Verifying Modal Workflow Specifications using Constraint Solving

Hadrien Bride[1,2], Olga Kouchnarenko[1,2], and Fabien Peureux[1]

[1] Institut FEMTO-ST – UMR CNRS 6174, University of Franche-Comté
16, route de Gray, 25030 Besançon, France
{hbride,okouchna,fpeureux}@femto-st.fr
[2] Inria Nancy Grand Est – CASSIS Project
Campus Scientifique, BP 239, 54506 Vandœuvre-lès-Nancy cedex
{hadrien.bride,olga.kouchnarenko}@inria.fr

Abstract. Nowadays workflows are extensively used by companies to improve organizational efficiency and productivity. This paper focuses on the verification of modal workflow specifications using constraint solving as a computational tool. Its main contribution consists in developing an innovative formal framework based on constraint systems to model executions of workflow Petri nets and their structural properties, as well as to verify their modal specifications. Finally, an implementation and promising experimental results constitute a practical contribution.

Keywords: Modal specifications, Workflow Petri nets, Verification of Business Processes, Constraint Logic Programming.

1 Introduction

Nowadays workflows are extensively used by companies in order to improve organizational efficiency and productivity by managing the tasks and steps of business processes. Intuitively, a workflow system describes the set of possible runs of a particular system/process. Among modelling languages for workflow systems [1, 2], workflow Petri nets (WF-nets for short) are well suited for modelling and analysing discrete event systems exhibiting behaviours such as concurrency, conflict, and causal dependency between events as shown in [3, 4]. They represent finite or infinite-state processes in a readable graphical and/or a formal manner, and several important verification problems, like reachability or soundness, are known to be decidable. With the increasing use of workflows for modelling crucial business processes, the verification of specifications, i.e. of desired properties of WF-nets, becomes mandatory. To accompany engineers in their specification and validation activities, modal specifications [5] have been designed to allow, e.g., *loose* specifications with restrictions on transitions. Those specifications are notably used within refinement approaches for software development. Modal specifications impose restrictions on the possible refinements by indicating whether activities (transitions in the case of WF-nets) are *necessary* or *admissible*. Modalities provide a flexible tool for workflow development as decisions can be delayed to later steps (refinements) of the development life cycle.

This paper focuses on the verification of modal WF-net specifications using constraint solving as a computational tool. More precisely, given a modal WF-net, a constraint system modelling its correct executions is built and then computed to verify modal properties of interest over this workflow specification. After introducing a motivating example in Sect. 2 and defining preliminaries on WF-nets with their modal specifications in Sect. 3, the paper describes its main contribution in Sect. 4. It consists in developing a formal framework based on constraint systems to model executions of WF-nets and their structural properties, as well as to verify their modal specifications. An implementation supporting the proposed approach and promising experimental results constitute a practical contribution in Sect. 5. Finally, a discussion on related work is provided before concluding.

2 Motivating Example

Our approach for verifying modal specifications is motivated by the increasing criticity of business processes, which define the core of many industrial companies and require therefore to be carefully designed. In this context, we choose a real-life example of an industrial business workflow, which is directly driven by the need to verify some behavioural properties possibly at the early stage of development life cycle, before going to implementation. This example concerns a proprietary issue tracking system used to manage bugs and issues requested by the customers of a tool provider company[3]. Basically, this system enables the provider to create, update and drop tickets reporting on customer's issues, and thus provides knowledge base containing problem definition, improvements and solutions to common problems, request status, and other relevant data needed to efficiently manage all the company projects. It must also be compliant with respect to several rules ensuring that business processes are suitable as well as streamlined, and implement best practices to increase management effectiveness.

Figure 1 depicts an excerpt of the corresponding business process—specified from textual requirements by a business analyst team of the company—modelled using a Petri net workflow (WF-net). The main process, in the top left model, is defined by two possible distinct scenarii (SubA and SubB), which are described by two other workflows. In the figure, big rectangles (as for SubA and SubB) define other workflows. Some of them are not presented here: this example is indeed deliberately simplified and abstracted to allow its small and easily understandable presentation in this paper; its complete WF-net contains 91 places and 113 transitions. For this business process, the goal is to verify, at the specification or design stage of the development, some required behavioural properties (denoted p_i for later references) derived from textual requirements and business analyst expertise such as: during a session, either the scenario SubA or the scenario SubB (and not both of them) must be executed (p_1); when the scenario SubB is considered then the user must login (p_2); once a critical situation request is

[3] For confidentiality reasons, the details about this case-study are not given.

pending, it can either be updated, validated and dispatched, or closed (p_3); once a critical situation is created, it can be updated and closed (p_4); at any time, a service request can be upgraded to a critical situation request (p_5); a logged user must logout to exit the current session (p_6).

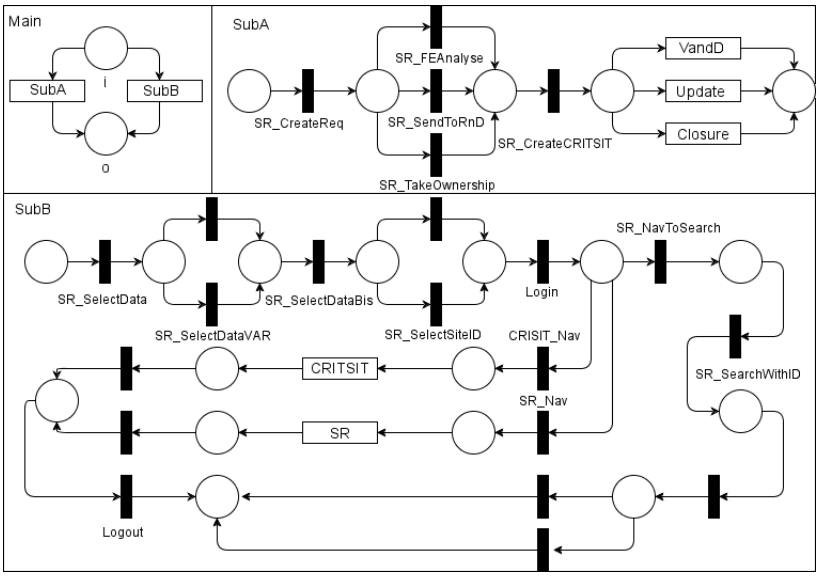


Fig. 1. Excerpt of issue tracking system WF-net

To ensure the specified business process model verifies this kind of business rules, there is a need to express and assess them using modal specifications. However, usual modal specifications are relevant to express properties on single transition by specifying that a transition shall be a (*necessary*) *must*-transition or a (*admissible*) *may*-transition, but they do not allow to express requirements on several transitions. For instance, expressing the property p_1 using usual modal specifications allows to specify that transitions of SubA and SubB shall be *may*-transitions. Nevertheless, such formula does not ensure that SubA or SubB has to occur in a exclusive manner, and specifying some transitions as *must*-transition cannot tackle this imprecision. That is why we propose in this paper to extend the expressiveness of usual modal specifications by using modalities over a set of transitions, and to define dedicated algorithms to automate their verification.

3 Preliminaries

This section reminds background definitions and the notations used throughout this article. It briefly describes workflow Petri nets as well as some of their behavioural properties. Modal specifications are also introduced.

3.1 Workflow Petri Nets

As mentioned above, workflows can be modelled using a class of Petri nets, called the workflow nets (WF-nets). Figure 2 provides an example of a Petri net where the places are represented by circles, the transitions by rectangles, and the arcs by arrows.

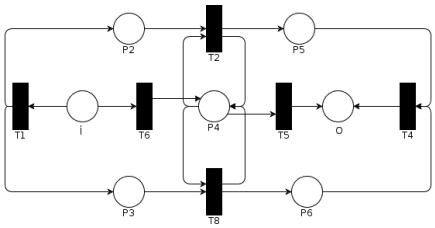


Fig. 2. Basic example of a WF-net (ex1)

Definition 1 (Petri net). *A Petri net is a tuple* (P, T, F) *where* P *is a finite set of places,* T *is a finite set of transitions* ($P \cap T = \emptyset$)*, and* $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs.*

Let $g \in P \cup T$ and $G \subseteq P \cup T$. We use the following notations: $g^\bullet = \{g' | (g, g') \in F\}$, ${}^\bullet g = \{g' | (g', g) \in F\}$, $G^\bullet = \cup_{g \in G}\, g^\bullet$, and ${}^\bullet G = \cup_{g \in G}\, {}^\bullet g$.
A *marking* of Petri net is a function $M : P \to \mathbb{N}$. The marking represents the number of token(s) on each place. The marking of a Petri net evolves during its execution. Transitions change the marking of a Petri net according to the following *firing rules*. A transition t is *enabled* if and only if $\forall p \in {}^\bullet t, M(p) \geq 1$. When an *enabled* transition t is *fired*, it *consumes* one token from each place of ${}^\bullet t$ and *produces* one token for each place of $t^\bullet$. With respect to these rules, a transition t is *dead* at marking M if it is not *enabled* in any marking M' reachable from M. A transition t is *live* if it is not *dead* in any marking reachable from the initial marking. A Petri net system is *live* if each transition is *live*.

Definition 2 (WF-net). *A Petri net* $PN = (P, T, F)$ *is a WF-net (Workflow net) if and only if* PN *have two special places* i *and* o*, where* ${}^\bullet i = \emptyset$ *and* $o^\bullet = \emptyset$*, and for each node* $n \in (P \cup T)$ *there exists a path from* i *to* o *passing through* n*.*

For example, the Petri net in Fig. 2 is a WF-net. Let us notice that in the context of *workflow*, specifiers are used to consider *ordinary* Petri nets [6], i.e. Petri nets with arcs of weight 1. In the rest of the paper, the following notations are used:

- $M_\emptyset$: the marking defined by $\forall p \in P, M(p) = 0$,
- $M_a \xrightarrow{t} M_b$: the transition t is *enabled* in marking M_a, and *firing* it results in the marking M_b,
- $M_a \to M_b$: there exists t such that $M_a \xrightarrow{t} M_b$,
- $M_1 \xrightarrow{\sigma} M_n$: the sequence of transitions $\sigma = t_1, t_2, ..., t_{n-1}$ leads from the marking M_1 to the marking M_n (i.e. $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} ... \xrightarrow{t_{n-1}} M_n$),
- $M_a \xrightarrow{*} M_b$: the marking M_b is reachable from marking M_a (i.e. there exists σ such that $M_a \xrightarrow{\sigma} M_b$).

We denote $M_{i(k)}$ the initial marking (i.e. $M_i(n) = k$ if $n = i$, and 0 otherwise) and $M_{o(k)}$ the final marking (i.e. $M_o(n) = k$ if $n = o$, and 0 otherwise). When k is not specified, it equals 1. A sequence σ of transitions of a Petri net is an execution if there are M_a, M_b such that $M_a \xrightarrow{\sigma} M_b$. A *correct* execution of a

WF-net is an execution σ such that $M_i \xrightarrow{\sigma} M_o$. For example, $M_i \xrightarrow{\sigma} M_o$ where $\sigma = T6,T5$ is a *correct* execution of the WF-net in Fig. 2. The behaviour of a WF-net is defined as the set Σ of all its correct executions. For the transition t and the execution σ, the function $O_t(\sigma)$ is the number of occurrences of t in σ.

Definition 3 (Siphon/Trap). *Let $N \subseteq P$ such that $N \neq \emptyset$:*

- *N is a trap if and only if $N^\bullet \subseteq {}^\bullet N$.*
- *N is a siphon if and only if ${}^\bullet N \subseteq N^\bullet$.*

Figure 3(a) displays an example of a Petri net with a siphon. Let $N = \{P4\}$, since ${}^\bullet N = \{T1, T8\} \subseteq N^\bullet = \{T1, T8\}$, the set of places $N = \{P4\}$ is a siphon.

Theorem 1 (from [7]). *An ordinary Petri net without siphons is live.*

3.2 WF-nets with Modalities

Modal specifications have been designed to allow *loose* specifications to be expressed by imposing restrictions on transitions [5]. They allow specifiers to indicate that a transition is *necessary* or just *admissible*. In the framework of WF-nets, this concept provides two kinds of transitions: the *must*-transitions and the *may*-transitions. A *may*-transition (resp. *must*-transition) is a transition fired by at least one execution (resp. all the executions) of the procedure modelled by a WF-net.

While basic modal specifications are useful, they usually lack expressiveness for real-life applications, as only individual transitions are concerned with. We propose to extend modal specifications to express requirements on several transitions and on their causalities. To this end, the language S of well-formed *modal* specification formulae is inductively defined by : $\forall t \in T, t$ is a well-formed *modal* formula, and given $A_1, A_2 \in S$, $A_1 \wedge A_2$, $A_1 \vee A_2$, and $\neg A_1$ are well-formed *modal* formulae. These formulae allow specifiers to express modal properties about WF-nets' correct executions. Any modal specification formula $m \in S$ can be interpreted as a *may*-formula or a *must*-formula. A *may*-formula describes a behaviour that has to be ensured by at least one correct execution of the WF-net. The set of *may*-formulae forms a subset of CTL formulae where only the *possibly* operator (i.e. along at least one path) is used. On the other hand, a *must*-formula describes a behaviour that has to be ensured by all the correct executions of the WF-net. The set of *must*-formulae forms a subset of CTL formulae where only the *inevitably* operator (i.e. along all paths) is used. For example, for the WF-net ex2 of 3(b), the *may*-formula $T9$ means that there exists a correct execution firing transition $T9$ at least once (i.e. $T9$ is a *may*-transition). More complex behaviours can be expressed. For example, the *must*-formula $(T8 \wedge T9) \wedge (\neg T6 \vee T5)$ means that $T8$ and $T9$ must be fired by every correct execution, and if an execution fires $T6$ then $T5$ is also fired at least once. Formally, given $t \in T$:

- $PN \models_{may} t$ if and only if $\exists\ \sigma \in \Sigma.\ O_t(\sigma) > 0$, and
- $PN \models_{must} t$ if and only if $\forall\ \sigma \in \Sigma.\ O_t(\sigma) > 0$.

Further, given a well-formed *may*-formula (resp. *must*-formula) $m \in S$, a WF-net PN satisfies m, written $PN \models_{may} m$ (resp. $PN \models_{must} m$), when at least one (resp. all) correct execution(s) of PN satisfies (resp. satisfy) m. The semantics of $\neg, \vee$ and $\wedge$ is standard.

Definition 4 (Modal Petri net). *A modal Petri net* $MPN = (P, T, F, m, M)$ *is a Petri net* $PN = (P, T, F)$ *together with a modal specification* (m, M) *where:*

- $m \in S$ *is a well-formed* modal *must-formula*[4]*, and*
- $M \subset S$ *is a set of well-formed* modal *may-formulae.*

We say that a WF-net PN satisfies a modal specification (m, M) if and only if $PN \models_{must} m$ and $\forall m' \in M, PN \models_{may} m'$.

3.3 Hierarchical Petri Nets

Modelling large and intricate WF-nets can be a difficult task. Fortunately, similarly to modular programming, WF-nets can be designed using other WF-nets as building blocks. One of the simple methods used to construct composed WF-nets is by transitions substitution. A composed WF-net built using this method has special transitions that represent several whole (composed or not) WF-nets. The composed WF-nets can then be viewed as WF-nets with multiple layers of details; they are called hierarchical WF-nets. While this does not add any expressiveness to WF-nets, it greatly simplifies the modelling work, allowing to model small parts of the whole process that are combined into a composed WF-net.

3.4 Constraint System

A constraint system is defined by a set of constraints (properties), which must be satisfied by the solution of the problem it models. Such a system can be represented as a Constraint Satisfaction Problem (CSP) [8]. It is such that each variable appearing in a constraint should take its value from its domain. Formally, a CSP is a tuple $\Omega =< X, D, C >$ where X is a set of variables $\{x_1, \ldots, x_n\}$, D is a set of domains $\{d_1, \ldots, d_n\}$, where d_i is the domain associated with the variable x_i, and C is a set of constraints $\{c_1(X_1), \ldots, c_m(X_m)\}$, where a constraint c_j involves a subset X_j of the variables of X. A CSP thus models NP-complete problems as search problems where the corresponding search space is the Cartesian product space $d_1 \times \ldots \times d_n$. The solution of a CSP Ω is computed by a labelling function $\mathcal{L}$, which provides a set v (called valuation function) of tuples assigning each variable x_i of X to one value from its domain d_i such that all the constraints C are satisfied. More formally, v is consistent—or satisfies a constraint $c(X)$ of C—if the projection of v on X is in $c(X)$. If v satisfies all the constraints of C, then Ω is a consistent or satisfiable CSP. In the rest of the paper, the predicate $SAT(C, v)$ is true if the corresponding CSP Ω is made satisfiable by v, and the predicate $UNSAT(C)$ is true if there exists no such v.

[4] We only need a single *must*-formula because $PN \models_{must} m_1 \wedge PN \models_{must} m_2$ if and only if $PN \models_{must} (m_1 \wedge m_2)$, for any two *must*-formulae m_1 and m_2.

Using Logic Programming for solving a CSP has been investigated for many years, especially using Constraint Logic Programming over Finite Domains, written CLP(FD) [9]. This approach basically consists in embedding consistency techniques into Logic Programming by extending the concept of logical variables to the one of the domain-variables taking their value in a finite discrete set of integers. In this paper, we propose to use CLP(FD) to solve the CSP that represent the modal specifications to be verified.

4 Verification of Modal Specifications

To verify a modal specification of a WF-net, we model the executions of a WF-net by a constraint system, which is then solved to validate or invalidate the modal specifications of interest.

4.1 Modelling Executions of WF-nets

Considering a WF-net $PN = (P, T, F)$, we start by modelling all the executions leading from a marking M_a to a marking M_b, i.e. all σ such that $M_a \xrightarrow{\sigma} M_b$.

Definition 5 (Minimum places potential constraint system). *Let $PN = (P, T, F)$ be a WF-net and M_a, M_b two markings of PN, the minimum places potential constraint system $\varphi(PN, M_a, M_b)$ associated with it is:*

$$\forall p \in P.\nu(p) = \sum_{t \in p^\bullet} \nu(t) + M_b(p) = \sum_{t \in {}^\bullet p} \nu(t) + M_a(p) \quad (1)$$

where $\nu : P \times T \to \mathbb{N}$ is a valuation function.

Equation (1) expresses the fact that for each place, the number of token(s) entering it plus the number of token(s) in M_a is equal to the number of tokens leaving it plus the number of token(s) in M_b. This constraint system is equivalent with respect to solution space to the state equation, aka the fundamental equation, of Petri nets, the only difference is that (1) explicitly gives information about the places involved in the modelled execution.

Theorem 2. *If $M_a \xrightarrow{*} M_b$ then a valuation satisfying $\varphi(PN, M_a, M_b)$ exists.*

Proof. Let $\sigma = t_1, t_2, ..., t_n$ and $M_a \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 ... M_{n-1} \xrightarrow{t_n} M_b$. We define:

- $\forall t \in T.\nu(t) = O_t(\sigma)$
- $\forall p \in P.\nu(p) = \sum_{j \in \{1,2,...,n-1\} \cup \{a,b\}} M_j(p)$

Then $\forall p \in P$:

- $\sum_{j \in \{1,2,...,n-1\} \cup \{a,b\}} M_j(p) = \sum_{t \in p^\bullet} O_t(\sigma) + M_b(p) = \sum_{t \in {}^\bullet p} O_t(\sigma) + M_a(p)$. Indeed, as the WF-net is an ordinary Petri net, the sum of tokens in all markings of a place is equal to the sum of the occurrences of transitions producing (resp. consuming) a token at this place plus the number of token(s) in marking M_b (resp. M_a).

– $\nu(p) = \sum_{t \in p^\bullet} \nu(t) + M_b(p) = \sum_{t \in {}^\bullet p} \nu(t) + M_a(p)$.

Consequently, ν is a valuation satisfying $\varphi(PN, M_a, M_b)$.

For example, $M_i \xrightarrow{\sigma} M_o$ where $\sigma = T6, T5$ is a correct execution of the WF-net in Fig. 2, therefore we can find a valuation $\nu(n) = 1$ if $n \in \{T6, T5, i, o\}$, and $\nu(n) = 0$ otherwise. By Th. 2, this valuation ν satisfies the constraints system $\varphi(ex1, M_i, M_o)$.

Theorem 2 allows to conclude that a WF-net PN does not have any correct executions if $\varphi(PN, M_i, M_o)$ does not have a valuation satisfying it. However, even if there is a valuation satisfying $\varphi(PN, M_i, M_o)$, it does not necessary correspond to a correct execution. For example, the valuation $\nu(n) = 1$ if $n \in \{T1, T2, T8, T4, i, P2, P3, P5, P6, o\}$, $\nu(n) = 2$ if $n \in \{P4\}$, and $\nu(n) = 0$ otherwise, satisfies $\varphi(ex1, M_i, M_o)$ but it does not correspond to any correct execution. This is due to the fact that transitions $T2$ and $T8$ cannot fire simultaneously using as an input token an output token of each other. Consequently, the set of solutions of $\varphi(PN, M_i, M_o)$ constitutes an over-approximation of the set of correct executions of PN. In the rest of the paper, the solutions of $\varphi(PN, M_i, M_o)$ that do not correspond to correct executions of PN are called spurious solutions. Hence our goal is to refine this over-approximation in order to be able to conclude on properties relative to all correct executions of a WF-net.

4.2 Verifying Structural Properties over Executions

While considering the modelling of WF-net executions, siphons and traps have interesting structural features. Indeed, an unmarked siphon will always be unmarked, and a marked trap will always be marked. Therefore a WF-net can only have siphons composed of at least the place i and traps composed of at least place o. Theorem 3 allows to conclude on the existence of a siphon in a WF-net.

Theorem 3. *Let $\theta(PN)$ be the following constraint system associated with a WF-net $PN = (P, T, F)$:*

– $\forall p \in P, \forall t \in^\bullet p. \sum_{p' \in {}^\bullet t} \xi(p') \geq \xi(p)$
– $\sum_{p \in P} \xi(p) > 0$

where $\xi : P \to \{0, 1\}$ is a valuation function. PN contains a siphon if and only if there is a valuation satisfying $\theta(PN)$.

Proof. ($\Leftarrow$) Let ξ be a valuation satisfying $\theta(PN)$, and $N \subseteq P$ such that $\xi(p) = 1 \Leftrightarrow p \in N$. Then $\forall p \in N, \forall t \in {}^\bullet p. \sum_{p' \in {}^\bullet t} \xi(p') \geq 1$, which implies ${}^\bullet N \subseteq N^\bullet$. Consequently, N is a siphon.
($\Rightarrow$) Suppose that N is a siphon then obviously the valuation $\xi(p)$ defined as: $\xi(p) = 1$ if $p \in N$, and 0 otherwise, satisfies $\theta(PN)$.

For example, for the WF-net of Fig. 2 where the set of places $N = \{P4\}$ is a siphon, $\xi(p) = 1 \; if \; p \in N, else \; 0$ is a valuation satisfying $\theta(ex1)$.

The places (excluding places i and o) and transitions composing a correct execution of a WF-net cannot form a trap or a siphon. Using this propriety we refine the over-approximation made using $\varphi(PN, M_i, M_o)$. Theorem 4 states that for any solution of $\varphi(PN, M_a, M_b)$, the subnet, composed of places (excluding place i and o) and of the transitions of the modelled execution, contains a trap if and only if it has also a siphon. Therefore we only need to check the presence of a siphon (or, respectively, of a trap).

Theorem 4. *Let $PN = (P, T, F)$ a WF-net, M_a, M_b two markings of PN, and $\nu : P \times T \to \mathbb{N}$ a valuation satisfying $\varphi(PN, M_a, M_b)$. We define the subnet $sPN(\nu) = (sP, sT, sF)$ where:*
- $sP = \{p \in P \setminus \{i, o\} \mid \nu(p) > 0\}$
- $sT = \{t \in T \mid \nu(t) > 0\}$
- $sF = \{(a, b) \in F \mid a \in (sP \cup sT) \wedge b \in (sP \cup sT)\}$

If $sPN(\nu)$ contains a trap (resp. siphon) N then N is also a siphon (resp. trap).

Proof. ($\Rightarrow$) Let $N \subseteq sP$ such that $N \neq \emptyset$, so $\sum_{p\in N} \nu(p) = \sum_{p\in N} \sum_{t\in p^\bullet} \nu(t)$ $= \sum_{p\in N} \sum_{t\in {}^\bullet p} \nu(t)$. It implies $\sum_{p\in N} \sum_{t\in p^\bullet \cap N^\bullet} \nu(t) + \sum_{p\in N} \sum_{t\in p^\bullet \cap sT/N^\bullet} \nu(t)$ $= \sum_{p\in N} \sum_{t\in {}^\bullet p \cap N^\bullet} \nu(t) + \sum_{p\in N} \sum_{t\in {}^\bullet p \cap sT/N^\bullet} \nu(t)$ that can be simplified as $\sum_{p\in N} \sum_{t\in p^\bullet} \nu(t) = \sum_{p\in N} \sum_{t\in {}^\bullet p \cap N^\bullet} \nu(t) + \sum_{p\in N} \sum_{t\in {}^\bullet p \cap sT/N^\bullet} \nu(t)$ because $\forall p \in N. p^\bullet \cap sT/N^\bullet = \emptyset$. Let N be a trap ($N^\bullet \subseteq {}^\bullet N$) such that N is not a siphon (${}^\bullet N \not\subseteq N^\bullet$). Thus, one has $\sum_{p\in N} \sum_{t\in p^\bullet} \nu(t) = \sum_{p\in N} \sum_{t\in {}^\bullet p \cap N^\bullet} \nu(t)$ implying $\sum_{p\in N} \sum_{t\in p^\bullet} \nu(t) = \sum_{p\in N} \sum_{t\in p^\bullet} \nu(t) + \sum_{p\in N} \sum_{t\in {}^\bullet p \cap sT/N^\bullet} \nu(t)$. We finally have $\forall p \in N. {}^\bullet p \cap sT/N^\bullet = \emptyset$ because $\forall t \in sT. \nu(t) > 0$. This implies ${}^\bullet N \subseteq N^\bullet$, a contradiction.
($\Leftarrow$) The proof that if N is a siphon then N is a trap, is similar.

Theorem 5. *The Petri net $sPN(\nu)$ contains no siphon and no trap if and only if $\theta(sPN(\nu))$ does not have a valuation satisfying it.*

Proof. Follows from Th. 3 and 4.

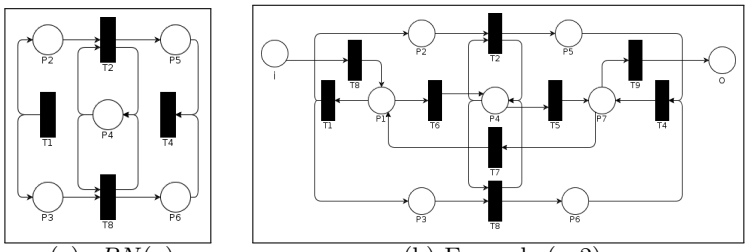
Using Th. 5 allows defining the constraint system in Th. 6, which refines $\varphi(PN, M_a, M_b)$. Thanks to this new system, the spurious solutions of $\varphi(PN, M_a, M_b)$ corresponding to an execution with siphon/trap are no more considered.

Theorem 6. *Let $PN = (P, T, F)$ be a WF-net and M_a, M_b two marking of PN. There exists $\nu : P \times T \to \mathbb{N}$ a valuation satisfying $\varphi(PN, M_a, M_b)$ such that $\theta(sPN(\nu))$ does not have a satisfying valuation if and only if there exist σ and $k \in \mathbb{N}$ such that $\forall p \in P \setminus \{i\}$. $M_{a'}(p) = M_a(p), M_{a'}(i) = k, \forall p \in P \setminus \{o\}. M_{b'}(p) = M_b(p), M_{b'}(o) = k, M_{a'} \xrightarrow{\sigma} M_{b'}$ and $\forall t \in T$. $O_t(\sigma) \geq \nu(t)$.*

Proof. ($\Rightarrow$) Suppose $\nu : P \times T \to \mathbb{N}$ is a valuation satisfying $\varphi(PN, M_a, M_b)$ such that $\theta(sPN(\nu))$ does not have a satisfying valuation. By Th. 5, $sPN(\nu)$ contains no siphon and therefore is live (cf. Th. 1). It implies that there is σ such that $M_a \xrightarrow{\sigma} M_b$ in $sPN(\nu)$ where $\forall t \in sT$. $O_t(\sigma) \geq \nu(t)$. Using the fact that a transition of σ is in $i^\bullet$, and a transition of σ is in ${}^\bullet o$, we can conclude that $M_{a'} \xrightarrow{\sigma} M_{b'}$ such that $\forall t \in T$. $O_t(\sigma) \geq \nu(t)$.

($\Leftarrow$) Suppose σ such that $M_a \xrightarrow{\sigma} M_b$ and $\forall t \in T.\ O_t(\sigma) = \nu(t)$. By Th. 2 we can complete the definition of ν to make ν a satisfying valuation of $\varphi(PN, M_a, M_b)$. In addition, $sPN(\nu)$ contains no siphon and no trap because it would contradict $M_a \xrightarrow{\sigma} M_b$. By Th. 5, $\theta(sPN(\nu))$ does not have a satisfying valuation.

For example, let us consider the WF-net of Fig. 2, the valuation $\nu(n) = 1$ if $n \in \{T1, T2, T8, T4, i, P2, P3, P5, P6, o\}$, $\nu(n) = 2$ if $n \in \{P4\}$, otherwise $\nu(n) = 0$, is a satisfying valuation of $\varphi(ex1, M_i, M_o)$. The set of places $N = \{P4\}$ is a trap/siphon. Figure 3(a) displays $sPN(\nu)$. Therefore by Th. 3 there is a valuation satisfying $\theta(sPN(\nu))$. By Th. 6, ν does not correspond to a correct execution of the WF-net of Fig. 2.



(a) $sPN(\nu)$ (b) Example (ex2)

Fig. 3. WF-net examples used to illustrate over-approximation

The constraint system of Th. 6 can be used to over-approximate the correct executions of a WF-net. Indeed, for the WF-net in Fig. 3(b), the valuation $\nu(n) = 1$ if $n \in \{T1, T8, T6, T2, T7, T8, T5, T9, T4, i, P2, P3, P5, P6, o\}$, $\nu(n) = 2$ if $n \in \{P1, P7\}$, and $\nu(n) = 3$ if $n \in \{P4\}$, is a valuation satisfying $\varphi(ex3, M_i, M_o)$ such that $\theta(sPN(\nu))$ does not have a satisfying valuation.
By Th. 6 there exist σ and k such that $M_{i(k)} \xrightarrow{\sigma} M_{o(k)}$ and $\forall t \in T.\ O_t(\sigma) \geq \nu(t)$. In this case there is no σ such that $k = 1$. Indeed, $P4$ cannot be empty when either $T2$ or $T8$ is fired, and therefore a marking with at least one token in $P4$ and one in either $P2$ or $P3$ must be reachable. As there is no execution possible with only one token that leads to such marking, we have $k > 1$.

While defining an over-approximation might be useful for the verification of safety property, in our case, we want to be able to verify a modal specification. As the approximation is difficult to handle, we need to be able to model an execution that violates the modal specification if it exists.

Theorem 7. *Let $PN = (P, T, F)$ be a WF-net, and M_a, M_b its two markings. If there is $\nu : P \times T \to \mathbb{N}$ such that $SAT(\varphi(PN, M_a, M_b), \nu) \wedge UNSAT(\theta(sPN(\nu)))$ $\wedge \forall n \in P \times T.\ \nu(n) \leq 1$ then $M_a \xrightarrow{\sigma} M_b$ and $\forall t \in T.\ O_t(\sigma) = \nu(t)$.*

Proof. Any place is involved with at most one transition consuming one token, and at most one transition producing one token. By Th. 6 one has $M_{a'} \xrightarrow{\sigma} M_{b'}$. Since at most one transition can consume a token in i (resp. produce a token in o), we have $M_{a'} = M_a$ (resp. $M_{b'} = M_b$).

In the rest of the paper, a segment of an execution is defined as an execution modelled by the constraint system in Th. 7. In this way, we now propose to decompose an execution modelled by the constraint system of Th. 6 into segment(s) modelled by Th. 7. If such a decomposition exists then the execution is a correct execution. Otherwise, we can conclude that the found solution is a spurious one. Indeed, spurious solutions can appear because the order of transition firing is not taken into account in the modelled execution. Therefore, decomposing the execution into segments forces the ordering of transitions where order matters.

Theorem 8. *Let $PN = (P, T, F)$ be a WF-net, and M_a, M_b its two markings. $M_a \xrightarrow{\sigma} M_b$ if and only if there exists $k \in \mathbb{N}$ such that $M_1 \xrightarrow{\sigma_1} M_2 \cdots M_k \xrightarrow{\sigma_{(k)}} M_{k+1}$, where $M_1 = M_a$, $M_{k+1} = M_b$ and for every i, $0 < i \leq k$, there is ν_i s.t. $SAT(\varphi(PN, M_i, M_{i+1}, \nu_i)) \wedge UNSAT(\theta(sPN(\nu_i))) \wedge \forall n \in P \times T.\ \nu_i(n) \leq 1$.*

Proof. ($\Rightarrow$) Suppose $M_a \xrightarrow{\sigma} M_b$ where $\sigma = t_1, \ldots, t_k$ then by definition there exist $M_1 \xrightarrow{t_1} M_2 \cdots M_k \xrightarrow{t_{(k)}} M_{k+1}$, where $M_1 = M_a$, $M_{k+1} = M_b$. Moreover, for every i, $0 < i \leq k$, there is ν_i such that $SAT(\varphi(PN, M_i, M_{i+1}, \nu_i)) \wedge UNSAT(\theta(sPN(\nu_i))) \wedge \forall n \in P \times T.\ \nu_i(n) \leq 1$, as ν_i is a valuation modelling the execution of a single transition.
($\Leftarrow$) Follows from Th. 7.

In the rest of the paper, we denote $\phi(PN, M_a, M_b, k)$ the constraint system of Th. 8, where k is the number of segments composing the execution. As $\phi(PN, M_i, M_o, k)$ can be used to model any execution of PN composed of k or less segments, we propose to use it to determine the validity of a WF-net with regards to a given modal specification.

4.3 Verifying Modal formulae

When determining whether or not a WF-net satisfies the modal properties of interest, we distinguish two decision problems. The first one, called the *K-bounded validity of a modal formula*, only considers executions formed by K segments, at most. The second one, called the *unbounded validity of a modal formula*, deals with executions formed by an arbitrary number of segments; it generalizes the first problem. To verify modalities over a single transition, constraint systems come very naturally into the play. Intuitively, for a *may*-transition t, determining one correct execution firing t at least once is enough to validate its *may*-specification. On the other hand, for a *must*-transition t, the lack of correct executions without firing it validates its *must*-specification.

In our approach, verifying modal specifications from Def. 4 relies on their expression by constraints. To build these constraints, for every transition $t \in T$, the corresponding terminal symbol of the formulae is replaced by $\nu(t) > 0$, where ν is the valuation of the constraint system. For example, for the modal formula $(T0 \wedge T5) \wedge (\neg T7 \vee T6)$, the corresponding constraint is $(\nu(T0) > 0 \wedge \nu(T5) > 0) \wedge (\neg\nu(T7) > 0 \vee \nu(T6) > 0)$. Given a modal formula $f \in S$, $C(f, \nu)$ denotes the constraint built from f, where ν is a the valuation of the constraint system. The following theorem extends the constraint systems to verify modal specifications.

Theorem 9. *Let $MPN = (P, T, F, m, M)$ a modal WF-net. The WF-net $PN = (P, T, F)$ satisfies the modal specification (m, M) if and only if:*

- *there is no $\nu, k \in \mathbb{N}$ such that $SAT(\phi(PN, M_i, M_o, k) \wedge \neg C(m, \nu), \nu)$, and*
- *for every $f \in M$, there exist $\nu, k \in \mathbb{N}$ such that $SAT(\phi(PN, M_i, M_o, k) \wedge C(f, \nu), \nu)$.*

Proof. By Th. 8, there exist $\nu, k \in \mathbb{N}$ such that $SAT(\phi(PN, M_i, M_o, k) \wedge \neg C(m, \nu), \nu)$ if and only if $PN \not\models_{must} m$. In addition, there are $\nu, k \in \mathbb{N}$ such that $SAT(\phi(PN, M_i, M_o, k) \wedge C(f, \nu), \nu)$ if and only if $PN \models_{may} f$.

Theorem 9 can be adapted to the case of hierarchical WF-nets. In this case, the modal formula has to be verified for the main WF-net, i.e. the highest level net, and also for the WF-nets substituting transitions at lower levels.

Theorem 10. *Let $PN = (P, T, F)$ be a WF-net, $\bar{R}_{must}$ the set of all well-formed must-formulae not satisfied by PN, and R_{may} the set of all well-formed may-formulae satisfied by PN. There exists K_{max} such that:*

- $\forall f \in \bar{R}_{must}, \exists\ \nu, k \leq K_{max}.\ SAT(\phi(PN, M_i, M_o, k) \wedge \neg C(f, \nu), \nu)$,
- $\forall f \in R_{may}, \exists\ \nu, k \leq K_{max}.\ SAT(\phi(PN, M_i, M_o, k) \wedge C(f, \nu), \nu)$.

Proof. Sketch. The set of correct executions of a WF-net is possibly infinite. This is due to the fact that T-invariants (i.e. sequence of transitions σ such that $M \xrightarrow{\sigma} M$) could be fired indefinitely. However, when considering the verification of modal formulae, we are only interested in the presence or absence of transitions in correct executions (i.e. the number of their firings does not matter). Therefore considering the set of correct executions where T-invariants are allowed to fire at most once is enough to check the validity of modal formulae. This restricted set of correct executions is finite. As a consequence, there exists K_{max} such that any execution of this set can be modelled by K_{max} segments, at most.

Theorem 10 implies that for any WF-net $PN = (P, T, F)$, there exists K_{max} such that any modal *may*-formula (resp. *must*-formula) f can be verified regarding the consistency of the constraint system $\phi(PN, M_i, M_o, K_{max}) \wedge C(f, \nu)$ (resp. $\phi(PN, M_i, M_o, K_{max}) \wedge \neg C(f, \nu)$). In other words, to verify any *may*-formula (resp. *must*-formula), it is not necessary to look for the existence (resp. non-existence) of correct execution respecting (resp. not respecting) the behaviour expressed by the *may*-formula (resp. *must*-formula) of this WF-net composed of more than K_{max} segments. However determining the K_{max} value of a WF-net from its structure is still an open problem. However, we can infer an upper-bound of $\sum_{j=1}^{|T|} j!$.

5 Implementation and Experiments

The proposed approach has been fully automated, allowing practitioners, at any stage of the workflow design, to verify modal formulae using an integrated tool chain. This section describes this tool chain developed to experimentally validate the proposed approach, and illustrates its use and obtained results on the case study introduced in Sect. 2.

5.1 Implementation Architecture

As a proof of concept, an implementation supporting the approach we propose has been developed to provide an integrated tool chain to design WF-nets and verify modal specifications. The architecture is shown in Fig. 4.

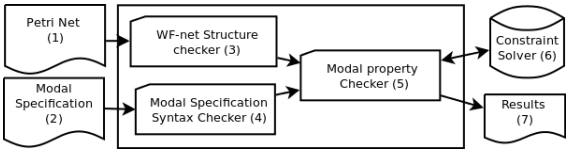


Fig. 4. Tool chain description

The tool chain takes as inputs a WF-net model (1) and the modal specifications (2) to be verified. WF-net model is exported from a third party software (e.g., Yasper [10], PIPE [11]) as an XML file, as well as the modal specifications that are expressed in a dedicated and proprietary XML format. From these inputs, the developed tool first checks the structure of the WF-net model (3) to exclude Petri nets that do not correspond to WF-nets definition (cf. Def. 2). It then checks the modal specifications regarding the syntax proposed in Sect. 4 (4). Once validated, these inputs are translated into a constraint system (5) that is handled using the CLP(FD) library of Sicstus Prolog [12] (6). Finally, a report about the validity of modal specifications of the WF-net is generated (7).

To verify a *may*-formula (resp. a *must*-formula) m (resp. M), the tool first checks if there exists a solution of the over-approximation, given by Th. 6, such that the modelled execution satisfies (resp. does not satisfy) m (resp. M). If such an execution exists, it then tries to find an execution of the under-approximation, given by Th. 8, which satisfies (resp. does not satisfy) m. As an illustration, Figure 5 gives the algorithm of the function checking the validity of a *must*-formula. It returns the K-bounded validity of a given modal formula m. To cope with the complexity raised by K_{max}, K can be fixed to a manageable value. Nevertheless, when fixing K to K_{max} (or greater than K_{max}), the algorithm enables to decide the unbounded validity of the *must*-formula m. The results in Sect. 4.1 ensure its soundness and completeness. Finally, solving a CSP over a finite domain being an NP-complete problem with respect to the domain size, this algorithm inherits this complexity.

5.2 Experimental Results

The approach and the corresponding implementation have been firstly validated on a set of models collected from the literature, especially from [4, 13, 14], and afterwards experimented in the field of issue tracking systems using the industrial example described in Sect. 2. Table 1 shows an extract of the experimental results obtained on this industrial example, focusing on the six properties (p_1 to p_6) and the WF-net model introduced in Sect. 2.

Inputs: PN - a WF-net, m - a *must*-formula, K a positive integer.
Results: $TRUE$ - $PN \models_{must} m$, $FALSE$ - $PN \not\models_{must} m$.
function IsMustValid(PN,m,K)
  if $SAT(\varphi(PN, M_i, M_o) \wedge \neg C(m, \nu), \nu) \wedge UNSAT(\theta(sPN(\nu)))$ **then**
    $k = max(\{v(n) | n \in T\})$
    if $k == 1$ **then return** $FALSE$
    else
      while $k \leq K$ **do**
        if $SAT(\phi(PN, M_i, M_o, k) \wedge \neg C(m, v), \nu)$ **then return** $FALSE$
        else $k = k + 1$
        end if
      end while
      return $TRUE$
    end if
  else return $TRUE$
  end if
end function

Fig. 5. Algorithm checking the validity of a *must*-formula

The properties p_1 to p_6 are representative of the kind of properties that have to be verified by engineers when they design the business process to be implemented. Moreover, These properties are sufficiently clear without a complete description of the workflow and enable to show all possible outcomes of our approach. The modal formula associated with each property is specified, and the result of the computation is given by its final result as well as the internal evaluation of φ. The input K and the corresponding computed value of $\phi(K)$ are also precised when it makes sense, i.e. when the algorithm cannot conclude without this bound.
When verifying *must*-formulae that are satisfied by the WF-net (see p_1, p_2 and p_3), or *may*-formulae that are not satisfied by the WF-net (see p_4), the over-approximation proposed in Th. 6 is usually enough to conclude. On the other hand, when verifying *may*-formulae that are satisfied by the WF-net (see p_5), or *must*-formulae that are not satisfied by the WF-net (see p_6), the decomposition into K segments is needed. We empirically demonstrate that this decomposition is very effective since values of K_{max} are usually moderate ($K_{max} = 6$ in the case of p_5, less than 10 with all the experimentations on this case-study). We can also notice the definitive invalidity of p_6 (a user can exit the current session without logout), which enabled to highlight an ambiguity in the textual requirements.

Thanks to the experiments, we can conclude that the proposed method is feasible and efficient. Moreover, the developed tool is able to conclude about the (in)validity of the studied properties in a very short time (less than a second).

#	Formula	φ	K	$\phi(K)$	Result
p_1	$PN \models_{must} (SubA \wedge \neg SubA) \vee (SubB \wedge \neg SubA)$	TRUE	-	-	TRUE
p_2	$PN \models_{must} SubB \Rightarrow Login$	TRUE	-	-	TRUE
p_3	$PN \models_{must} SR\_CreateCRITSIT \Rightarrow (VandD \vee Update \vee Closure)$	TRUE	-	-	TRUE
p_4	$PN \models_{may} SR\_CreateCRITSIT \Rightarrow (Update \wedge Closure)$	FALSE	-	-	FALSE
p_5	$PN \models_{may} SR\_UpgradeToCRITSIT$	TRUE	1	FALSE	FALSE
			6	TRUE	TRUE
p_6	$PN \models_{must} Login \Rightarrow Logout$	FALSE	1	FALSE	FALSE

Table 1. Experimentation results

6 Conclusion and Related Work

Modal specifications introduced in [15] allow loose or partial specifications in a process algebraic framework. Since, modal specifications have been ported to Petri nets, as in [16]. In this work, a relation between generated modal languages is used for deciding specifications' refinement and asynchronous composition. Instead of comparing modal languages, our approach deals with the correct executions of WF-nets modelled by constraint systems. A lot of work has been done [17–19] in order to model and to analyse the behaviour of Petri nets by using equational approaches. Among popular resolution techniques, the constraint programming framework has been successfully used to analyse properties of Petri net [20, 21]. But, like in [21], the state equation together with a trap equation are used in order to verify properties such as deadlock-freedom. Our approach also takes advantage of trap and siphon properties in pursuance of modelling correct executions. Constraint programming has also been used to tackle the reachability problem—one of central verification problems. Let us quote [22] where a decomposition into *step sequences* was modelled by a constraint system. Our approach is similar, the main difference is that the constraints we propose on step sequences, i.e. segments, are stronger. This is due to the fact that we are not only interested in the reachability of a marking, but also in the transitions involved in the sequences of transitions that reach it.

This paper hence presents an original and innovative formal framework based on constraint systems to model executions of WF-nets and their structural properties, as well as to verify their modal specifications. It also reports on encouraging experimental results obtained using a proof-of-concept tool chain. In particular, a business process example from the IT domain enables to successfully assess the reliability of our contributions. As a future work, we plan extensive experimentation to determine and improve the scalability of our verification approach based on constraint systems. We also need to improve its readiness level in order to foster its use by business analysts. For instance, we could propose a user-friendly patterns to express the modal properties. Finally, generalizing our approach by handling coloured Petri nets is another research direction.

Acknowledgment

This project is performed in cooperation with the Labex ACTION program (contract ANR-11-LABX-0001-01) – see `http://www.labex-action.fr/en`.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Journal of Information Systems **30**(4) (June 2005) 245–275
2. Dumas, M., Hofstede, A.H.M.t.: UML Activity Diagrams As a Workflow Specification Language. In: Proc. of the 4^{th} Int. Conf. on The Unified Modeling Language (UML'01), Toronto, Canada, Springer-Verlag (October 2001) 76–90

3. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems, and Computers **8**(1) (February 1998) 21–66
4. van der Aalst, W.M.P.: Three Good reasons for Using a Petri-net-based Workflow Management System. Journal of Information and Process Integration in Enterprises **428** (December 1997) 161–182
5. Larsen, K.G.: Modal Specifications. In: Proc. of the Int. Workshop on Automatic Verification Methods for Finite State Systems. Volume 407 of LNCS., Grenoble, France, Springer-Verlag (June 1989) 232–246
6. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Proc. of the 18^{th} Int. Conf. on Application and Theory of Petri Nets (ICATPN'97). Volume 1248 of LNCS., Toulouse, France, Springer (June 1997) 407–426
7. Heiner, M., Gilbert, D., Donaldson, R.: Petri Nets for Systems and Synthetic Biology. In: Proc. of the 8^{th} Int. School on Formal Methods for Computational Systems Biology (SFM'08). Volume 5016 of LNCS., Springer (June 2008) 215–264
8. Macworth, A.K.: Consistency in networks of relations. Journal of Artificial Intelligence **8**(1) (1977) 99–118
9. Tsang, E.: Foundation of constraint satisfaction. Academic Press (1993)
10. van Hee, K., et al.: Yasper: a tool for workflow modeling and analysis. In: Proc. of the 6^{th} Int. Conf. on Application of Concurrency to System Design (ACSD'06), Turku, Finland, IEEE CS (June 2006) 279–282
11. Bonet, P., Lladó, C.M., Puijaner, R., Knottenbelt, W.J.: PIPE v2.5: A Petri net tool for performance modelling. In: Proc. of the 23^{rd} Latin American Conference on Informatics (CLEI'07), San Jose, Costa Rica (October 2007)
12. Carlsson, M., et al.: SICStus Prolog user's manual (Release 4.2.3), Swedish Institute of Computer Science, Kista, Sweden. (October 2012)
13. Kouchnarenko, O., Sidorova, N., Trcka, N.: Petri Nets with May/Must Semantics. In: Proc. of the Workshop on Concurrency, Specification, and Programming (CS&P'09), Kraków-Przegorzaly, Poland (September 2009) 291–302
14. van der Aalst, W.M.P.: Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Managemen. In: Lectures on Concurrency and Petri Nets. Volume 3098 of LNCS., Springer (2004) 1–65
15. Larsen, K.G., Thomsen, B.: A modal process logic. In: Proc. of the 3^{rd} Annual Symp. on Logic in Computer Science (LICS'88), IEEE (July 1988) 203–210
16. Elhog-Benzina, D., Haddad, S., Hennicker, R.: Refinement and asynchronous composition of modal petri nets. In: Transactions on Petri Nets and Other Models of Concurrency V. Volume 6900 of LNCS. Springer (2012) 96–120
17. Desel, J.: Basic linear algebraic techniques for place/transition nets. In: Lectures on Petri Nets I: Basic Models. Volume 1491 of LNCS. Springer (1998) 257–308
18. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri Net State Equation. In: Proc. of the 17^{th} Int. Conf. on Tools and Alg. for the Construction and Analysis of Systems (TACAS'11). Volume 6605 of LNCS., Springer (March 2011) 224–238
19. Schmidt, K.: Narrowing Petri Net State Spaces Using the State Equation. Fundamenta Informaticae **47**(3-4) (October 2001) 325–335
20. Soliman, S.: Finding minimal P/T-invariants as a CSP. In: Proc. of the 4^{th} Workshop on Constraint Based Methods for Bioinformatics (WCB'08). (May 2008)
21. Melzer, S., Esparza, J.: Checking system properties via integer programming. In: Proc. of the 6^{th} Eur. Symp. on Programming Languages and Systems (ESOP'96). Volume 1058 of LNCS., Linköping, Sweden, Springer (April 1996) 250–264
22. Bourdeaud'huy, T., Hanafi, S., Yim, P.: Incremental Integer Linear Programming Models for Petri Nets Reachability Problems. Petri Net: Theory and Applications (February 2008) 401–434