Modular Modelling of Software Product Lines with Feature Nets

Radu Muschevici *José Proença* *Dave Clarke*

Report CW 609, July 2011

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Modular Modelling of Software Product Lines with Feature Nets

Radu Muschevici *José Proença* *Dave Clarke*

Report CW 609, July 2011

Department of Computer Science, K.U.Leuven

Abstract

Formal modelling and verification are critical for managing the inherent complexity of systems with a high degree of variability, such as those designed following the software product line (SPL) paradigm. SPL models tend to be large—the number of products in an SPL can be exponential in the number of features. Modelling these systems poses two main challenges. Firstly, a modular modelling formalism that scales well is required. Secondly, the ability to analyse and verify complex models efficiently is key in order to ensure that all products behave correctly. The choice of a system modelling formalism that is both expressive and well-established is therefore crucial. In this paper we show how SPLs can be modelled in an incremental, modular fashion using a formal method based on Petri nets. We continue our work on *Feature Petri Nets*, a lightweight extension to Petri nets, by presenting a framework for modularly constructing Feature Petri Nets to model SPLs.

Modular Modelling of Software Product Lines with Feature Nets[*]

Technical Report

Radu Muschevici, José Proença, and Dave Clarke

DistriNet & IBBT, Dept. Computer Science, Katholieke Universiteit Leuven, Belgium
{radu.muschevici, jose.proenca, dave.clarke}@cs.kuleuven.be

Abstract. Formal modelling and verification are critical for managing the inherent complexity of systems with a high degree of variability, such as those designed following the software product line (SPL) paradigm. SPL models tend to be large—the number of products in an SPL can be exponential in the number of features. Modelling these systems poses two main challenges. Firstly, a modular modelling formalism that scales well is required. Secondly, the ability to analyse and verify complex models efficiently is key in order to ensure that all products behave correctly. The choice of a system modelling formalism that is both expressive and well-established is therefore crucial. In this paper we show how SPLs can be modelled in an incremental, modular fashion using a formal method based on Petri nets. We continue our work on *Feature Petri Nets*, a lightweight extension to Petri nets, by presenting a framework for modularly constructing Feature Petri Nets to model SPLs.

1 Introduction

The need to tailor software applications to varying requirements, such as specific hardware, markets or customer demands, is growing. If each application variant is maintained individually, the overhead of managing all the variants quickly becomes infeasible [19]. Software Produce Line Engineering (SPLE) is seen as a solution to this problem. A Software Product Line (SPL) is a set of software products that share a number of core properties but also differ in certain, well-defined aspects. The products of an SPL are defined and implemented in terms of *features*, which are subsequently combined to obtain the final software products. The key advantage hereby over traditional approaches is that all products can be developed and maintained together. A challenge for SPLE is to ensure that all products meet their specifications without having to check each product individually, by checking the product line itself. This raises the need for novel SPL-specific formalisms to model SPLs and reason about and verify their properties.

[*] This research is partly funded by the EU project FP7-231620 HATS: Highly Adaptable and Trustworthy Software using Formal Methods (http://www.hats-project.eu), and the K.U.Leuven BOF-START project STRT1/09/031 DesignerTypeLab.

The main contribution of this paper is a modular modelling framework for specifying the behaviour of software product lines. We use *Feature Petri Nets* [18], or *feature nets* (FN) for short, as the modelling formalism. Feature nets are a Petri net extension that enables the specification of the behaviour of an entire software product line (a set of systems) in one single model. The behaviour of an FN is conditional on the features appearing in the product line. The paper makes three contributions. Firstly, it presents a variant of feature nets that improves upon their original definition [18]. Secondly, it gives a technique for constructing larger feature nets from smaller ones to model the addition of new features to an SPL. Thirdly, it provides correctness criteria for ensuring that the resulting composition preserves the behaviour of the original model(s).

Organisation: The next section motivates the need for feature nets. Section 3 formally introduces feature nets. Section 4 details an approach at modular modelling with FN, and Section 5 discusses behaviour preservation. Section 6 discusses related work. Section 7 presents our conclusions and future work.

2 Software Product Line Modelling Challenge

We illustrate the modelling challenge in software product line engineering (SPLE) using an example of a software product line of coffee vending machines. A manufacturer of coffee machines offers products to match different demands, from the basic black coffee dispenser to more sophisticated machines, such as ones that can add milk or sugar, or charge a payment for each serving. Each machine variant needs to run software adapted to the selected set of hardware features. Such a family of different software products that share functionality is typically developed using an SPLE approach, as one piece of software structured along distinct features. This has major advantages in terms of code reuse, maintenance overhead, and so forth. The challenge is ensuring that the software works appropriately in all product configurations.

Petri nets [17] are used to specify how systems behave. Fig. 1 presents an example of a Petri net for a coffee machine. This has a capacity for n coffee

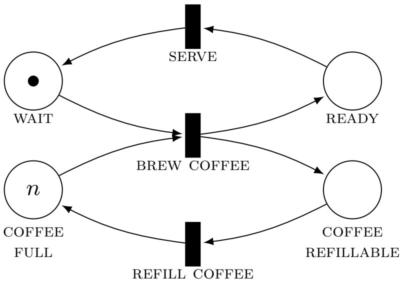


Fig. 1: Petri net model of a basic coffee machine that can only dispense coffee.

servings; it can brew and dispense coffee, and refill the machine with new coffee supplies. If we now add an optional *Milk* feature, so that the machine can also add milk to a coffee serving, we need to adapt the Petri net, not only to include the functionality of adding milk, but also to be able to control whether or not this feature is present in the resulting software product.

To address the challenge of modelling a software product line with multiple features, which may or may not be included in any given product, we introduced *Feature Petri Nets* [18]. Feature Petri Net transitions are annotated with *application conditions* [20], which are propositional formula over features that reflect when the transition is enabled. In this paper we use a variation of Feature Petri Nets in which application conditions are placed on arcs, rather than transitions, called *arc-labelled Feature Petri Nets*, though we shall just call them feature nets. One advantage of feature nets is that they enable the *superposition* of the behaviour of the various products (given by different feature selections) in the same model.

Fig. 2 exemplifies a feature net of a coffee machine with a milk reservoir. It considers a product line whose products are over the set of features $\{Coffee, Milk\}$, where *Coffee* is compulsory and *Milk* is optional. The application condition above each arc reflects that the arc is present only when the condition evaluates to true. Only then does the arc affect behaviour. If the condition is false, the arc has no effect on behaviour. Consequently, the three transitions on the left-hand side can only fire when the *Coffee* feature is present; the two transitions on the right-hand side can be taken only when the feature *Milk* is present. Observe that the restriction of this example net to the transitions that can fire for feature selection $\{Coffee\}$ is exactly the Petri net in Fig. 1, after removing unreachable places.

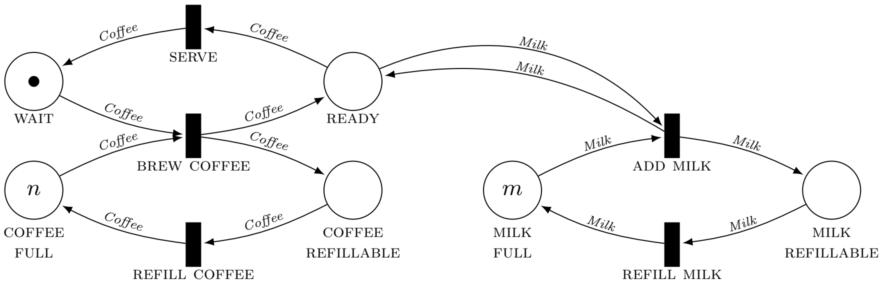


Fig. 2: Feature net of the product line $\{\{Coffee\}, \{Coffee, Milk\}\}$. Each arc has an application condition attached.

3 Feature Nets

A feature net (FN) [18] is a Petri net variant used to model the behaviour of an entire software product line. For this purpose feature nets have *application conditions* [20] attached to their arcs. An application condition is a propositional logical formula over a set of features, describing the feature combinations to which the arc applies. If the application condition is false, it is as if the arc were not present.

We define feature nets and give their semantics. We adapt the definition of feature nets described in previous work [18], where application conditions apply to transitions instead of arcs. In that paper two semantic definitions of feature nets were presented. The first semantics *directly* models the FN for a given feature selection. The second semantics, which we use and adapt here, is given by *projecting* the FN for a given feature selection onto a Petri net by removing arcs with unsatisfied application conditions. These two notions have been shown to coincide [18]. We start with some necessary preliminaries, first by defining multisets and basic operations over multisets, then by defining feature nets and their behaviour. Our terminology is standard for Petri nets [8].

Definition 1 (Multiset). *A* multiset *over a set S is a mapping $M : S \to \mathbb{N}$.*

We view a set S as a multiset in the natural way, that is, $S(x) = 1$ if $x \in S$, and $S(x) = 0$ otherwise. We also lift arithmetic operators to multisets as follows. For any function $\odot : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and multisets M_1, M_2, define $M_1 \odot M_2$ as $(M_1 \odot M_2)(x) = M_1(x) \odot M_2(x)$.

Definition 2 (Application condition [20]). *An* application condition *φ is a propositional formula over a set of features F, defined by the following grammar:*

$$\varphi \quad ::= \quad a \quad | \quad \varphi \wedge \varphi \quad | \quad \neg\varphi,$$

where $a \in F$. Write Φ_F to denote the set of all application conditions over F.

Definition 3 (Satisfaction of application conditions). *Given an application condition $\varphi \in \Phi_F$ and a set of features $FS \subseteq F$, called a* feature selection*, we say that FS satisfies φ, written as $FS \models \varphi$, defined as follows:*

$$\begin{array}{ll} FS \models a & \textit{iff } a \in FS \\ FS \models \varphi_1 \wedge \varphi_2 & \textit{iff } FS \models \varphi_1 \textit{ and } FS \models \varphi_2 \\ FS \models \neg\varphi & \textit{iff } FS \not\models \varphi. \end{array}$$

Definition 4 (Feature Net). *A feature net is a tuple $N = (S, T, R, M_0, F, f)$, where S and T are two disjoint finite sets, R is a relation on $S \cup T$ (the* flow relation*) such that $R \cap (S \times S) = R \cap (T \times T) = \emptyset$, and M_0 is a multiset over S, called the* initial marking*. The elements of S are called* places *and the elements of T are called* transitions*. Places and transitions are called* nodes*. The elements of R are called* arcs*. Finally, F is set of features and $f : R \to \Phi_F$ is a function associating each arc with an application condition from Φ_F.*

Without f and F, a feature net is just a Petri net. Sometimes we omit the initial marking M_0. The function f determines a node's pre- and post-set, defined below.

Definition 5 (Marking of a feature net). *A marking M of a feature net (S, T, R, F, f) is a multiset over S. A place $s \in S$ is* marked *iff $M(s) > 0$.*

Definition 6 (Pre-sets and post-sets). *Given a node x of a feature net and a feature selection FS, the set ${}^{(FS)}x = \{y \mid (y, x) \in R, FS \models f(y, x)\}$ is the* pre-set *of x and the set $x^{(FS)} = \{y \mid (x, y) \in R, FS \models f(x, y)\}$ is the* post-set *of x.*

Definition 7 (Enabling). *Given a feature selection FS, a marking M* enables *a transition $t \in T$ if it marks every place in ${}^{(FS)}t$, that is, if $M \geq {}^{(FS)}t$.*

We now define the behaviour of feature nets for a given feature selection.

Definition 8 (Transition occurrence). *Let $N = (S, T, R, M_0, F, f)$ be a feature net and $FS \subseteq F$ a feature selection. A transition $t \in T$* occurs*, leading from a state with marking M_i to a state with marking M_{i+1}, denoted $M_i \xrightarrow{t,FS} M_{i+1}$, iff the following two conditions are met:*

$$M_i \geq {}^{(FS)}t \qquad \text{(enabling)}$$

$$M_{i+1} = (M_i - {}^{(FS)}t) + t^{(FS)} \qquad \text{(computing)}$$

The transition rule for FN is used to define traces that describe the FN's behaviour. We now define the semantics of a feature net by projecting it onto a Petri net for a given feature selection.

Definition 9 (Projection). *Given a feature net $N = (S, T, R, M_0, F, f)$ and a feature selection $FS \subseteq F$, the* projection *of N onto FS, denoted $N \downarrow FS$, is a Petri net (S, T, R', M_0), with $R' = \{(x, y) \mid (x, y) \in R, FS \models f(x, y)\}$.*

One *projects* N onto a feature selection *FS* by evaluating all application conditions $f(x, y)$ with respect to *FS* for all arcs $(x, y) \in R$. If *FS* does not satisfy $f(x, y)$, then (x, y) is removed from the Petri net.

The behaviour of a feature net is the union of the behaviour of the Petri nets obtained by projecting all possible feature selections. The behaviour of a Petri net $N = (S, T, R, M_0)$ is given by the set of all of its traces [12], written $\mathrm{Beh}(N) = \{M_0 \xrightarrow{t_1} \cdots \xrightarrow{t_s} M_n \mid M_i \subseteq S, i \in 1..n, M_{i-1} \xrightarrow{t_i} M_i\}$, and does not include application conditions nor feature selections.

Definition 10 (FN Behaviour). *Given an FN $N = (S, T, R, M_0, F, f)$, we define* $\mathrm{Beh}(N)$ *as follows:*

$$\mathrm{Beh}(N) = \bigcup_{FS \subseteq F} \mathrm{Beh}(N \downarrow FS).$$

A feature net combines the behaviour of a set of Petri nets in a single model. Feature nets do not exceed the expressive power of Petri nets. This is indicated by the fact that a feature net can be encoded as a set of Petri nets. Such an encoding involves two steps: first encoding a FN as a transition-labelled Feature Petri Net [18], and secondly describing the behaviour of the Feature Petri Net using a set of regular Petri nets. The first encoding replaces each transition attached to n arcs in R by 2^n transitions, one for each possible combination of the possible arcs. The second encoding step into Petri nets can be achieved by encoding the satisfaction condition of FN transition occurrences by considering for each feature F two places, F ON and F OFF, marked in mutual exclusion depending on whether the feature is selected or not. The details of this encoding are in a previous paper [18].

Given that feature nets are as expressive as Petri nets, analysis techniques for Petri nets still apply to feature nets. At the same time, feature nets offer a concise way to describe the systems in an SPL.

4 Modular modelling

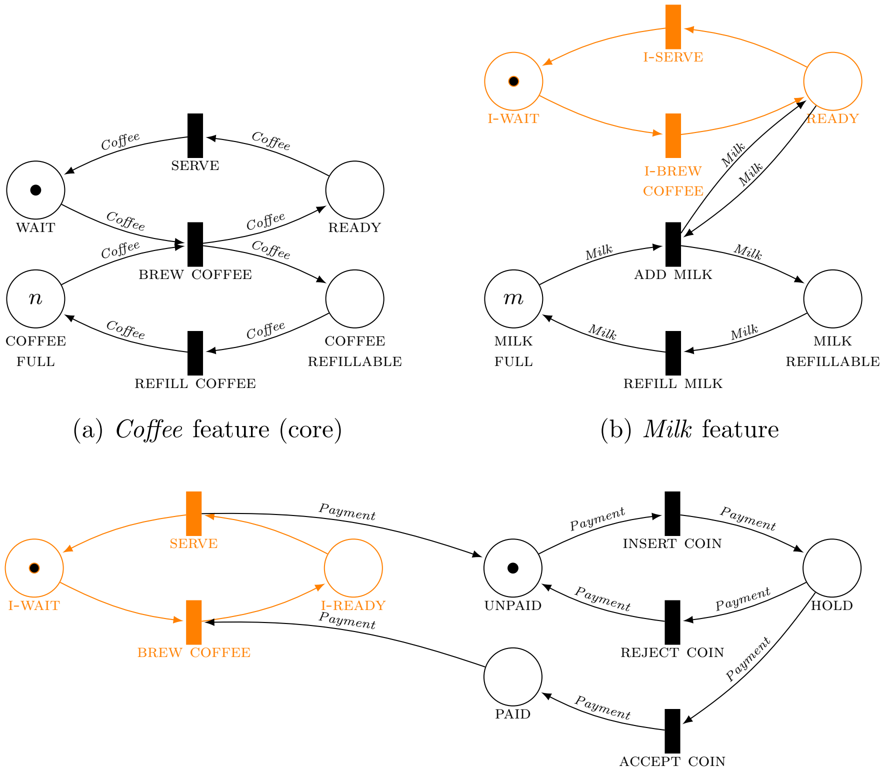
For a modelling formalism to be useful in practice, it needs to facilitate modular development techniques. This is especially important for modelling software product lines: a single SPL model combines the behaviour of a set of different systems, which are often too complex to develop simultaneously.

Modular approaches include top-down techniques, where initially an abstract model is sketched and more details are added incrementally, and bottom-up approaches, where subsystems are modelled separately and later plugged together to a global model. Petri nets support both approaches [12]. In the following we propose a bottom-up composition technique for feature nets. It is based on the idea of modelling features of the system individually and then combining them to obtain a model of the entire SPL. Our approach starts by building a model of the *core* system that is the behaviour which is common to all products of the SPL. Optional features are modelled as separate nets, which also specify how they interact with the core through an *interface*. Core and additional features are then composed stepwise, by incrementally applying each feature to the core. We show how this technique can be applied to modularly specify a coffee machine product line from the three features *Coffee*, *Payment* and *Milk*.

4.1 Feature net Composition

We devise a modular modelling approach in which features are first expressed as separate FNs. A feature's interaction with the rest of the system (the *core*) is modelled using an *interface*. Features are modelled separately in such a way that they can be attached to the core, in order to incrementally build a larger model. The interface simulates the behaviour of the core that the features are designed to be plugged into. A feature modelled using this technique can be seen as a partially specified model of the entire SPL, where the feature's behaviour is fully specified, whereas everything else is underspecified. Composition then entails connecting the interface to the core to obtain a specification of the combined system.

The three features of our example coffee machine are modelled as separate FNs (Fig. 3). Apart from when a feature's behaviour is self-contained (such as the *Coffee* net in Fig. 3a) it will typically interact with other features that are part of the larger system. To faithfully model such interactions we include an interface. The aim of the interface is to abstract part of the larger system's behaviour. The interface will also be used to show that the net exposes the same behaviour as it does when it is part of the combined system. For example, the model of *Milk* in Fig. 3b reflects the fact that adding milk depends on a state



(a) *Coffee* feature (core)

(b) *Milk* feature

(c) *Payment* feature

Fig. 3: Individual feature nets modelling the features *Coffee*, *Milk* and *Payment* of a product line of coffee machines. Interfaces are highlighted in orange.

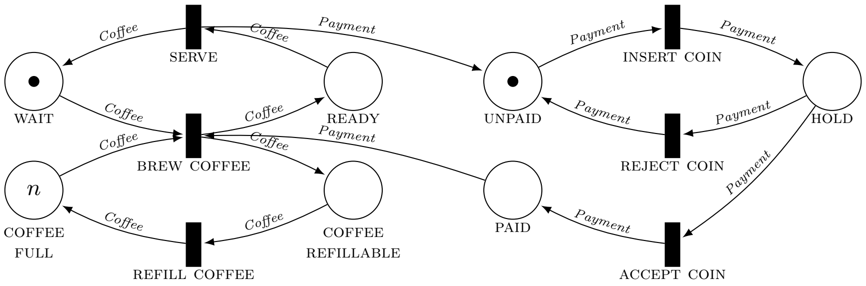


Fig. 4: A software product line over feature set $\{\mathit{Coffee}, \mathit{Payment}\}$ obtained by applying the delta net *Payment* (Fig. 3c) to the core net modelling *Coffee* (Fig. 3a).

of the system in which a cup of fresh coffee is available. The larger system is represented abstractly by the highlighted interface, which models the availability of coffee in the place READY; a token in this place would denote a state in which a freshly brewed cup of coffee is available. Similarly, Fig. 3c models the fact that after a payment has been accepted, the overall system is able to BREW COFFEE, and after serving the coffee, the system goes back to an UNPAID state.

Constructing a model of the whole SPL is done by stepwise applying the delta nets of each feature to a core model. The intuition behind delta net application is that each interface is replaced with a more complex feature net. In our example, the first step could be to refine *Payment*'s interface by replacing it with the feature net for *Coffee*. In a second step, the feature *Milk* is refined by replacing its interface with the net obtained in the previous step.

We now formally introduce the application of a delta net to a core net.

Definition 11 (Delta Feature Net). *A delta feature net N is a FN with a designated interface, denoted $N = (S, T, R, F, f, S_I, T_I)$, where $S_I \subseteq S$, $T_I \subseteq T$.*

Delta feature nets specify the behaviour of features designed to be added to a larger system. A set of delta FN is combined with a stand-alone FN, the *core*, by sequentially *applying* each delta net to the core. Delta nets include an interface, which models interactions with the core. Such interactions are modelled by transitions or places common to both core and delta net.

Definition 12 (Delta Net Application). *Let $N = (S, T, R, F, f)$ be a feature net and $D = (S_d, T_d, R_d, F_d, f_d, S_I, T_I)$ a delta feature net with $S \cap S_d \neq \emptyset$. The* application *of D to N results in a net $N' = (S', T', R', F', f')$, written as $N \oplus D$, where*

$$\begin{aligned} S' &= (S_d \setminus S_I) \cup S & F' &= F \cup F_d \\ T' &= (T_d \setminus T_I) \cup T & f' &= (f \cup f_d) \restriction R' \\ R' &= \{(s,t) \in (R \cup R_d) \mid s \in S', t \in T'\} \\ &\quad \cup \{(t,s) \in (R \cup R_d) \mid t \in T', s \in S'\}. \end{aligned}$$

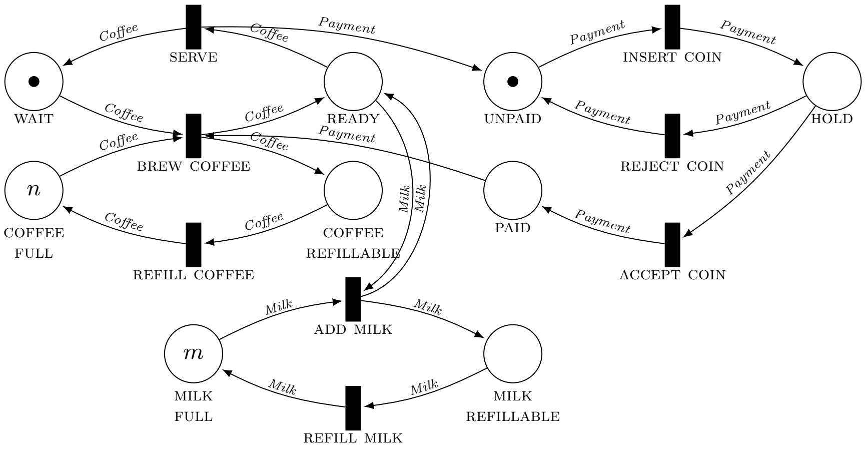


Fig. 5: FN model of an SPL over the feature set $\{\mathit{Coffee}, \mathit{Payment}, \mathit{Milk}\}$ obtained by sequential application of the delta nets for the features *Payment* (Fig. 3c) and *Milk* (Fig. 3b).

When applying a delta net to a core, the interface is dropped and the two nets are "fused" along their common nodes. The arcs that previously connected the delta net interface now connect the core. The applicability of a delta net is limited to certain cores. Let S_B and T_B represent the border of the interface, that is, $S_B = \{s \in S_I \mid \exists t \in T_d \setminus T_I : (s,t) \in R'\}$ and $T_B = \{t \in T_I \mid \exists s \in S_d \setminus S_I : (t,s) \in R'\}$. A delta net is applicable to a core net if the border of the interface is preserved, that is, if $S \cap S_d = S_B$ and $T \cap T_d = T_B$.

We show how delta net application is used to build a model of the example coffee machines SPL. Starting with the separate sub-models in Fig. 3, delta nets are applied stepwise to a growing core. First, a model with the two features *Coffee* and *Payment* is composed by applying the delta net from Fig. 3c to the core shown in Fig. 3a. These nets have the two transitions SERVE and BREW COFFEE in common. The result after applying the delta feature net is the new core feature net shown in Fig. 4. In a second step, we add the *Milk* behaviour by applying the feature net in Fig. 3b to the core obtained in the previous step. These two nets have the place READY in common. The result after delta net application is the model shown in Fig. 5. Note that the order in which we apply the two delta nets does not matter in this case, because neither feature (*Milk* or *Payment*) depends on the other. In general, features can depend on other features. This would be reflected by the design of their interfaces, effectively restricting the applicability and ensuring that the delta nets can only be applied in a valid order. As a consequence, delta net application is not commutative.

5 Correctness

When is the application of a delta net D to a core net N *correct*? We consider this application correct if the traces of N and D are in some way the same as the traces of $N \oplus D$, introduced in Definition 12, after projecting onto the transitions of N and D. However, there are various ways to compare these traces. We can focus only on the features used by the original nets or on the features used by the combined net. Also checking correctness of the core net can be different from checking correctness of the delta net. Finally, it might be enough to consider only trace inclusion between the original nets and the combined net. The three dimensions are summarised as:

- **Original** vs. **combined** features. When comparing the behaviour of one of the original nets with the combined net, we can either consider the combined features in the final net or just the features in one of the original nets.
- **Core** vs. **delta**. We can evaluate the correctness of the core or delta net behaviour, always in comparison to the combined net's behaviour.
- **Liveness**, **safety**, or **both**. Liveness states that a net cannot inhibit behaviour in the other net, while safety states that a net cannot introduce new behaviour to the other net. For example, we say a delta application is safe with respect to the core net N if the traces of the combined net are included in the traces of N, when considering the common transitions.

By choosing different parameters along these dimensions we obtain different notions of correctness. We formulate a parametrised notion of correctness for the application of delta net D to a core net N as follows:

$$\forall FS \subseteq \Theta_1 : \mathrm{Beh}(\Theta_2 {\downarrow} FS) \;\; \Theta_3 \;\; \mathrm{Beh}((N \oplus D) {\downarrow} FS) \quad \text{(parametrised correctness)}$$

where Θ_1 can be either the full set of features or the features of the net Θ_2, Θ_2 can be either the core or the delta net, and Θ_3 is an inclusion or equivalence relation between the two sets of traces, with respect to a set of relevant transitions. When Θ_3 is a superset relation, it represents *safety*, since no new traces can be introduced by combining the two nets. On the other hand, a subset relation represents *liveness*, since all traces in the original net are still valid traces in the combined net. When we have both safety and liveness assurances, we say that the behaviour is *preserved*, and instantiate Θ_3 to be the equality of the traces with respect to the common transitions.

Not all combinations of these dimensions are desirable in all cases. For example, sometimes we might want to inhibit or extend the behaviour of a core net with respect to the combined set of features, breaking the liveness or safety criteria. However, it seems desirable to preserve this behaviour with respect to the features of the core net. In fact, it is open to debate which combination of these dimensions are ideal. In this paper, we provide sufficient conditions to guarantee:

1. *Preservation* of the behaviour of N with respect to the *original* features.
2. *Preservation* of the behaviour of D with respect to the *combined* features.
3. *Safety* of the behaviour of N with respect to the *combined* features.

5.1 Mathematical preliminaries

We defined liveness and safety as inclusion of traces with respect to a relevant set of traces. We formalise this concept below.

Definition 13 (Behaviour inclusion $\subseteq_{Ts}$). *Let $N_i = (S_i, T_i, R_i)$ be a pair of Petri nets, for $i \in 1..2$, and Ts be a set of transitions. We say that the behaviour of N_1 is included by the behaviour of N_2 with respect to Ts, written* $\mathrm{Beh}(N_1) \subseteq_{Ts} \mathrm{Beh}(N_2)$, *if* $\mathrm{Beh}(N_1) \upharpoonright Ts \subseteq \mathrm{Beh}(N_2) \upharpoonright Ts$, *where* $\mathrm{Beh}(N) \upharpoonright Ts = \{tr \upharpoonright Ts \mid tr \in \mathrm{Beh}(N)\}$ *and:*

$$M \upharpoonright Ts = \varepsilon \qquad (M \xrightarrow{t} tr) \upharpoonright Ts = \begin{cases} t \cdot (tr \upharpoonright Ts) & \textit{if } t \in Ts \\ tr \upharpoonright Ts & \textit{otherwise.} \end{cases}$$

Similarly, we write $\supseteq_{Ts}$ and $=_{Ts}$ to represent superset inclusion and equality for the transitions in Ts.

We now define *weak bisimulation* between two feature nets, which we will use to relate the interface of a delta net with the net to which the delta is applied to, based on the notion of bisimulation described by Schnoebelen and Sidorova [21].

Definition 14 (Weak bisimulation). *Let $N_i = (S_i, T_i, R_i, M_{0i}, F_i, f_i)$ be two feature nets, for $i \in 1..2$, $\mathcal{M}_i$ the set of markings of N_i, and $\mathcal{B} \subseteq (\mathcal{M}_1 \times \mathcal{M}_2) \cup (T_1 \times T_2)$ a relation over markings and transitions. Recall also the notion of occurrence of transitions introduced in Definition 8. In the following we write $t \in \mathcal{B}$ to denote that t is in the domain or codomain of $\mathcal{B}$. $\mathcal{B}$ is a weak bisimulation if, for any feature selection FS:*

1. $M_{01}\ \mathcal{B}\ M_{02}$
2. *$\forall (M_1, M_2) \in \mathcal{B}$, if $M_1 \xrightarrow{t_1, FS} M'_1$ and $t_1 \notin \mathcal{B}$, then $M'_1\ \mathcal{B}\ M_2$;*
3. *$\forall (M_1, M_2) \in \mathcal{B}$, if $M_1 \xrightarrow{t_1, FS} M'_1$ and $t_1 \in \mathcal{B}$, then there exists $t_2 \in T_2$ and M'_2 such that $M_2 \xrightarrow{t_2, FS}_{\mathcal{B}} M'_2$, $M'_1\ \mathcal{B}\ M'_2$, and $t_1\ \mathcal{B}\ t_2$;*
4. *conditions (2) and (3) also hold for $\mathcal{B}^{-1}$;*

where $M \xrightarrow{t, FS}_{\mathcal{B}} M'$ denotes that there are n transitions $t_1 \ldots t_n$ such that

$$M \xrightarrow{t_1, FS} \cdots \xrightarrow{t_n, FS} M_n \xrightarrow{t, FS} M' \quad \textit{and} \quad \forall j \in 1..n : t_j \notin \mathcal{B}.$$

If a weak bisimulation exists between N_1 and N_2 we say that they are weakly bisimilar, written $N_1 \approx N_2$.

Let C be the feature net for the *Coffee* feature (Fig. 3a), and P the delta net dealing with *Payment* (Fig. 3c). The interface of P can be seen as a feature net P_I. It holds that $C \approx P_I$. Furthermore, there exists a bisimulation $\mathcal{B}$ that relates the transitions with the same name of the two nets, namely SERVE and BREW COFFEE. More specifically, the relation $\mathcal{B}$ below is a bisimulation, where we write $\mathcal{M}_C$ and $\mathcal{M}_{P_I}$ to denote all the markings of C and P_I, respectively.

$$\begin{aligned} &\{(M, M') \mid M \in \mathcal{M}_C, M' \in \mathcal{M}_{P_i}, M(\textsc{wait}) = 1, M'(\textsc{wait}) = 1\} \cup \\ &\{(M, M') \mid M \in \mathcal{M}_C, M' \in \mathcal{M}_{P_i}, M(\textsc{wait}) = 0, M'(\textsc{wait}) = 0\} \cup \\ &\{(\textsc{serve}, \textsc{serve}), (\textsc{brew coffee}, \textsc{brew coffee})\} \end{aligned}$$

5.2 Preservation of the core behaviour for the original features

Our first criterion compares the core net with the combined net, considering only the features originally present in the core net. We require the behaviour of the core net to be *preserved* in the combined net, that is, their traces must coincide with respect to the transitions in the core net. We formalise this criterion as follows.

Criterion 1 (preservation/core/original) *Let* $N = (S, T, R, F, M_0, f)$ *be a core net and* D *a delta net. We say that* $N \oplus D$ *preserves the behaviour of* N *for the features in* F *iff*

$$\forall FS \subseteq F \ : \ \mathrm{Beh}(N {\downarrow} FS) =_T \mathrm{Beh}(N \oplus D {\downarrow} FS).$$

To verify that a delta net application obeys the above correctness criteria, it is sufficient (although not necessary) to verify the following condition. Check that the arcs between the interface and the non-interface nodes of D require at least one 'new' feature to be present. By new feature we mean a feature that is not in F. This syntactic check ensures that, when considering only the features from the core net, the arcs connecting it to the delta net will never be active.

Theorem 1. *Let* $D = (S_d, T_d, R_d, M_{0d}, F_d, f_d, S_I, T_I)$ *be a delta net,* $N = (S, T, R, M_0, F, f)$ *a feature net, and* $R_I \subseteq R_d$ *be the set of arcs connecting interface nodes* $(S_I \cup T_I)$ *to non-interface nodes. The behaviour of* N *is preserved by* $N \oplus D$ *for the features in* F *(Criterion 1) if:*

$$\forall (x, y) \in R_I \ : \ \forall FS \in F_d \cup F \ : \ FS \models f(x, y) \quad \rightarrow \quad FS \cap (F_d \backslash F) \neq \emptyset. \qquad (1)$$

Proof. Assume that Equation (1) holds for $N \oplus D = N'$. We show that the core behaviour is preserved, i.e., that $\forall FS \subseteq F \cdot \mathrm{Beh}(N \downarrow FS) =_T \mathrm{Beh}(N' \downarrow FS)$. Observe that, for every $FS \subseteq F$, $FS \cap (F_d \backslash F) = \emptyset$. Hence, by assuming Equation (1) we conclude that for every arc $(x, y) \in R_I$, $FS \not\models f(x, y)$. Therefore the traces $t \in \mathrm{Beh}(N {\downarrow} FS)$ coincide with the traces of $\mathrm{Beh}(N' {\downarrow} FS)$ with respect to the transitions of N.

In both our examples of delta applications, that is, adding payment to a coffee machine and adding milk to the resulting net, the condition in Equation (1) holds. The intuition is that, for example, when the *Payment* feature is not available, the *Coffee* feature net is detached from the *Payment* feature net in the combined net. Hence its behaviour is not affected by the *Payment* net and is preserved.

5.3 Preservation of the delta behaviour for the combined features

We now define the second correctness criterion.

Criterion 2 (preservation/delta/combined) *Let* $N = (S, T, R, M_0, F, f)$ *be a core net and* $D = (S_d, T_d, R_d, M_{0d}, F_d, f_d, S_I, T_I)$ *a delta net. We say that* $N \oplus D$ *preserves the behaviour of* D *with respect to features from the combined net iff*

$$\forall FS \subseteq F \cup F_d \ : \ \mathrm{Beh}(D {\downarrow} FS) =_{T_d \backslash T_I} \mathrm{Beh}(N \oplus D {\downarrow} FS).$$

As with the correctness Criterion 1, we present a sufficient condition that guarantees the preservation of the Criterion 2. However, as opposed to the previous case, this condition is based on a *semantic property* of the the interface and the core net.

Theorem 2. *Let $D = (S_d, T_d, R_d, M_{0d}, F_d, f_d, S_I, T_I)$ be a delta net, $N_I = (S_I, T_I, R_I, M_{0D}, F_d, f_d)$ be the interface of D, $N = (S, T, R, F, f)$ a (core) feature net, and $R_B \subseteq R_d$ denote the arcs connecting interface to non-interface nodes. The behaviour of D is preserved by $N \oplus D$ (Criterion 2) if $N \approx N_I$ and there is a weak bisimulation $\mathcal{B} \subseteq (\mathcal{M}_1 \times \mathcal{M}_2) \cup (T_1 \times T_2)$ such that:*

$$\{(t, t) \mid t \in T \cap T_I\} \subseteq \mathcal{B}, \tag{2}$$

$$\forall s \in S \cap S_I, (M, M') \in \mathcal{B} : M(s) = M'(s), \tag{3}$$

$$\forall (s, t) \in R_B, s \in S_I, (M, M') \in \mathcal{B} \;:\; (M - \{s \mapsto 1\})\ \mathcal{B}\ (M' - \{s \mapsto 1\}) \tag{4}$$

$$\forall (t, s) \in R_B, s \in S_I, (M, M') \in \mathcal{B} \;:\; (M + \{s \mapsto 1\})\ \mathcal{B}\ (M' + \{s \mapsto 1\}) \tag{5}$$

For Equation (4) we assume that, if $M(s) = M'(s) = 0$, then subtracting $\{s \mapsto 1\}$ does not change the markings.

Proof. Let $FS \subseteq F \cup F_d$, and $\mathcal{B}$ the bisimulation described by Theorem 2. We write M^N, M^D, and M^I to denote the markings M restricted to the places of N, D, and the interface of D, respectively.

We prove safety, while liveness can be shown in an analogous way, because $\mathcal{B}$ is symmetric. We show by induction the following property. Assume that $tr = t_1 \cdots t_n$ is a trace both in $\mathrm{Beh}(N \oplus D{\downarrow}FS) \restriction (T_d \backslash T_I)$ and in $\mathrm{Beh}(D{\downarrow}FS) \restriction (T_d \backslash T_I)$, ending in marking M and L respectively, and $M^N\ \mathcal{B}\ L^I$. Then for every transition t such that $(tr \cdot t) \in \mathrm{Beh}(N \oplus D{\downarrow}FS) \restriction (T_d \backslash T_I)$ ending in marking M', it also holds that $(tr \cdot t) \in \mathrm{Beh}(D{\downarrow}FS) \restriction (T_d \backslash T_I)$ ending in marking L' and $M'^N\ \mathcal{B}\ L'^I$. We now distinguish three scenarios for t.

1. $t \in T_d \backslash T_I$. t can also be performed by D. The possible problem is when places in ${}^{(FS)}t$ or $t^{(FS)}$ are in the interface. However, the property of $\mathcal{B}$ described by Theorem 2 guarantees that the shared markings between N and the interface of D have the same tokens for the shared places. The final marking in this case will still be in the bisimulation, i.e., the final marking L' from D will preserve the number of tokens in the M'^N and L'^I. Hence, by Equation (4) we conclude $M'^N\ \mathcal{B}\ L'^I$, and using induction hypothesis we conclude also that $(tr \cdot t) \in \mathrm{Beh}(D{\downarrow}FS) \restriction (T_D \backslash T_I)$.
2. $t \in T \cap T_I$. t is a transition from N and from D. Then the bisimulation gives that $t\ \mathcal{B}\ t$ and $L^I \xrightarrow{t,FS} L'^I$ is a firing from D, where $M^N\ \mathcal{B}\ L'^I$. Using the induction hypothesis we conclude that also $(tr \cdot t) \in \mathrm{Beh}(D{\downarrow}FS) \restriction (T_D \backslash T_I)$.
3. $t \in T \backslash T_I - t$ is a transition from N but not from D. Since $\mathcal{B}$ is a weak bisimulation and $M^N\ \mathcal{B}\ L^I$, then the interface of D can perform zero or more transitions in T_I (hence not visible when restricting to $T_D \backslash T_I$) until a transition L'^I such that $M'^N\ \mathcal{B}\ L'^I$. Again, the induction hypothesis allows us to conclude that $(tr \cdot t) \in \mathrm{Beh}(D{\downarrow}FS) \restriction (T_D \backslash T_I)$.

By (1), (2), and (3), and because the empty trace is always globally safe, we conclude by induction that any trace in $\mathrm{Beh}(N \oplus D{\downarrow}FS)$ is also in $\mathrm{Beh}(D{\downarrow}FS)$, when restricted to $T_d \backslash T_I$.

Intuitively, Theorem 2 is justified by the fact that, if the interface captures the expected behaviour of the core net, then the behaviour of the delta net is preserved with respect to the combined net. Equations (2) and (3) ensure that the shared transitions and places between the core net and the interface are related by a weak bisimulation. Equations (4) and (5) ensure that, even when the interface net and the core net exchange tokens with the delta net, they are still weakly bisimilar. Weak bisimulation (Definition 14) relates the interface of the delta net and the core net.

Recall our running examples. As explained in the end of Section 5.1, there is a weak bisimulation between the interface of the delta net for payment P and the core net for coffee C. This simulation obeys Equation (2) because the shared transitions are related by $\mathcal{B}$, Equation (3) because there places of C and P are disjoint, and Equation (4) because, in this case, $\mathrm{dom}(R) \cap S_I = \emptyset$. Hence the composition $CP = C \oplus P$ is correct with respect to Criterion 2. Consider now the application of the delta net for milk M to the previously obtained core CP. A possible weak bisimulation between CP and the interface of M relates equal markings of the places READY in CP and READY in M, as well as of the places WAIT and I-WAIT. Note that, in order to use Theorem 2, we need to include markings for any number of tokens in READY, because of Equations (4) and (5). Hence, Equation (2) trivially holds, and our specific bisimulation relation also captures Equation (3). We conclude that the composition $CP \oplus M$ is also correct with respect to Criterion 2.

5.4 Safety of the core behaviour for the combined features

Our last correctness criterion compares the core net with the combined net with respect to all features, as opposed to the first criterion that only considered the features of the core net. When including the features in the delta net, we consider it *safe* to inhibit traces that were initially possible, provided that no new traces are introduced. We formalise safety using trace inclusion.

Criterion 3 (safety/core/combined) *Let $N = (S, T, R, M_0,\ F, f)$ be a core net and $D = (S_d, T_d, R_d, M_{0d}, F_d, f_d, S_I, T_I)$ a delta net. We say that $N \oplus D$ is safe with respect to D and to the combined features iff*

$$\forall FS \subseteq F \cup F_d \ : \ \mathrm{Beh}(N{\downarrow}FS) \supseteq_T \mathrm{Beh}(N \oplus D{\downarrow}FS).$$

We claim that, when applying a delta net connecting only places from the interface to the rest of the delta, the delta net application is safe with respect to D and the combined features.

Theorem 3. *When applying a delta net $D = (S_d, T_d, R_d, M_{0D}, F_d, f_d, S_I, T_I)$ to a core net N, we guarantee that $N \oplus D$ is safe with respect to D and the combined features if:*

$$\forall s \in S_I, t \in T_d \backslash T_I \ : \ (s, t) \notin R_D. \tag{6}$$

The theorem is easily justified by the fact that the core net will only be connected to the rest of the delta net through transitions. When the application of a delta net respects Equation (6), we are increasing the pre- and post-sets of these transitions, thereby further restricting when they can be fired.

We exemplify the application of two delta nets in this paper: the *Payment* and the *Milk* nets (Fig. 3c and 3b). The first net obeys the condition in Theorem 3, hence the correctness Criterion 3 holds. The second delta net has arcs connecting places from the interface to a non-interface transition, invalidating Equation (6). However, in this case the safety criterion is still preserved, because a token that exits the core when firing ADD MILK is transported back to its origin in the same step.

6 Related Work

Our research relates to Petri net based formalisms, and to the behavioural specification of software product lines. We highlight the most relevant works in these areas. Petri net composition and decomposition strategies that preserve some properties of the initial net(s) have been studied thoroughly [4,22,21,12]. In *Open Petri Nets* [2], places designated as open represent an interface towards the environment. Open nets are composed by fusing common open places, and the composition operation is shown to preserve behaviour with respect to an inverse decomposition operation. Our Petri net model uses a similar notion of interface, which includes an abstraction of the net that will be matched during application. We use an incremental approach using application of deltas instead of a symmetric composition operation, guided by the intuition that larger systems are build by extending more fundamental systems. The main focus of open Petri nets is the study of properties in a category of nets, while we have a more practical focus on the incremental development of nets. Various formalisms have been adopted for specifying the behaviour of software product lines, with the aim of providing a basis for analysis and verification of such models. A survey of formal methods for software product lines has recently been published [5]. *UML activity diagrams* have been used to model the behaviour of SPL by superimposing several such diagrams in a single model [7]. Attached to the activity diagram's elements are "presence expressions," which are similar to application conditions. Compared to activity diagrams, Petri nets have a stronger formal foundation, with a larger spectrum of analysis and verification techniques, although, several studies have expressed the semantics of UML diagram using Petri nets (e.g. [10]). Gruler et al. extended Milner's CCS with a product line variant operator that allows an alternative choice between two processes [14]. The *PL-CCS* calculus includes information about variability: by defining dependencies between features, one can control the set of valid configurations [13]. Variability is often modelled using transition systems enhanced with product-related information. *Modal transition systems* (MTS) [15] allow optional transitions, lending themselves as a tool for modelling a set of behaviours at once [11]. Generalised extended MTS [9] introduce cardinality-based variability operators and propose to use temporal logic formulas to associate related variation points. Asirelli et al. encode MTS using propositional deontic logic formulas and provide a framework for reasoning about behavioural aspects [1]. *Modal I/O automata* [16] are a behavioural formalism for describing the variability of components based on MTS and I/O automata. Mechanisms for component composition are provided to support a product line theory. These approaches do not relate behaviour to elements of a structural variability model. *Featured transition systems* (FTS) [6] are an extension of labeled transition systems. Similar to feature nets, transitions are explicitly labeled with respect to a feature model, and a feature selection determines the subset of active transitions. In FTS, transitions are mapped to single features. Transition priorities are used to deal with undesired non-determinism when selecting from transitions associated to different features. With application conditions, priorities are no longer required because we can negate the features in other transitions to obtain the same effect.

7 Conclusion and Future Work

This paper introduces a modular framework for modelling systems with a high degree of variability, addressing an important challenge in software product line engineering. The modelling formalism used is feature nets, a lightweight Petri net extension in which the presence of arcs is conditional on the presence of certain features through *application conditions*. We present an approach to composing behavioural models from separately engineered models of individual features. Three correctness criteria for such compositions are also presented.

Feature nets capture the behaviour of entire product lines in a single, concise model, opening the way for efficient analysis and verification. We will follow this direction in future work, applying model checking techniques to our models and studying the question of verification. The practical applicability of our proposed approach will be examined in a future study, which will also determine how well the approach scales, considering that features are not always independent.

References

1. Asirelli, P., ter Beek, M.H., Gnesi, S., Fantechi, A.: A deontic logical framework for modelling product families. In: Benavides et al. [3], pp. 37–44
2. Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional semantics for open Petri nets based on deterministic processes. Mathematical Structures in Computer Science 15(01), 1–35 (2005)
3. Benavides, D., Batory, D.S., Grünbacher, P. (eds.): International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS), vol. 37. Universität Duisburg-Essen (2010)

4. Berthelot, G.: Transformations and decompositions of nets. Petri Nets: Central Models and Their Properties pp. 359–376 (1987)
5. Clarke, D.: Quality Assurance for Diverse Systems, chap. 5, pp. 27–37. Deliverable 1.2 of the EternalS Coordination Action (FP7-247758), supported by the 7th Framework Programme of the EC within the FET scheme (2011), https://www.eternals.eu/sites/default/file/D1_2_TF1_stateOfTheArt.pdf
6. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A., Raskin, J.F.: Model checking lots of systems: Efficient verification of temporal properties in software product lines. In: International Conference on Software Engineering. pp. 335–344. IEEE Press (2010)
7. Czarnecki, K., Antkiewicz, M.: Mapping features to models: A template approach based on superimposed variants. In: Generative Programming and Component Engineering, LNCS, vol. 3676, pp. 422–437. Springer (2005)
8. Desel, J., Esparza, J.: Free choice Petri nets. Cambridge University Press, New York, NY, USA (1995)
9. Fantechi, A., Gnesi, S.: Formal modeling for product families engineering. In: International Software Product Line Conference. pp. 193–202. IEEE Press (2008)
10. Farooq, U., Lam, C.P., Li, H.: Transformation methodology for UML 2.0 activity diagram into colored Petri nets. In: Advances in Computer Science and Technology. pp. 128–133. ACTA Press (2007)
11. Fischbein, D., Uchitel, S., Braberman, V.: A foundation for behavioural conformance in software product line architectures. In: International Workshop on the Role of Software Architecture in Analysis and Testing. pp. 39–48. ACM Press (2006)
12. Girault, C., Valk, R.: Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications. Springer, Secaucus, NJ, USA (2001)
13. Gruler, A., Leucker, M., Scheidemann, K.: Calculating and modeling common parts of software product lines. In: International Software Product Line Conference. pp. 203–212. IEEE Press (2008)
14. Gruler, A., Leucker, M., Scheidemann, K.: Modeling and model checking software product lines. In: International Conference on Formal Methods for Open Object-based Distributed Systems. LNCS, vol. 5051, pp. 113–131. Springer (2008)
15. Larsen, K., Thomsen, B.: A modal process logic. In: Third Annual Symposium on Logic in Computer Science. pp. 203–210. IEEE Press (1988)
16. Larsen, K., Nyman, U., Wąsowski, A.: Modal I/O automata for interface and product line theories. In: Programming Languages and Systems. LNCS, vol. 4421, pp. 64–79. Springer (2007)
17. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (Apr 1989)
18. Muschevici, R., Clarke, D., Proença, J.: Feature Petri Nets. In: International Software Product Line Conference. vol. 2, pp. 99–106. Lancaster University (2010)
19. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering. Springer (2005)
20. Schaefer, I.: Variability modelling for model-driven development of software product lines. In: Benavides et al. [3], pp. 85–92
21. Schnoebelen, P., Sidorova, N.: Bisimulation and the reduction of Petri nets. In: Application and Theory of Petri Nets, LNCS, vol. 1825, pp. 409–423. Springer (2000)
22. Souissi, Y., Memmi, G.: Composition of nets via a communication medium. In: Advances in Petri Nets, LNCS, vol. 483, pp. 457–470. Springer (1991)