Towards Extensible Structural Analysis of Petri Net Product Lines

Elena Gómez-Martínez, Juan de Lara, and Esther Guerra

Universidad Autónoma de Madrid,
Escuela Politécnica Superior, Departamento de Ingeniería Informática, Spain
{mariaelena.gomez,Juan.deLara,esther.guerra}@uam.es

Abstract. In order to represent the behaviour of a (potentially large) set of concurrent systems, we propose a notion of *product line* of Petri nets, where presence conditions can be flexibly attached to places, transitions and arcs. To enable an efficient analysis of the whole set of nets, we have lifted several structural analysis methods for Petri nets, to the product line level. This avoids analysing each particular net in isolation. Finally, we propose an extensible tool infrastructure, based on Eclipse and on top of FeatureIDE, which supports the approach and permits adding new analysis methods in a non-intrusive way.

Keywords: Petri nets, Product lines, Model-driven engineering

CEUR-WS.org/Vol-2424/paper3.pdf

1 Introduction

Petri nets is a popular formalism to model concurrent systems [13]. It is widely used due to its rich body of theoretical results enabling analysis, and the plethora of existing supporting tools. However, in scenarios that require modelling families of similar systems (e.g., variants of machine controllers with different characteristics, or design variants of flexible assembly lines), one needs to build many variations of a base Petri net. If the set of nets is large, then it becomes challenging to build, maintain and analyse.

To facilitate this task, we combine Petri nets with software product lines (SPLs) [17] to define a notion of Petri net product line (PNPL). This allows modelling the variability space using a feature model, and automatically producing specific Petri nets from given feature configurations. As the main contribution of this paper, we propose lifting structural analysis of Petri nets to the product line level. This means that we do not need to analyse each Petri net that can be produced from a PNPL separately, but our analysis works on the whole set of Petri nets directly. In this paper, we explain how to lift the analysis of the marked graph property to the PNPL, but other structural analysis techniques [13] can be lifted in a similar way. As a second contribution, we present extensible prototype tool support to model and analyse PNPLs. Our tool is based on Eclipse, and has an extension point to enable contributing further analysis.

In the following, Section 2 introduces PNPLs, Section 3 proposes lifting the analysis of structural properties to the PNPLs and lifts the marked graph property analysis as an example, Section 4 presents tool support, Section 5 compares with related research, and Section 6 concludes the paper and presents lines of future work.

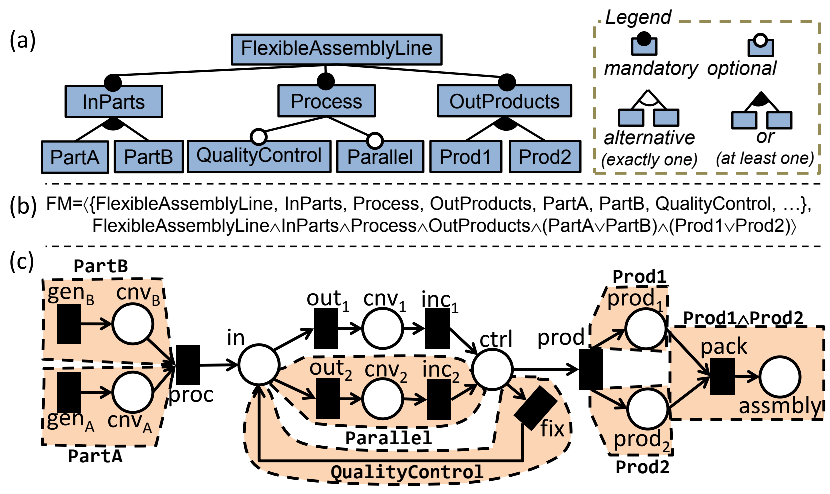


Fig. 1. PNPL modelling a flexible assembly line.

2 Petri Net Product Lines

This section defines PNPLs, and derivation of concrete Petri nets via feature configurations. We consider a simple notion of Petri net, but the approach can be easily adapted to other more complex versions. In particular, we assume that a Petri net is a tuple $PN = (P, T, A)$ where P and T are disjoint sets of places and transitions, and $A \subseteq (P \times T) \cup (T \times P)$ is the set of arcs connecting either places to transitions or vice versa. Given an arc $a \in A$, we use a_0 to refer to its source, and a_1 to refer to its target.

PNPLs build on the notion of a feature model that defines the variability space of possible configurations.

Definition 1 (Feature model). *A* feature model $FM = (F, \phi)$ *consists of a set of features* $F = \{f_1, ..., f_n\}$ *and a propositional formula* ϕ *fixing the allowed feature configurations.*

Example. As an illustration, we will be using a family of Petri nets describing the behaviour of a flexible assembly line. Figure 1(a) shows the feature model using a diagrammatic notation [8], and Figure 1(b) using Def. 1. Our assembly line can be configured to accept one or two kinds of input parts (PartA, PartB), can optionally have a quality control process (QualityControl) and a parallel conveyor (Parallel), and can produce one or two kinds of products (Prod1, Prod2).

A PNPL is a Petri net whose elements can be annotated with boolean formulae over the features of the feature model.

Definition 2 (Petri net product line). *A PNPL* $PNL = (FM, PN, \Phi)$ *is made of a feature model* FM*, a Petri net* PN *(called the 150% Petri net), and a tuple* $\Phi = (\Phi_P, \Phi_T, \Phi_A)$ *of mappings. Each mapping* Φ_X *(for* $X \in \{P, T, A\}$*) consists of pairs* $\langle x, \Phi_x \rangle$ *mapping an element (a place, a transition, an arc)* $x \in X$ *to a propositional formula* Φ_x *(called the presence condition (PC) of* x*) over the features in* FM*.*

PNL *is well-formed if* $\forall a \in A \bullet \Phi_a \Rightarrow \Phi_{a_0} \wedge \Phi_a \Rightarrow \Phi_{a_1}$.

As noticed, we use an annotative approach to facilitate the analysis. The approach relies on the definition of a 150% Petri net that contains all variants of the PNPL, and the assignment of PCs to its elements, so that a particular Petri net can be obtained by removing the elements with false PC (so-called negative variability). Instead, other approaches to SPLs use positive variability [18]. Our method can also be applied to them as long as they permit building a 150% Petri net.

In Def. 2, the well-formedness condition requires the PC of an arc to be stronger than the PC of its source and target elements. This ensures that, if the arc is present in a product Petri net, its source and target elements will be present as well.

Example. Figure 1 shows the PNPL of the flexible assembly line. The 150% Petri net in Figure 1(c) uses dashed regions as a shortcut to assign the same formula to all the elements in the region. For example, formula PartB is attached to transition gen_B, to place cnv_B, and to the arcs from/to place cnv_B.

The way to obtain a particular Petri net from a PNPL is by selecting a subset of the features in its feature model. This is called a feature configuration.

Definition 3 (Feature configuration). *A* valid feature configuration *ρ of a PNPL PNL with feature model $FM = (F, \phi)$ is a subset of its features satisfying ϕ, i.e., ϕ evaluates to true when each variable $f \in \phi$ is substituted by true when $f \in \rho$, and by false otherwise. We use $P(FM) = \{\rho_i\}_{i \in I}$ for the set of all valid configurations of PNL.*

Given a feature configuration, we obtain the corresponding Petri net by removing from the 150% Petri net those elements whose PC is false.

Definition 4 (Petri net derivation). *A Petri net PN_ρ is* derived *from a PNPL PNL with feature model FM using configuration $\rho \in P(FM)$ if PN_ρ contains exactly those elements (places, transitions, arcs) from the 150% Petri net whose PCs are satisfied for the features in ρ.*

We write Prod(PNL) *for the set of all derivable Petri nets from the PNPL PNL.*

Example. Figure 1 admits 36 configurations, each one producing a different Petri net.

Analysing each derivable Petri net of a PNPL one by one can be time-consuming, as the number of Petri nets can be exponential in the number of features in the worst case. Hence, the next section proposes a method to lift the analysis of structural properties to the product line level.

3 Structural Analysis of Petri Net Product Lines

In this paper, we focus on the analysis of structural properties of the set of nets that can be derived from a PNPL PNL. Structural properties depend only on the net topology and are independent of the initial marking [13]. These properties include connectedness, state machine, marked graph, free-choice and place invariants, among others.

As an example, next we provide the definition of the marked graph (MG) property. In a MG Petri net, each place has exactly one input transition and one output transition, whereas each transition may have multiple input and output places. Therefore, a MG allows concurrent and synchronization structures with no conflict.

Definition 5 (Marked graph, from [13]). *A Petri net $PN = (P, T, A)$ is a* marked graph *iff $\forall p_i \in P \bullet |{}^\bullet p| = |p^\bullet| = 1$, being ${}^\bullet p$ and $p^\bullet$ the sets of input and output transitions of the place p, respectively.*

This definition permits analysing a Petri net, but we have a product line of them. To improve the efficiency of their analysis, we do not examine each derivable Petri net in $Prod(PNL)$ separately. Instead, we work at the product line level by analysing the PCs in the 150% Petri net to determine if a particular element (place, transition or arc) is in $Prod(PNL)$. For this purpose, first we lift the definition of the MG property to the product line level. A PNPL is a MG if all its derivable nets are MGs.

Definition 6 (MG product line). *A Petri net product line PNL is a marked graph iff $\forall PN_\rho \in Prod(PNL) \bullet PN_\rho$ is a marked graph.*

In other words, if we can derive from the product line PNL a net that is not a MG, then PNL is not a MG product line. In particular, given a feature configuration ρ, a Petri net derivation PN_ρ is not a MG if it has a place p with more than one input transition or more than one output transition. To analyse this without explicitly generating PN_ρ, we extend the elements of the pre- and post-sets of each place p with the PCs of its incoming and outgoing arcs.

Definition 7 (Lifted pre-/post-sets of a place). *Given a PNPL $PNL = (FM, PN = (P, T, A), \Phi)$, and a place $p \in P$, the lifted pre-set of p is ${}^\circ p = \{(t, \Phi_{(t,p)}) \mid (t,p) \in A\}$, while its lifted post-set is $p^\circ = \{(t, \Phi_{(p,t)}) \mid (p,t) \in A\}$.*

Remark. In the previous definition, we can use the PC of the arc Φ_a instead of the PC of the transition Φ_{a_1} because, according to Def. 2, in a well-formed PNPL, $\Phi_a \Rightarrow \Phi_{a_0} \wedge \Phi_a \Rightarrow \Phi_{a_1}$, and so, $\Phi_a \wedge \Phi_{a_0} \equiv \Phi_a \equiv \Phi_a \wedge \Phi_{a_1}$.

The size of the lifted pre-set ${}^\circ p = \{(t_0, \Phi_{(t_0,p)}), ..., (t_n, \Phi_{(t_n,p)})\}$ of a place p will depend on the feature configuration ρ. To analyse the MG product line property, we require that its size is one for every possible configuration. This is the case if the following formula is true:

$$\begin{aligned}\Phi_{{}^\circ p} \triangleq\ & (\Phi_{(t_0,p)} \wedge \neg\Phi_{(t_1,p)} \wedge ... \wedge \neg\Phi_{(t_n,p)}) \vee \\ & (\neg\Phi_{(t_0,p)} \wedge \Phi_{(t_1,p)} \wedge ... \wedge \neg\Phi_{(t_n,p)}) \vee \\ & ... \\ & (\neg\Phi_{(t_0,p)} \wedge \neg\Phi_{(t_1,p)} \wedge ... \wedge \Phi_{(t_n,p)})\end{aligned} \tag{1}$$

The formula is made of a disjunction of conjunctions, where only one term in each conjunction can be true. This ensures that, regardless of the configuration, the pre-set of the place will have size one. The lifted post-set of a place Φ_{p° is defined similarly.

This way, a PNPL includes some Petri net that is a MG if there is a feature configuration ρ such that for every place p in the PNPL PNL:

- p is not in PN_ρ, therefore Φ_p is false; or
- p is in PN_ρ, and therefore $\Phi_{{}^\circ p}$ and Φ_{p° need to be true.

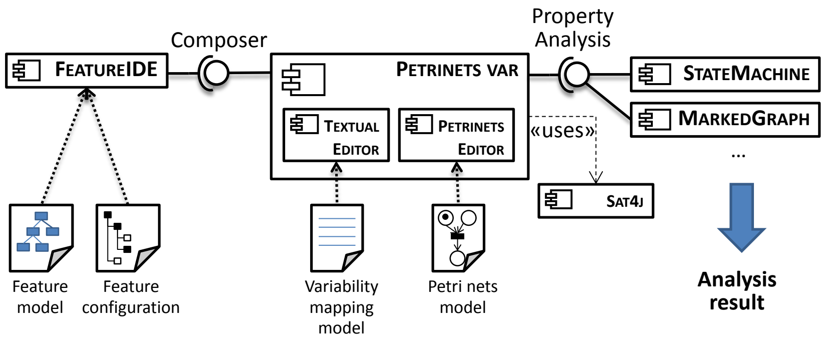


Fig. 2. Tool architecture.

We can express these conditions as the logical formula $\Phi = (\bigwedge_{p \in P}[\neg\Phi_p \vee (\Phi_p \wedge \Phi_{{}^\circ p} \wedge \Phi_{p^\circ})]$. This way, if $SAT(\phi \wedge \Phi)$ (with SAT a predicate that holds if the formula is satisfiable, and ϕ the formula of the feature model), then some Petri net in the PNPL is a MG. We can use a constraint solver to obtain a feature configuration that satisfies the formula, if such a configuration exists. The Petri net derived using this feature configuration is ensured to be a MG.

Conversely, the feature configurations that yield nets which are not MGs are those making the formula $\Phi = (\bigwedge_{p \in P}[\neg\Phi_p \vee (\Phi_p \wedge (\neg\Phi_{{}^\circ p} \vee \neg\Phi_{p^\circ}))]$ true.

Example. In the PNPL of Figure 1, the interesting cases are those for places in and ctrl. In the latter case, any Petri net that contains either both transitions inc_1 and inc_2, or both transitions prod and fix, is not a MG because place ctrl would have either two incoming or two outgoing arcs. This is the case for the Petri nets derived from configurations that select the features Parallel or QualityControl. Similarly, place in will have two incoming arcs for configurations that select the feature QualityControl, and two outgoing arcs for configurations that select the feature Parallel, resulting in nets that are not MGs.

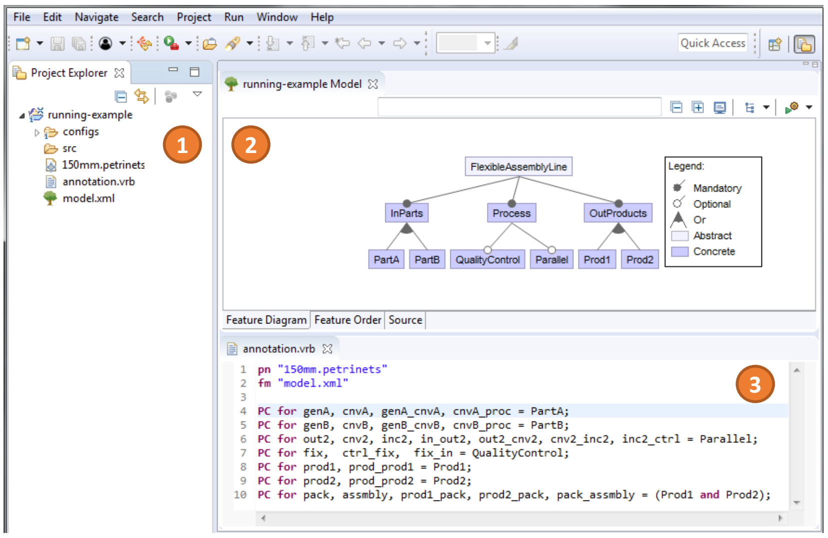
The analysis of other structural properties, like state-machine, free-choice or asymmetric choice, can be lifted in a very similar way.

4 Tool Support and Assessment

In this section we present a prototype realization of our approach (Section 4.1), and assess the efficiency gain of our analysis w.r.t. an enumerative approach (Section 4.2).

4.1 Architecture and tool support

We have implemented an Eclipse plugin, called Petrinets var, which supports the presented approach. Figure 2 shows its architecture. Our tool provides two dedicated editors: one to specify the 150% Petri net, and another to assign PCs to its elements in a so-called variability mapping model. We use the Eclipse Modeling Framework (EMF) [20] as the underlying modelling technology, and therefore, both models are EMF-based and conform to their respective Ecore meta-models.



```
pn "150mm.petrinets"
fm "model.xml"

PC for genA, cnvA, genA_cnvA, cnvA_proc = PartA;
PC for genB, cnvB, genB_cnvB, cnvB_proc = PartB;
PC for out2, cnv2, inc2, in_out2, out2_cnv2, cnv2_inc2, inc2_ctrl = Parallel;
PC for fix,  ctrl_fix,  fix_in = QualityControl;
PC for prod1, prod_prod1 = Prod1;
PC for prod2, prod_prod2 = Prod2;
PC for pack, assmbly, prod1_pack, prod2_pack, pack_assmbly = (Prod1 and Prod2);
```

Fig. 3. Screenshot of our tool.

We rely on FeatureIDE [12] to specify the feature model and feature configurations. FeatureIDE provides an extension point Composer that our tool instantiates to automate the derivation of specific Petri nets from the 150% Petri net given a feature configuration. In its turn, our tool defines an extension point, called Property Analysis, that allows extending the tool with new analysis methods. Our framework provides facilities to transform the conjunction of the analysis formula and the formula of the feature model into Conjunctive Normal Form (CNF) as the Sat4J solver requires. These facilities can be used by any analysis. We currently provide two instances of this extension point to analyse whether some/all Petri nets in a PNPL are state machines or marked graphs.

Figure 3 shows a screenshot of our tool. The Eclipse project explorer (label *1*) contains the FeatureIDE project with the definition of the PNPL used as a running example. This project is configured with our composer and declares the 150% Petri net (file *150mm.petrinets*), the feature model (file *model.xml* that is being edited in the window labelled *2*), and the variability model (file *annotation.vrb* that is being edited in the window labelled *3*). As the figure shows, there are dedicated editors for each kind of file. A popup menu on the variability model allows selecting the lifted analysis to perform.

Note that, as a proof of concept, our current implementation uses its own EMF meta-model to represent 150% Petri nets. This meta-model supports a simple notion of net like the one we have used in the paper. However, we are planning to use the standard Petri Net Markup Language (PNML) [16] instead, for which there is an EMF implementation available.

4.2 Efficiency assessment

Next, we report on a small-scale experiment to assess the efficiency gain of our lifted analyses, compared to generating all derivable nets in a PNPL and analysing each net separately. In particular, we measure the time for analysing the MG and state-machine (SM) properties.

The experiment uses the example of Figure 1, for which the analyses report that some Petri nets in the PNPL are neither MGs nor SMs. Table 1 shows the average analysis time in milliseconds of running 10 times each analysis, where we discarded the first execution to avoid warmup effects. As it can be observed, both lifted analyses are two orders of magnitude faster than the time to generate and analyse each net in isolation. We expect further efficiency gains with bigger PNPLs (with more features), but performing more experiments is up to future work.

Structural analysis	**Lifted analysis of PNPL**	**Analysis of all products**
Marked graph	38	3356
State machine	41	4603

Table 1. Analysis time (ms) for MG and SM.

5 Related Work

The main analysis techniques for Petri nets can be classified into three groups [5]: i) enumeration, ii) transformation (mainly reduction), and iii) structural. *Enumeration* methods are based on the construction of a reachability/coverability graph, but they suffer the *state explosion problem*. Transformation methods obtain a slice of a Petri net that is easier to analyse but preserves the properties under study [3]. Structural analysis techniques are based on the net structure and its initial marking, and can be divided into two subgroups: *linear programming techniques* based on the state equation, and *graph-based techniques* based on "ad hoc" reasoning frequently derived from the firing rule. A survey on Petri nets models and their analysis techniques can be found at [19].

There are several mechanisms to model variability for SPL. Most of them can be classified into annotation-based and composition-based techniques [1]. In annotation-based approaches, parts of a model are annotated with information about their mapping to products of the product line. They are widely used since they are easy to implement. Nevertheless, they work under the closed world assumption, i.e., the set of features is fixed. In composition-based modelling, the product line is decomposed into separate modules representing features that can be composed to derive products. They support positive variability, that is, composition units are added on demand. Surveys on SPL modelling techniques can be found in [2,4,21].

Just like us, some works have added variability to Petri nets using SPL techniques. *Feature Petri nets*(FN) [14] extend Petri nets to allow modelling the behaviour of an entire SPL. A FN transition is activated if its input places are marked and its application condition (a logical constraint over features) is true under the current configuration state. *Dynamic feature Petri nets* (DFPN) [15] extend FN to control feature bindings at runtime, and allow the evaluation of some dynamic properties using model checking. These works lift the analysis based on the reachability graph to the product line level, by adding presence conditions to this graph. They follow an annotative approach to model variability. Similarly, some works have used variability in Petri nets with the purpose of expressing variability in higher-level languages – like activity diagrams – and use a variable reachability graph for analysis [7]. This work is also an annotative approach for SPL. With respect to these works, our variability model is more general: [7] only supports variability in edges, [15] supports variability just in arcs and transitions, while our approach supports presence conditions in arcs, places and transitions. With respect to analysis, while they focus on the reachability graph, we lift structural analysis techniques.

In addition to SPL methods, other techniques to handle variability in Petri nets have been proposed. *Conditional Petri nets* [22] associate to each transition a condition defined with the family of $\mathcal{L}$ languages. Therefore, a transition is conditioned by the transition sequence previously applied. Likewise, *logical Petri nets* [10] limit transition firing by means of constraints on first-order logic. Reconfigurable nets [11] can change the net topology at runtime by means of rewriting rules. Instead, PNPLs are static: a configuration needs to be provided to derive a Petri net. Regarding analysis of model-based product lines, Czarnecki and Pietroszek [6] proposed an approach to check whether all possible derivable models satisfy the OCL constraints of their meta-model. We may have encoded the MG property in OCL and used that technique. However, our solution permits generating specific constraints for the analysed PNML (instead of relying on one generic OCL constraint), which therefore can be solved using simpler and potentially more efficient standard SAT-solving techniques. Instead, Czarnecki's approach requires extending an existing OCL-based checker to consider presence conditions.

Concerning SPL analysis of temporal properties, Legay et al. [9] represent the behaviour of variability-intensive systems by means of an extension of transition systems, called Feature Transition System. These authors also propose in [4] model checking algorithms to verify of all products of a SPL. Unlike this approach, we only focus on static properties, but we would plan to explore behavioural properties in further works.

Altogether, to the best of our knowledge, there are no previous works on the analysis of structural properties of PNPLs. Our work is a first step in this direction, which we have realized in practice through extensible tooling.

6 Conclusions and Future Work

In this paper, we propose the notion of Petri net product line, show how to analyse structural properties (specifically the marked graph property) at the product line level, and presented an extensible prototype on top of FeatureIDE.

In the future, we plan to support more types of static analysis techniques, exploit compositionality of Petri nets in these analyses, and perform more thorough experiments. Moreover, our idea is to develop a domain-specific language to express such analyses, which then can be compiled into standard SAT solving procedures. At the technical level, we will use the PNML meta-model, to ease the connection of our approach with Petri net tools like CPN Tools [23]. Finally, we are also planning to explore the lifting of dynamic analyses, and also the consideration of variability of the Petri net language itself.

Acknowledgments

Work funded by the Spanish Ministry of Science (project MASSIVE, RTI2018-095255-B-I00) and the R&D programme of Madrid (project FORTE, P2018/TCS-4314). We thank the anonymous referees for their useful comments.

References

1. Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines - Concepts and Implementation*. Springer, 2013.
2. Fabian Benduhn, Thomas Thüm, Malte Lochau, Thomas Leich, and Gunter Saake. A Survey on Modeling Techniques for Formal Behavioral Verification of Software Product Lines. In *Proceedings of the 9th International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '15, pages 80:80–80:87, New York, NY, USA, 2015. ACM.
3. Gérard Berthelot. Transformations and decompositions of nets. In *Advances in Petri nets*, volume 254 of *LNCS*, pages 359–376. Springer, 1987.
4. Andreas Classen, Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, Axel Legay, and Jean-François Raskin. Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. *IEEE Trans. Software Eng.*, 39(8):1069–1089, 2013.
5. José Manuel Colom, Enrique Teruel, and Manuel Silva. *Performance models for discrete event systems with synchronisations: Formalisms and analysis techniques*. Editorial KRONOS, 1998.
6. Krzysztof Czarnecki and Krzysztof Pietroszek. Verifying feature-based model templates against well-formedness OCL constraints. In *Proc. GPCE*, pages 211–220. ACM, 2006.
7. André Heuer, Vanessa Stricker, Christof J. Budnik, Sascha Konrad, Kim Lauenroth, and Klaus Pohl. Defining variability in activity diagrams and petri nets. *Sci. Comput. Program.*, 78(12):2414–2432, 2013.
8. Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.
9. Axel Legay, Gilles Perrouin, Xavier Devroey, Maxime Cordy, Pierre-Yves Schobbens, and Patrick Heymans. On Featured Transition Systems. In *Proceedings of Theory and Practice of Computer Science - 43rd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2017)*, volume 10139 of *Lecture Notes in Computer Science*, pages 453–463. Springer, 2017.
10. Wei Liu, Pin Wang, Yuyue Du, Mengchu Zhou, and Chun Yan. Extended logical Petri nets-based modeling and analysis of business processes. *IEEE Access*, 5:16829–16839, 2017.
11. Marisa Llorens and Javier Oliver. Structural and dynamic changes in concurrent systems: Reconfigurable Petri nets. *IEEE Trans. Computers*, 53(9):1147–1158, 2004.
12. Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. *Mastering software variability with FeatureIDE*. Springer, 2017. See also `https://featureide.github.io/`.
13. Tadao Murata. Petri Nets: Properties, Analysis and Applications. *Proc. IEEE*, 77(4):541–580, 1989.
14. Radu Muschevici, Dave Clarke, and José Proença. Feature Petri nets. In *SPLC Workshops*, pages 99–106. Lancaster University, 2010.
15. Radu Muschevici, José Proença, and Dave Clarke. Feature nets: behavioural modelling of software product lines. *Software & Systems Modeling*, 15(4):1181–1206, 2016.

16. Petri Net Markup Language. www.pnml.org.
17. Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering. Foundations, Principles and Techniques*. Springer-Verlag Berlin Heidelberg, 2005.
18. Christoph Seidl, Ina Schaefer, and Uwe Aßmann. DeltaEcore – A Model-Based Delta Language Generation Framework. In *Modellierung*, volume 225 of *LNI*, pages 81–96, Bonn, 2014. GI.
19. Manuel Silva. Half a century after Carl Adam Petri's Ph.D. thesis: A perspective on the field. *Annual Reviews in Control*, 37(2):191 – 219, 2013.
20. David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.
21. Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. A Classification and Survey of Analysis Strategies for Software Product Lines. *ACM Comput. Surv.*, 47(1):6:1–6:45, June 2014.
22. Ferucio Laurentiu Tiplea, Toader Jucan, and Cristian Masalagiu. Conditional Petri net languages. *Elektronische Informationsverarbeitung und Kybernetik*, 27(1):55–66, 1991.
23. Michael Westergaard and Lars Michael Kristensen. The Access/CPN framework: A tool for interacting with the CPN tools simulator. In *Proc. PETRI NETS*, volume 5606 of *Lecture Notes in Computer Science*, pages 313–322. Springer, 2009.