pf1mandatoryf2*) and 2 Arcs
Optional (f_1 —o f_2)	2 Places (*Pf1optionalf2*, *Pf2NotSelected*), 4 Arcs and 1 Transition (*TNoSelectionf2*)
Alternative (f_p [min,max] — f_1 .. f_n)	3 Places (*PfpalternativeIf1fn*, *PNoSelectionfpAlternativef1fn*, *Pf1fnNotSelected*), 4 Arcs and 1 Transition (*TNoSelectionfpAlternativef1fn*) For each alternative: 1 place (*PMaxCardfi*) and 1 arc
Requires (f_1 require f_2)	1 Place (*Pf1requires f2*) and 2 Arcs
Excludes (f_1 excludes f_2)	1 Place (*Pf1excludesf2*) and 2 Arcs

We will call PN_{FM} to the resultant PN. From the Petri net PN_{FM} is possible to study its dynamic and show the relationship between its marks and the possible configurations of the underlying FM. The marks that interest us are the ones which have no enabled transitions (t), this is that all feature selection decisions have been already resolved. From this marks, the places associated with features are representing a potential configuration of the SPL.

Bellow we introduce the used notation. Considering p_1 as a place which represents a feature f_1, M as the net marking in an specific instant of time, and $M(p_1)$ the number of marks of p_1 in M, $M(p_1)$=1 represents the selection of a feature f_1, y $M(p_1)$=0 indicates the non-inclusion of the f_1 feature. The trigger sequence σ is the list of events (selections) needs to reach M. The following notation is used to denote certain pre-sets and post-sets:

$\bullet t = \{p \mid (p, t) \in A\}$, set of input places of t,
$t\bullet = \{p \mid (t, p) \in A\}$, set of output places of t,
$\bullet p = \{t \mid (t, p) \in A\}$, set of input transitions of p,
$p\bullet = \{t \mid (p, t) \in A\}$, set of output transitions of p.

3.1 Root feature of the FM

The root feature, f_0, of the FM is represented with the PN as is shown in Fig. 1. The PN includes a place *P0* which is originally marked, this is $M_0(P0)$=1 (M_0 is the initial marking). The mark in *P0* (Fig. 1) enables the transition *TNoSelectionf0* and *TSelectionf0*, this transitions model the no-selection and selection of the f_0 feature, respectively. This model allows two different configurations $\{\varnothing, \{f_0\}\}$. The first one ($\varnothing$) does not consider the root feature f_0 and is represented by $M_i(Pf0NoSelected)$= 1, i >0. On the other hand, the second configuration ($\{f_0\}$) considers the selection of f_0, therefore $M_i(Pf0Selected)$= 1, for all i >0.

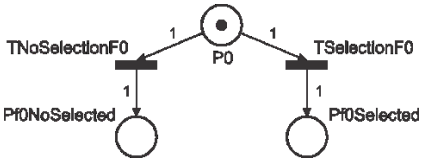

Fig. 1. Petri net proposed to represent the root feature, f_0, of a FM.

3.2 Mandatory variability dependency

As it has been explained in Section 2.1, a mandatory variability dependency (Fig. 2(a)) between a feature f_1 and a feature f_2 indicates that the consideration of feature f_1 is enough condition for including feature f_2. The Fig. 2 (a) shows a FM with feature f_1 linked to feature f_2 by this type of variability dependency. Two possible configurations $\{\varnothing, \{f_1, f_2\}\}$ exists for this model. The first one ($\varnothing$) does not consider feature f1, on the other hand the second one ($\{f_1, f_2\}$) as is considering f_1 includes mandatorily f_2.

Fig. 2. Mandatory variability dependency. (a) FM including the mandatory feature (f_2). (b) Proposed Petri net.

The proposed PN shown in Fig. 2 (b) represents the mandatory variability dependency through the place *Pf1mandatoryf2* and the arcs which link transitions *TSelectionf1* y *TSelectionf2*. These transitions correspond to the selection of features f_1 and f_2 respectively. The place *Pf2Selected* is representing the feature f_2 as part of the final product configuration when $M_i(Pf1Selected)= 1$, for some $i >0$.

3.3 Optional variability dependency

In an optional relationship (Fig. 3 (a)) between a feature f_1 and a feature f_2, the selection of the feature f_1 does not impose the inclusion of the feature f_2. This model allows two different configurations $\{\varnothing, \{f_1\}, \{f_1, f_2\}\}$.

The proposed PN (Fig. 3 (b)) represents the optional variability dependency trough the place *Pf1optionalf2* and the arcs which link the transitions *TSeleccionf1*, *TSeleccionf2*, y *TNoSeleccionf2*. These transitions represent the selection of the features f_1 and f_2, or the selection of the feature f_1 and non-selection of feature f_2, respectively. The place *Pf2Selected* represents a final configuration which includes f_2 when $M_i(Pf2Selected)= 1$, for some $i >0$. On the other hand, a mark in *Pf2NoSelected* represents the decision of not including the optional feature f_2.

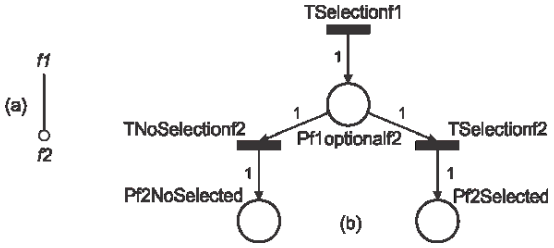

Fig. 3. Optional variability dependency. (a) FM including the optional feature (f_2). (b) Proposed Petri net.

3.4 Alternative variability dependency

An alternative selection establishes that a selected feature f_p (Fig. 4 (a)) is sufficient condition for the inclusion of at least *min* and at last *max* alternative features f_i, $i=1..n$ (Fig. 4 (a)). For instance, if n=2, *min*=0, y *max*=1 the set of possible configurations would be $\{\varnothing, \{f_p\}, \{f_p, f_1\}, \{f_p, f_2\}\}$.

The proposed PN topology is shown in Fig. 4 (b) and it represents the alternative variability dependency through the place *Pfpalternativef1fn* and the arcs which link it with the transitions *TSelectionfp*, *TSelectionf1*, *TSelectionf2*, …, *TSelectionfn*, and *TNoSelectionfpAlternative1fn*. The arc that link *TSelectionfp* with *Pfpalternativef1fn* has a weight equals to *max*. This arc establishes the maximum number of selection of alternative features f_i, this enables the transitions *TSelectionf1*, *TSelectionf2*, …, *TSelectionfn*.

The place *PfjSelected* ($j= 1..n$) represents a feature f_j as part of the configuration of the final product when $M_i(PfjSelected)= 1$, for some $i >0$. The number of times that

an alternative feature can be selected is determinate by the places *PCardMaxf1*, *PCardMaxf2*, …, *PCardMaxfn*, which are marked in M_0. On the other hand, each mark in *Pf1fnNoSelected* represents the decision of not including an alternative feature and the place *PNoSelectionfpAlternativef1fn*, with an initial marking equals to a *max* – *min*, restricts the number of non-selections that can be done if f_p is selected. If we consider again *n*=2, *min*=0, y *max*=1 the possible configurations would be:

- σ = *TSelectionfp TSelectionf1*. The resulting marking is: *M*(*Pf1Selected*)=1, *M*(*Pf2Selected*)=0, *M*(*Pf1f2NoSelected*)=0, and this represents the configuration $\{f_p, f_1\}$.
- σ = *TSelectionfp TSelectionf2*. The resulting marking is: *M*(*Pf1Selected*)=0, *M*(*Pf2Selected*)=1, *M*(*Pf1f2NoSelected*)=0, and this represents the configuration $\{f_p, f_2\}$.
- σ = *TSelectionfp TNoSelectionfpAlternativef1f2*. The resulting marking is: *M*(*Pf1Selected*)=0, *M*(*Pf2Selected*)=0, *M*(*Pf1f2NoSelected*)=1, and this represents the configuration $\{f_p\}$.

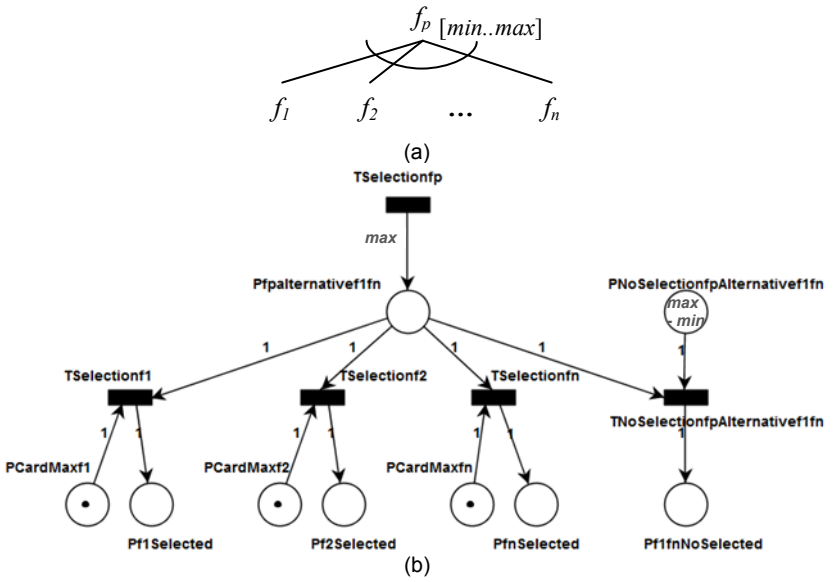

Fig. 4. Alternative variability dependency. (a) FM including *n* alternative features. (b) Proposed Petri net.

3.5 Requires dependency restriction

In a requires dependency the consideration of a feature is attached to the inclusion of it associated feature. Fig. 5 (a) shows that feature f_1 *requieres* feature f_2. The proposed topology in this case is shown in Fig. 5 (b). Transitions *TSelectionf1* and *TSelectionf2* correspond to the selection events of features f_1 and f_2, respectively. The place *Pf1requieresf2* represents this restriction. By triggering *TSelectionf2* (conditioning

event) a mark is put in *Pf1requieresf2* enabling transition *TSelectionf1* (conditioned event).

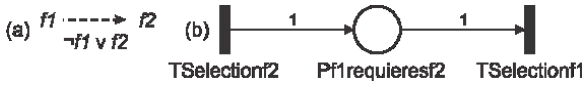

Fig. 5. Requires dependency restriction. (a) FM where feature f_1 requires feature f_2. (b) Proposed Petri net.

3.6 Excludes dependency restriction

An excludes dependency establishes that two features are mutually exclusive. Fig. 6 (a) shows an *excludes* relationship between features f_1 and f_2. In Fig. 6 (b) is specified the proposed topology for this case. Transitions *TSelectionf1* and *TSelectionf2* are related to the selection events of features f_1 and f_2, respectively. The place *Pf1excludesf2* represents the excludes restriction. The mark in *Pf1excludesf2* enable transitions *TSelectionf1* and *TSelectionf2*, but only one of them can be triggered.

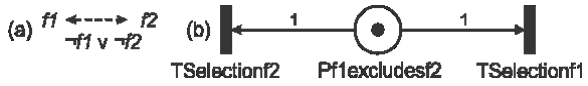

Fig. 6. Excludes dependency restriction. (a) FM representing the excludes relationship between two features. (b) Proposed Petri net.

4 PN_{FM} formal definition, concepts and properties

In this section we will formally define the PN_{FM} net together with the interpretation of concepts and properties of PNs in the FM context.

4.1 PN_{FM} formal definition

In Table 3 a PN_{FM} is defined based in a *FM*. The notation proposed represents the set of places with P, the set of transitions with T and the set of arcs with A. We split the places and transitions sets P and T into subsets according to its interpretation. Some of these subsets represent explicitly the elements of a FM. These sets are P^F (features), P^M (mandatory), P^O (optional dependency), P^G (alternatives dependency), P^E (exclude restriction), and P^Q (require restriction). The other subsets of places are related to representing the cardinality, the possibility of non-selection of a feature and the root of a FM. Each place in P^C is defined to indicate the maximum number of times that a certain feature can be included, in this case the value is always 1. P^A specifies the minimum number of features that must be selected (or the maximum number of non-selections). Transitions in T^F are representing the selection of features. Instead, transitions in T^N are specifying the non-selection of features; and are used to disable transitions in the PN_{FM}.

Conditions described in Table 4 allow representing FM rules associated with the initial marking and the income and outcome arcs from and to the places and transitions. For instance, place $P0$ ($\in P^R$) does not have any income transaction ($\bullet P0 = \varnothing$) and the number of marks on its initial marking is 1 ($M_0(P0) = 1$). The outcome transactions must be $P0\bullet = \{TSelectionf0, TNoSelectionf0\}$ and are related with the selection and no-selection of the root feature of the FM.

Table 3. PN_{FM} formal definition.

For a feature model FM defined as (F, r, M, O, G, C, E, Q), where: $F = \{f_1, f_2, \ldots, f_n\}$ is a finite set of features, $r \in F$, is the root feature of the FM, $M \subseteq F \times F$ is a set of mandatory dependency relations, $O \subseteq F \times F$ is a set of optional dependency relations, $G \subseteq F \times \mathbb{P}(F)$ is a set of alternative dependency relations, $C: G \to \mathbb{N} \times \mathbb{N}$ is a cardinality function, $E \subseteq F \times F$ is a relation set of exclusive restrictions, $Q \subseteq F \times F$ is a relation set of require restrictions.
A PN_{FM} is defined as a 5-tuple (P, T, A, W, M_0), where: $P = P^F \cup P^R \cup P^M \cup P^O \cup P^G \cup P^N \cup P^C \cup P^A \cup P^E \cup P^Q$ $T = T^F \cup T^N$ $A \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs, $W: A \to \mathbb{N}$ is a weight function, $M_0: P \to \mathbb{N}_0$ is the initial marking, $P \cap T = \varnothing$ and $P \cup T \neq \varnothing$.
where $P^F = \{PfiSelected \mid i = 1..n\}$ selected features $P^R = \{P0\}$ root $P^M = \{Pfimandatoryfj \mid (f_i, f_j) \in M\}$ mandatory dependencies $P^O = \{Pfioptionalfj \mid (f_i, f_j) \in O\}$ optional dependencies $P^G = \{Pfpalternativef1fk \mid (f_p, A) \in G, A = \{f_1, \ldots, f_k\} \subset F\}$ alternative dependencies $P^N = \{PfjNoSelected \mid f_j = r \vee (\exists f_i \mid (f_i, f_j) \in O)\} \cup \{Pf1fkNoSelected \mid (\exists f_p \mid (f_p, A) \in G, A = \{f_1, \ldots, f_k\} \subset F)\}$ non-selected features (root, optional or alternative) $P^C = \{PCardMaxfj \mid (\exists f_p \mid (f_p, A) \in G, f_j \in A)\}$ maximum cardinality of each feature in a group $P^A = \{PnoSelectionfpAlternativef1fk \mid (f_p, A) \in G, A = \{f_1, \ldots, f_k\} \subset F)\}$ maximum cardinality of non-selectable features in a group $P^E = \{Pfiexcludesfj \mid (f_i, f_j) \in E\}$ excludes restrictions $P^Q = \{Pfirequiresfj \mid (f_i, f_j) \in Q\}$ requires restrictions $T^F = \{TSelectionfi \mid i = 1..n\}$ feature selection event $T^N = \{TNoSelectionfj \mid f_j = r \vee (\exists f_i \mid (f_i, f_j) \in O)\} \} \cup \{TNoSelectionfpAlternativef1fk \mid (\exists f_p \mid (f_p, A) \in G, A = \{f_1, \ldots, f_k\} \subset F)\}$ feature non-selection event (root, optional o alternative)

4.2 PN concepts applied to a FM

The meaning of some PNs concepts allows us to understand the relationship between the PN dynamic and the allowed configurations on a FM. In the next paragraphs we analyses the concepts of marking, mark and firing sequence.

- Marking. Is an *n*-vector where *n* is the total number of places in the PN. The ith component of *M* (*M*(*i*)) represents the number of marks in the *i* place. Each *M* describes a specific product (or configuration) of a SPL and the marks indicate which features are included. The markings that we are interested in are the ones without enabled transitions, this is, those where all decisions have been made during the configuration process.
- Mark. The presence of a mark in a place has different meanings depending on the associated FM concept. In the case of a feature, a mark in a place P^F_i ($P^F_i \in P^F$) indicates that the represented feature by that place is included in the configuration. For a dependency restriction, the mark ensures that a mutually exclusive (inclusive) restriction disables (enables) a transition at the moment that another transition of the restriction is triggered. In the cardinality case, the number of marks in a place restricts the maximum or the minimum number of transactions that can be triggered; when a place is empty it is not possible to select any more features.
- Firing sequence. The firing sequence from the initial marking M_0 to a marking where does not exist an enabled transition represents the event sequence to reach a certain configuration. This sequence gives us the information about the feature selection to build a possible configuration in a SPL.

Table 4. Imposed conditions in Places and Transitions of a PN_{FM}.

Places conditions:
$\bullet P0 = \varnothing$, $P0\bullet = \{TSelectionf0, TNoSelectionf0\}$, $M_0(P0) = 1$, $P0 \in P^R$
$\bullet p \subseteq T^F$, $\lvert\bullet p\rvert = 1$, $p\bullet = \varnothing$, $M_0(p) = 0$, $\forall\, p \in P^F$
$\bullet p \subseteq T^F$, $\lvert\bullet p\rvert = 1$, $p\bullet = \subseteq T^F$, $\lvert p\bullet\rvert = 1$, $M_0(p) = 0$, $\forall\, p \in P^M$
$\bullet p \subseteq T^F$, $\lvert\bullet p\rvert = 1$, $p\bullet = \subseteq T^F \cup T^N$, $\lvert p\bullet\rvert = 2$, $M_0(p) = 0$, $\forall\, p \in P^O$
$\bullet p \subseteq T^F$, $\lvert\bullet p\rvert = 1$, $p\bullet = \subseteq T^F \cup T^N$, $\lvert p\bullet\rvert > 1$, $M_0(p) = 0$, $\forall\, p \in P^G$
$\bullet p \subseteq T^N$, $\lvert\bullet p\rvert = 1$, $p\bullet = \varnothing$, $M_0(p) = 0$, $\forall\, p \in P^N$
$\bullet p = \varnothing$, $p\bullet \subseteq T^F$, $\lvert p\bullet\rvert = 1$, $M_0(p) = 1$, $\forall\, p \in P^C$
$\bullet p = \varnothing$, $p\bullet \subseteq T^N$, $\lvert p\bullet\rvert = 1$, $M_0(p) > 0$, $\forall\, p \in P^A$
$\bullet p = \varnothing$, $p\bullet \subseteq T^F$, $\lvert p\bullet\rvert = 2$, $M_0(p) = 1$, $\forall\, p \in P^E$
$\bullet p \subseteq T^F$, $\lvert\bullet p\rvert = 1$, $p\bullet \subseteq T^F$, $\lvert p\bullet\rvert = 1$, $M_0(p) = 0$, $\forall\, p \in P^Q$
Transitions conditions:
$\bullet t \subseteq P^R \cup P^M \cup P^O \cup P^G \cup P^Q \cup P^E$, $\lvert\bullet p\rvert \geq 1$, $t\bullet = P^F \cup P^M \cup P^O \cup P^G \cup P^Q$, $\lvert p\bullet\rvert \geq 1$, $\forall\, t \in T^F$
$\bullet t \subseteq P^R \cup P^O \cup P^G \cup P^A$, $\lvert\bullet p\rvert \geq 1$, $t\bullet = P^N$, $\lvert p\bullet\rvert = 1$, $\forall\, t \in T^N$

4.3 PN_{FM} properties

In this work we focused in those properties of the PN_{FM} which are related to the operations identified in Section 1. Next, we will explain how boundedness, reachability,

and potentially triggerable ("L1-live") properties allow addressing problems of consistency and satisfiability of a FM in a SPL.

- Boundedness. A PN is *k*-bounded if, from an initial marking M_0, for any reachable state the number of marks in any place is not greater than *k*. In a non-bounded Petri net, regardless of the firing sequence the upper limit of the marks in any of the places cannot be specified. A PN_{FM} is *k*-bounded, where *k* is the maximum value of the maximum cardinality of the alternative dependencies. If the model does not have alternative dependencies the bound is equals to 1.
- Reachability. Given a PN with an initial marking M_0, M_n is reachable if exists a firing sequence σ which enables to reach M_n from M_0. One way of analysing this property is through a reachability graph. A reachability graph is a directed graph where nodes represent the markings and arcs the transition from one marking to its successor. The initial node corresponds to the initial marking (M_0), the intermediate nodes to the ones with at least one transition enables and the terminal nodes to those which does not have any transition enabled. Reachability graph in *k*-bounded Petri nets are finite [10]. By analysing a *k*-bounded PN_{FM}, it is possible to obtain all the possible products (possible configurations) that are derivable from a *FM*.
- L1-live or potentially triggerable. The liveness property is associated with the absence of deadlocks or dead nodes. A live PN guarantees that is possible to find a sequence for trigger any transition. This property ensures the complete absence of dead nodes. There are several liveness levels [10]. The L1-live level, also called potentially triggerable, establishes that a transition can be triggered in a sequence at least once. If all transitions accomplish the L1-live level, then the PN is L1-live.

Taking the two first properties as a starting point and as a PN_{FM} is bounded, we can confirm that its reachability graph is finite. The nodes of the reachability graph represent all the possible configurations of the SPL, but the ones we are interested in are the terminal nodes. These nodes represent configurations which does not have any pending decisions related to the inclusion or non-inclusion of features. This is useful in order to identify all possible configurations in a SPL and to provide support to *find a product* and *get all possible products*. Moreover, if a change in the FM takes place (and its PN_{FM}), the resulting graph can be compared with the graph of the previous net in order to detect viable configurations in the previous model and non-viable ones in the new model.

The L1-live property is useful to address the *detection of dead nodes*. If every transition is potentially triggerable, any feature will be included in at least one configuration, thus the FM will not have unviable features.

5 Case study

The case study includes in this section partially describes the variability of a mobile phones SPL. In first place we will introduce the FM, then we will generate the PN_{FM} to represent the FM and analyse its properties.

Fig. 7 partially illustrates the FM of the case study. Due to space limitations, we include only 7 features. The model was built using SPLOT [11]. The root feature is *MPhone*. If this feature is selected, features *Calls* and *Screen* must also be considered because there are linked to *MPhone* through a mandatory dependency (fill circle). Instead, *GPS* and *Camera* are optional features (empty circle). *Basic* and *HRes* are alternatives of the *Screen* feature but only one of them can be selected (cardinality [1, 1] in Fig. 7). The features *GPS* and *Basic* are mutually exclusive and the selection of *Camera* will require the inclusion of *HRes* (bottom part of Fig. 7).

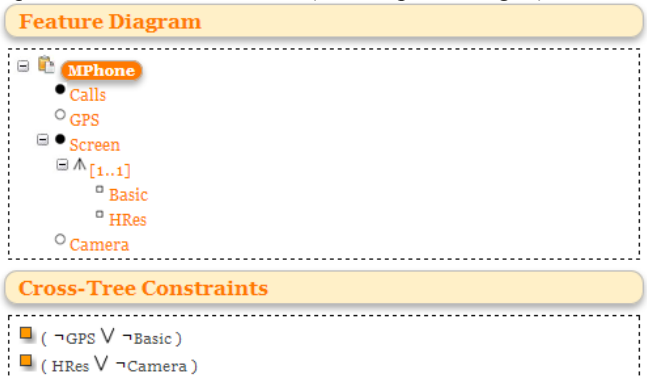

Fig. 7. Partial mobile phone FM.

Fig. 8 represents the PN_{FM} net obtained from the FM illustrated in Fig. 7. For building the PN_{FM} net according to the topology described in Section 3 we have developed a tool called FM2PN. This tool generates a Petri net PN_{FM} from a FM created with SPLOT. FM2PN was built using the JLex lexical analyser [12] and the parser generator CUP [13], both of them based on Java. For the study case a net with 22 places, 11 transitions and 34 arcs was obtained.

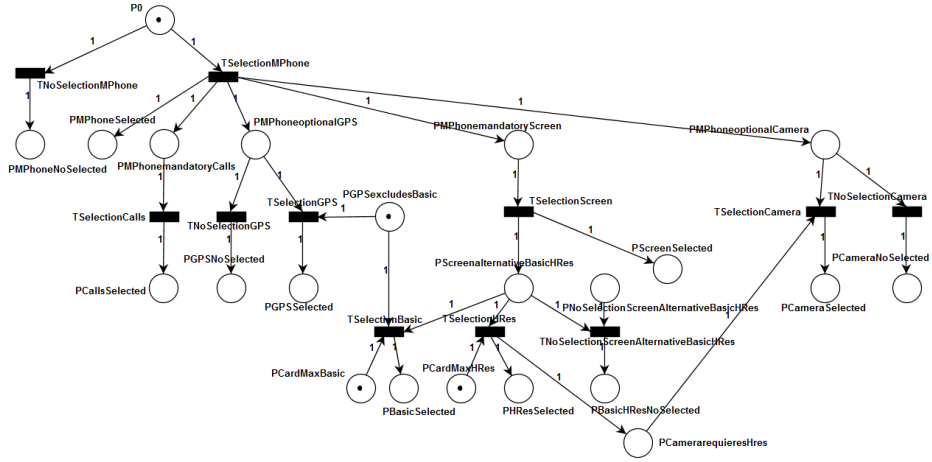

Fig. 8. PN_{FM} of the mobile phone FM.

The *MPhone* feature is represented by the place *PMPhoneSelected* (Fig. 8) and the non-selection of this feature is represented by *PMPhoneNoSelected*. Mandatory features *Calls* and *Screen* are modelled by the places *PCallsSelected* and *PScreenSelected*, respectively. For the optional features *GPS* and *Camera,* the places *PGPSSelected* and *PCameraSelected* where included to represent their selection, and the places *PGPSNoSelected* and *PCameraNoSelected* to represent the non-selection. The excludes restriction ($\neg GPS \vee \neg Basic$ in Fig. 7) is represented by the place *PGPSexcludesBasic* and the requires restriction ($HRes \vee \neg Camera$ in Fig. 7) by the place *PCamerarequieresHres*. Transitions *TSelectionMPhone*, *TSelectionCalls*, *TSelectionGPS*, *TSelectionScreen*, *TSelectionBasic*, *TSelectionHRes*, and *TSelectionCamera* (*TNoSelectionMPhone*, *TNoSelectionGPS*, *TNoSelectionScreenAlternativeBasicHRes*, and *TNoSelectionCamera*) represent the selection (non-selection) of the features.

FM2PN generates the PN_{FM} net into an xml file. This net is analysed using PIPEv2 (Fig. 8, Fig. 9, Fig. 10) in order to detect the unviability problems. PIPEv2 [14] allows to build, among other things, the reachability graph (Fig. 9) of a certain PN and simulate the net as well (Fig. 10).

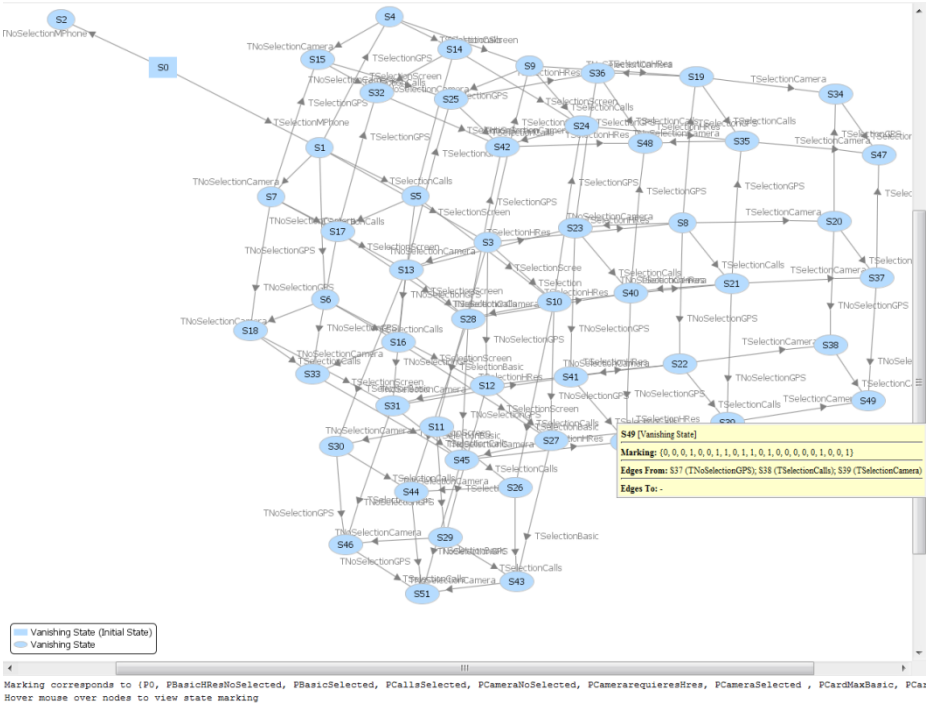

Fig. 9. Partial view of the reachability graph.

The node *S0* (Fig. 9, represented as a rectangle) represents the initial marking (M_0) and the terminal nodes are representing the different configurations in the SPL, for instance the node *S49* in Fig. 9, has the marking {0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0,

0, 0, 0, 0, 1, 0, 0, 1} where the marked places are: *PCallsSelected, PCameraSelected, PCardMaxBasic, PGPSExcludesBasic, PGPSNoSelected, PHResSelected, PMPhone-Selected,* and *PScreenSelected*. This marking represents the selection of features *MPhone*, *Calls*, *Camera*, *HRes*, and *Screen*, this is, the configuration {*MPhone*, *Calls*, *Camera*, *HRes*, *Screen*}. Moreover, the reachability graph provides information about the event sequence that achieved this configuration. Especially, one of the possible sequences that allows to reach the configuration represented by the node *S49* is given by the events σ = *TSelectionMPhone TNoSelectionGPS TSelectionCalls TSelection-Screen TSelectionHRes TSelectionCamera*, this triggering sequence is illustrated in Fig. 10, highlighting the final marking.

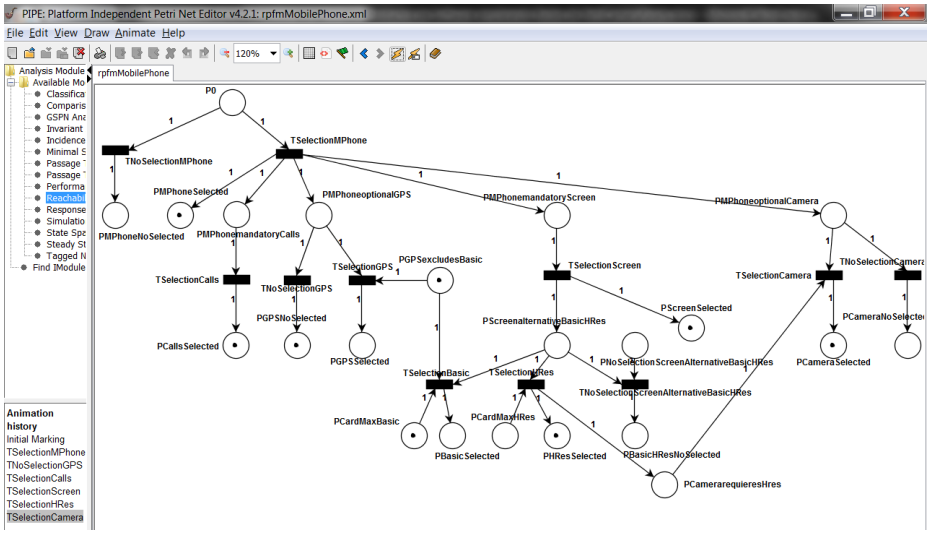

Fig. 10. Resulting marking after applying σ = *TSeleccionMPhone TNoSeleccionGPS TSelec-cionCalls TSeleccionScreen TSeleccionHRes TSeleccionCamera* in M_0.

Regarding the satisfiability, the set of allowed configurations is {∅, {*MPhone*, *Calls*, *Screen*, *Basic*}, {*MPhone*, *Calls*, *Screen*, *HRes*}, {*MPhone*, *Calls*, *Screen*, *HRes*, *Camera*}, {*MPhone*, *Calls*, *Screen*, *GPS*, *HRes*}, {*MPhone*, *Calls*, *Screen*, *GPS*, *HRes*, *Camera*}}.

6 FMTEST

We need to analyze the PN_{FM} to determine the collection of feasible configurations of the SPL. For this reason, it was decided to develop a tool that could generate the reachability graph and perform an automatic analysis based on the study of the Petri nets dynamics. This transformation to a PN_{FM} and the analysis of the reachability graph should be carried out without the participation of the SPL analyst. In the next section is presented an algorithm for building a reachability graph and its implementation in the tool FMTEST.

6.1 Reachability Graph

In this section is introduced the algorithm for building the reachability graph. Algorithm 1 defines the initial state of the graph, thus is M_0 of PN_{FM}, and adds to the reachability graph. After that, a list of triggerable transitions from the initial state is built. For each triggerable transition Algorithm 2 is called. This invocation generates the successor state using the state and the transition provided as parameters.

As it is possible to arrive to a state obtained previously, this subroutine verifies this situation. When this situation takes place the algorithm only calculates the corresponding state transition and adds it to the graph structure. Instead, if the algorithm arrives into a new state it stores it in the graph as well as the state transition between the previous and this new state. Furthermore, the corresponding list of triggerable transitions from this new state is obtained. If there are no triggerable transitions for this new state the algorithm finalizes. Otherwise the Algorithm 2 is called again for each triggerable transition in the list.

Algortihm 1. getReachabilityGraph(PetriNet)

```
getReachabilityGraph(PetriNet petriNet){
  State initialState= getState(petriNet);
  addStateToGraph(newState);
  List<Transitions> triggerableTransitions=
      getTriggerableTransitions(initialState);
  forEach(T in triggerableTransitions){
    execute(initialState, T);
  }
}
```

Algortihm 2. execute(State, Transition)

```
execute(State aState, Transition aTransition){
  State newState=generateNewState(aState, aTransition);
  StateTransition aStateTransition=
          newTransition(aState, existingState);
  addTransitionToGraph (aStateTransition);
  State existingState=existsState(newState);
  if(existingState ==null){ /* newState is a new state */
    addStateToGraph(newState);
    List<Transitions> triggerableTransitions=
          getTriggerableTransitions(newState);
    forEach(T in triggerableTransitions){
      execute(newState, T);
    }
  }
  return;
}
```

6.2 Equivalent configuration final states

Since a final state represents a product configuration, two final states are equivalent configuration when the places that are representing the selection of features (places that belong to P^F) have the same number of marks. In other words, if two final states represent the selection of the same set of features they are configuring the same product. This is important for the analysis of the reachability graph in order to obtain unique configurations for the product family.

6.3 An implementation of FMTEST in Java

Taking the implementation of FM2PN as a starting point, the development of a new tool was addressed applying the knowledge gained in the previous experience. FM2PN performs a SPL analysis implementing the algorithms described in Section 6.1 and applying the PN_{FM} properties discussed in Section 4.3.
Fig. 11 shows a screenshot of the user interface of FMTEST. FMTEST provides the following functionality: search for a configuration, list all possible configurations, detection of dead nodes and calculation of metrics.

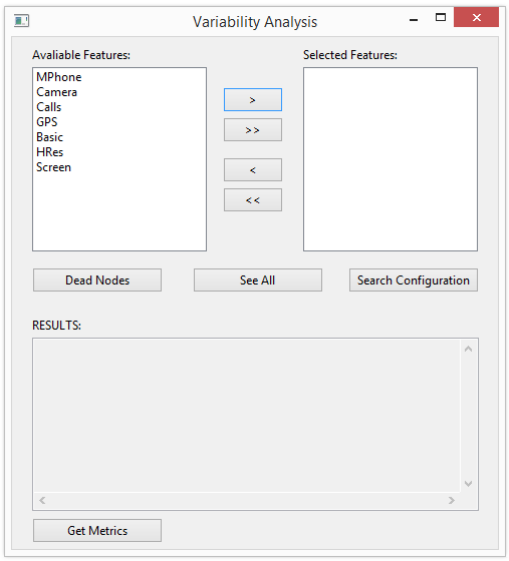

Fig. 11 FMTEST user interface.

- Search for a Configuration: Using the arrows and the two lateral list panels it is possible to select different available features of the model and search for a configuration that includes them. For instance, if feature *GPS* and feature *Camera* are selected, the configuration obtained is {*MPhone*, *Camera*, *Calls*, *GPS*, *HRes*, *Screen*} as it is shown in Fig. 12.

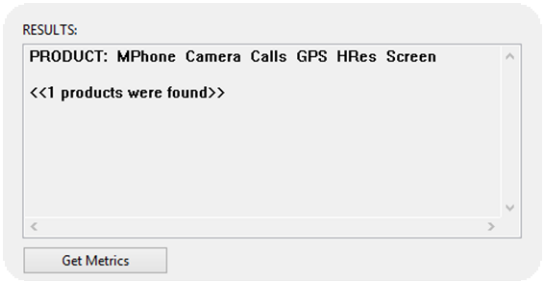

Fig. 12 “Search Configuration” result.

- List all possible configurations: FMTEST gives the possibility to list all the possible products of a SPL represented as a feature model. The “See All” button performs this action listing all the configurations found for the provided FM (Fig. 13).

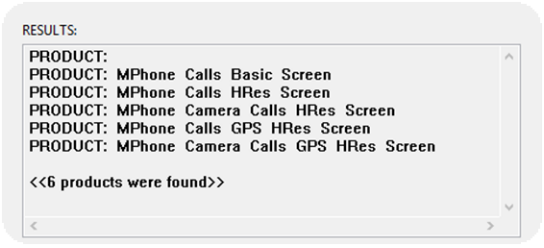

Fig. 13 “List all” result.

- Detection of dead nodes: This functionality analyse the net for detecting dead nodes. In this case no dead nodes where founded (Fig. 14).

Fig. 14 “Dead Nodes” result

- Calculation of metrics: This function allows acquiring dimension about the complexity of the model and the underlying structures created to perform the analysis. For this case, the PN created has 22 places, 11 transitions and 34 arcs. For this net, the corresponding reachability graph has 52 states and 119 transitions (Fig. 15).

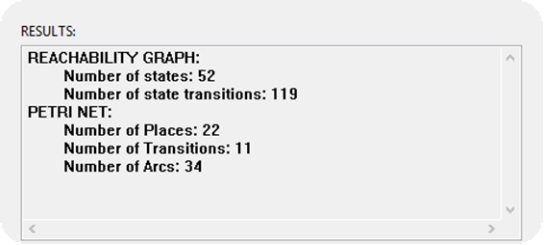

Fig. 15 “Get Metrics” result.

7 Conclusions and future work

In this work a Feature Model Petri Net, PN_{FM}, is specified by the definition of the topology of a sub-class of PN which allows us to represent a FM and develop a formal study of FMs trough the development and evolution of a SPL. The analysis is focused on the PN properties, in particular boundedness, reachability, and potentially triggerable. This approach provides a solid base to analyse satisfiability and consistency of feature models (properties described in [4] and [6]). The proposed model was implemented initially into a tool, FM2PN, which allows generating the Petri net PN_{FM} from a FM and perform the appropriate analysis using a PN analysis tool.
Subsequently, we developed a tool, FMTEST, that integrates model transformations and related analysis. Details about the PN and reachability graphs are encapsulated by FMTEST.

An important challenge is related to the size of the FMs. The ascending trend in the number of features [6] requires transformation techniques in order to reduce and simplify the PN_{FM} and the reachability graph but always keeping their properties. It’s also a pending challenge extend the study to other PNs properties within the PN_{FM} domain. Among them we can mention reversibility and synchronous distance. The first one allows us to recover the initial marking from any other marking, in other words, this will permit to rebuild a PN_{FM} from its configurations. The synchronous distance is a metric of the degree of relationship between transitions and can be used to obtain qualitative information about the dependencies in the underlying FM.

Acknowledgements. Authors thank the financial support from CONICET, Universidad Tecnológica Nacional, and Agencia Nacional de Promoción Científica y Tecnológica.

References

1. Northrop, L., Clements, P., Bachmann, F., Bergey, J., Chastek, G., Cohen, S. Donohoe, P., Jones, L., Krut, R., Little, R., McGregor, J., O'Brien, L.: A framework for product line practice, version 5.0, http://www.sei.cmu.edu/productlines/frame_report/index.html (2009)
2. van der Linden, F., Schmid, K., Rommes, E.: Software product lines in action: the best industrial practice in product line engineering. Springer Heidelberg (2007)

3. Pohl, K., Böckle, G., van der Linden, F.: Software product line engineering: foundations, principles, and techniques, Springer: Heidelberg (2005)
4. 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 (1990)
5. Schobbens, P., Trigaux, J., Heymans, P., Bontemps, Y.: Generic semantics of feature diagrams. Computer Networks 51, 456-479 (2007)
6. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review, Journal of Information Systems 35, 615-636 (2010)
7. Batory, D.: Feature models, grammars, and propositional formulas. In: Software Product Lines Conference. LNCS, vol. 3714, pp. 7–20 (2005)
8. Sun, J., Zhang, H., Li, Y. Wang, H.: Formal semantics and verification for feature modeling. In: Proceedings of the ICECSS05, pp. 303-312 (2005)
9. Benavides, D., Ruiz-Cortés, A., Trinidad, P.: Using constraint programming to reason on feature models. In: 17th International Conference on Software Engineering and Knowledge Engineering, pp. 677-682 (2005)
10. Murata, T.: Petri Nets: properties, analysis and applications. In: proceedings of the IEEE, Vol. 77:4, pp.541-580 (1989)
11. M. Mendonca, M. Branco, D. Cowan, "S.P.L.O.T. - Software Product Lines Online Tools", In 24th ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, Orlando, Florida, USA, 2009.
12. E. Berk, C. Scott Ananian, "JLex: A Lexical Analyzer Generator for Java", disponible en: http://www.cs.princeton.edu/~appel/modern/java/JLex/
13. S. Hudson, F. Flannery, C. Scott Ananian, "CUP Parser Generator for Java", disponible en: http://www.cs.princeton.edu/~appel/modern/java/CUP/
14. N. Dingle, W. Knottenbelt, T. Suto, "PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets", ACM SIGMETRICS Performance Evaluation Review, Vol. 36(4), 34-39, 2009.