Recent Advances in Petri Nets and Concurrency

Susanna Donatelli, Jetty Kleijn,
Ricardo J. Machado, João M. Fernandes (eds.)

CEUR Workshop Proceedings, vol. 827
urn:nbn:de:0074-827-8
ISSN 1613-0073

January 2012

Contents

Preface

These proceedings, part of the CEUR series, contain contributions accepted for and presented at the workshops organized as satellite events to the "31st International Conference on Application and Theory of Petri Nets and Other Models of Concurrency" (PETRI NETS 2010) and the "10th International Conference on Application of Concurrency to System Design" (ACSD 2010), in Braga, Portugal, on June 21-22, 2010.

The five workshops were the following:

- Biological Processes & Petri Nets (BioPPN)
  organized by Claudine Chaouiya and Monika Heiner
- Applications of Region Theory (ART)
  organized by Jörg Desel and Alex Yakovlev
- Petri Nets and Software Engineering (PNSE)
  organized by Michael Duvigneau and Daniel Moldt
- Abstractions for Petri Nets and Other models of Concurrency (APNOC)
  organized by Alexander Serebrenik and Natalia Sidorova,
- Scalable and Usable Model checking for Petri nets and other models of concurrency (SUMo)
  organized by Didier Buchs, Fabrice Kordon, Yann Thierry-Mieg, and Jeremy Sproston

All these workshops have been organised as a discussion platform for researchers interested in the application of Petri nets and other formal concurrency models in many current fields of study, namely in biology, synthesis, software engineering, and model checking. Independent reviewers have carefully reviewed all papers.

For more details, please see the PETRI NETS / ACSD 2010 website:
http://acsd-petrinets2010.di.uminho.pt

January 2012

Susanna Donatelli
Jetty Kleijn
Ricardo J. Machado
João M. Fernandes

Biological Processes & Petri Nets (BioPPN)

Introduction

This chapter contains the nine peer-reviewed contributions of the First International Workshop on Biological Processes & Petri Nets (BioPPN 2010), held as a satellite event of PETRI NETS 2010, in Braga, Portugal, at June 21, 2010. This workshop has been organised as a communication platform for researchers interested in the application of Petri nets in the broad field of integrative biology.

Integrative biology aims at deciphering essential biological processes that are driven by complex mechanisms, involving miscellaneous interacting molecular compounds. In this context, the need for appropriate mathematical and computational modelling tools is widely advocated. Petri nets have proved their usefulness for the modelling, analysis, and simulation of a diversity of biological networks, covering qualitative, stochastic, continuous and hybrid models. The deployment of Petri nets to study biological applications has not only generated original models, but has also motivated fundamental research.

We received two types of contributions: research papers and work-in-progress papers. All have been reviewed by four to five referees coming from or being recommended by the workshop's Program Committee. In summary, the workshop proceedings enclose theoretical contributions as well as biological applications, demonstrating the interdisciplinary nature of the topic.

The workshop was complemented by an invited talk *Why aren't Petri nets widely used in biological research?* given by Jorge Carneiro from Instituto Gulbenkian de Ciência (IGC, Oeiras, Portugal). He argued that software tools for stochastic Petri nets are well-suited for engineering artificial systems, but do not yet offer all the functionalities one would wish to have at hand when modelling a natural biological system. He used two application examples of stochastic Petri nets to illustrate his concerns – modelling somatic recombination of immune receptor genes and ion channel gating in sea urchin spermatozoa.

The workshop gathered about 30 researchers actively working on or merely interested in the application of Petri nets to biological processes. Its main goal was to demonstrate that this field of application raises new challenges and that Petri nets can be highly effective to tackle such challenges. We take the lively discussion throughout the whole day of workshop as proof that this goal had been reached. For more details see the workshop website `http://www-dssz.informatik.tu-cottbus.de/BME/BioPPN2010`.

Claudine Chaouiya
Instituto Gulbenkian de Ciência, IGC
Oeiras, Portugal
chaouiya@igc.gulbenkian.pt

Monika Heiner
Brandenburg University of Technology at Cottbus
Computer Science Institute, Germany
monika.heiner@informatik.tu-cottbus.de

Program Committee

- David Angeli, I, UK
- Gianfranco Balbo, I
- Rainer Breitling, NL
- David Gilbert, UK
- Simon Hardy, CA
- Ralf Hofestädt, D
- Peter Kemper, US
- Hanna Klaudel, F
- Kurt Lautenbach, D
- Wolfgang Marwan, D
- Eduardo Mendoza, Philippines, D
- Satoru Miyano, JP
- P.S. Thiagarajan, SG

Cycle structure in SR and DSR graphs: implications for multiple equilibria and stable oscillation in chemical reaction networks

Murad Banaji

Department of Mathematics, University of Portsmouth, Lion Gate Building, Lion Terrace, Portsmouth, Hampshire PO1 3HF, UK.

Abstract. Associated with a chemical reaction network is a natural labelled bipartite multigraph termed an SR graph, and its directed version, the DSR graph. These objects are closely related to Petri nets. The construction of SR and DSR graphs for chemical reaction networks is presented. Conclusions about asymptotic behaviour of the associated dynamical systems which can be drawn easily from the graphs are discussed. In particular, theorems on ruling out the possibility of multiple equilibria or stable oscillation in chemical reaction networks based on computations on SR/DSR graphs are presented. These include both published and new results. The power and limitations of such results are illustrated via several examples.

1 Chemical reaction networks: structure and kinetics

Models of chemical reaction networks (CRNs) are able to display a rich variety of dynamical behaviours [1]. In this paper, a spatially homogeneous setting is assumed, so that CRNs involving n chemicals give rise to local semiflows on $\mathbb{R}^n_{\geq 0}$, the nonnegative orthant in $\mathbb{R}^n$. These local semiflows are fully determined if we know 1) the CRN *structure*, that is, which chemicals react with each other and in what proportions, and 2) the CRN *kinetics*, that is, how the reaction rates depend on the chemical concentrations. An important question is what CRN behaviours are determined primarily by reaction network structure, with limited assumptions about the kinetics.

A variety of representations of CRN structure are possible, for example via matrices or generalised graphs. Of these, a signed, labelled, bipartite multigraph, termed an *SR graph*, and its directed version, the *DSR graph*, are formally similar to Petri nets. This relationship is discussed further below.

It is now well established that graphical representations can tell us a great deal about asymptotic behaviours in the associated dynamical systems. Pioneering early work on CRNs with mass-action kinetics ([2, 3] for example), had a graph-theoretic component (using graphs somewhat different from those to be presented here). More recently, graph-theoretic approaches have been used to draw conclusions about multistationarity and oscillation in CRNs with restricted classes of kinetics [4, 5].

The applicability of such work, particularly in biological contexts, is greatly increased if only weak assumptions are made about kinetics. Consequently, there is a growing body of recent work on CRNs with essentially arbitrary kinetics. It has been shown that examination of Petri nets associated with a CRN allows conclusions about persistence, that is, whether ω-limit sets of interior points of $\mathbb{R}^n_{\geq 0}$ can intersect the boundary of $\mathbb{R}^n_{\geq 0}$ [6]. Work on multistationarity has been extended beyond the mass-action setting [7, 8]: some conclusions of this work will be outlined below. Finally, recent work applying the theory of monotone dynamical systems [9, 10] in innovative ways to CRNs [11] has close links with some of the new material presented below.

Outline. After some preliminaries, the construction of SR and DSR graphs is presented, and their relationship to Petri nets is discussed. Some recent results about multistationarity based on cycle structure in these objects are described. Subsequently, a new result on monotonicity in CRNs is proved. This result, Proposition 4, is a graph-theoretic corollary of results in [12]. It bears an interesting relationship to results in [11], which provide stronger conclusions about convergence, but make different assumptions, and a somewhat different claim. Finally, several examples, some raising interesting open questions, are presented. At various points, in order to simplify the exposition, the results are presented in less generality than possible, with more technical results being referenced.

2 Preliminaries

2.1 A motivating example

Consider the following simple family of CRNs treated in [13, 14]:

$$\begin{array}{llll} \textbf{SYS } 1 & \textbf{SYS } 2 & \cdots & \textbf{SYS } n \\ \boxed{\begin{array}{c} A_1 + A_2 \rightleftharpoons B_1 \\ A_2 + A_3 \rightleftharpoons B_2 \\ A_3 \rightleftharpoons 2A_1 \end{array}} & \boxed{\begin{array}{c} A_1 + A_2 \rightleftharpoons B_1 \\ A_2 + A_3 \rightleftharpoons B_2 \\ A_3 + A_4 \rightleftharpoons B_3 \\ A_4 \rightleftharpoons 2A_1 \end{array}} & \cdots & \boxed{\begin{array}{c} A_i + A_{i+1} \rightleftharpoons B_i, \\ \quad i = 1, \ldots, n+1 \\ A_{n+2} \rightleftharpoons 2A_1 \end{array}} \end{array} \tag{1}$$

The reader may wish to look ahead to Figure 2 to see representations of the SR graphs associated with the first three CRNs in this family. This family will be revisited in Section 7, and the theory to be presented will imply the following conclusions (to be made precise below): when n is even, **SYS** n does not allow multiple nondegenerate equilibria; when n is odd, **SYS** n cannot have a nontrivial periodic attractor. Both conclusions require only minimal assumptions about the kinetics.

2.2 Dynamical systems associated with CRNs

In a spatially homogeneous setting, a chemical reaction system in which n reactants participate in m reactions has dynamics governed by the ordinary differential equation

$$\dot{x} = \Gamma v(x). \tag{2}$$

$x = [x_1, \ldots, x_n]^T$ is the nonnegative vector of reactant concentrations, and $v = [v_1, \ldots, v_m]^T$ is the vector of reaction rates, assumed to be C^1. A reaction rate is the rate at which a reaction proceeds *to the right* and may take any real value. Γ is the (constant) $n \times m$ **stoichiometric matrix** of the reaction system. Since reactant concentrations cannot be negative, it is always reasonable to assume invariance of $\mathbb{R}^n_{\geq 0}$, i.e. $x_i = 0 \Rightarrow \dot{x}_i \geq 0$.

The jth column of Γ, termed Γ_j, is the **reaction vector** for the jth reaction, and a stoichiometric matrix is defined only up to an arbitrary signing of its columns. In other words, given any $m \times m$ signature matrix D (i.e. any diagonal matrix with diagonal entries ± 1), one could replace Γ with ΓD and $v(x)$ with $Dv(x)$. Obviously the dynamical system is left unchanged. The subspace $\mathrm{Im}(\Gamma)$ of $\mathbb{R}^n$ spanned by the reaction vectors is called the **stoichiometric subspace**. The intersection of any coset of the $\mathrm{Im}(\Gamma)$ with $\mathbb{R}^n_{\geq 0}$ is called a **stoichiometry class**.

Two generalisations of (2) which include explicit inflow and outflow of substrates are worth considering. The first of these is a so-called CFSTR

$$\dot{x} = q(x_{in} - x) + \Gamma v(x). \tag{3}$$

$q \in \mathbb{R}$, the flow rate, is generally assumed to be positive, but we allow $q = 0$ so that (2) becomes a special case of (3). $x_{in} \in \mathbb{R}^n$ is a constant nonnegative vector representing the "feed" (i.e., inflow) concentrations. The second class of systems is:

$$\dot{x} = x_{in} + \Gamma v(x) - Q(x). \tag{4}$$

Here $Q(x) = [q_1(x_1), \ldots, q_n(x_n)]^T$, with each $q_i(x_i)$ assumed to be a C^1 function satisfying $\frac{\partial q_i}{\partial x_i} > 0$, and all other quantities defined as before. Systems (4) include systems (3) with $q \neq 0$, while systems (2) lie in the closure of systems (4).

Define the $m \times n$ matrix $V = [V_{ji}]$ where $V_{ji} = \frac{\partial v_j}{\partial x_i}$. A very reasonable, but weak, assumption about many reaction systems is that reaction rates are monotonic functions of substrate concentrations as assumed in [14–16] amongst other places. We use the following definition from [14] (there called NAC):

A reaction system is **N1C** if i) $\Gamma_{ij} V_{ji} \leq 0$ for all i and j, and ii) $\Gamma_{ij} = 0 \Rightarrow V_{ji} = 0$.

As discussed in [14], the relationship between signs of entries in Γ and V encoded in the N1C criterion is fulfilled by all reasonable reaction kinetics (including mass action and Michaelis-Menten kinetics for example), provided that reactants never occur on both sides of a reaction.

3 Introduction to SR and DSR graphs

3.1 Construction and relation to Petri nets

SR graphs are signed, bipartite multigraphs with two vertex sets V_S (termed "S-vertices") and V_R (termed "R-vertices"). The edges E form a multiset, consisting of unordered pairs of vertices, one from V_S and one from V_R. Each edge is signed and labelled either with a positive real number or the formal label ∞. In other words, there are functions $\mathrm{sgn} : E \to \{-1, 1\}$, and $\mathrm{lbl} : E \to (0, \infty) \cup \{\infty\}$. The quintuple $(V_S, V_R, E, \mathrm{sgn}, \mathrm{lbl})$ defines an SR graph.

DSR graphs are similar, but have an additional "orientation function" on their edges, $\mathcal{O} : E \to \{-1, 0, 1\}$. The sextuple $(V_S, V_R, E, \mathrm{sgn}, \mathrm{lbl}, \mathcal{O})$ defines a DSR graph. If $\mathcal{O}(e) = -1$ we will say that the edge e has "S-to-R direction", if $\mathcal{O}(e) = 1$, then e has "R-to-S direction", and if $\mathcal{O}(e) = 0$, then e is "undirected". An undirected edge can be regarded as an edge with both S-to-R and R-to-S direction, and indeed, several results below are unchanged if an undirected edge is treated as a pair of antiparallel edges of the same sign. SR graphs can be regarded as the subset of DSR graphs where all edges are undirected.

Both the underlying meanings, and the formal structures, of Petri nets and SR/DSR graphs have some similarity. If we replace each undirected edge in a DSR graph with a pair of antiparallel edges, a DSR graph is simply a Petri net graph, i.e. a bipartite, multidigraph. Similarly, an SR graph is a bipartite multigraph. S-vertices correspond to variables, while R-vertices correspond to processes which govern their interaction. The notions of variable and process are similar to the notions of "place" and "transition" for a Petri net. Edges in SR/DSR graphs tell us which variables participate in each process, with additional qualitative information on the nature of this participation in the form of signs, labels, and directions; edges in Petri nets inform on which objects are changed by a transition, again with additional information in the form of labels (multiplicities) and directions. Thus both Petri net graphs and SR/DSR graphs encode partial information about associated dynamical systems, while neither includes an explicit notion of time.

There are some important differences, however. Where SR/DSR graphs generally represent the structures of continuous-state, continuous-time dynamical systems, Petri nets most often correspond to discrete-state, discrete-time systems, although the translation to a continuous-state and continuous-time context is possible [17]. Although in both cases additional structures give partial information about these dynamical systems, there are differences of meaning and emphasis. Signs on edges in a DSR graph, crucial to much of the associated theory, are analogous to directions on edges in a Petri net: for example for an irreversible chemical reaction, an arc from a substrate to reaction vertex in the Petri net would correspond to a negative, undirected, edge in the SR/DSR graph. Unlike SR/DSR graphs, markings (i.e. vertex-labellings representing the current state) are often considered an intrinsic component of Petri nets.

Apart from formal variations between Petri nets and SR/DSR graphs, differences in the notions of state and time lead naturally to differences in the questions asked. Most current work using SR/DSR graphs aims to inform on the existence, nature, and stability of limit sets of the associated local semiflows. Analogous questions are certainly possible with Petri nets, for example questions about the existence of stationary probability distributions for stochastic Petri nets [18]. However, much study, for example about reachability, safeness and boundedness, concerns the structure of the state space itself, and has no obvious analogy in the SR/DSR case. This explains to some extent the importance of markings in the study of Petri nets; in the case of SR/DSR graphs, the underlying space is generally assumed to have a simple structure, and the aim is to draw conclusions which are largely independent of initial conditions.

3.2 SR and DSR graphs associated with CRNs

SR and DSR graphs can be associated with arbitrary CRNs and more general dynamical systems [7, 8]. For example, the construction extends to situations where there are modulators of reactions which do not themselves participate in reactions, and where substrates occur on both sides of a reaction. Here, for simplicity, the construction is presented for an N1C reaction system with stoichiometric matrix $\mathit{\Gamma}$. Assume that there is a set of substrates $V_S = \{S_1, \ldots, S_n\}$, having concentrations $x_1, \ldots, x_n$, and reactions $V_R = \{R_1, \ldots, R_m\}$ occurring at rates $v_1, \ldots, v_m$. The labels in V_S and V_R will be used to refer both to the substrate/reaction, and the associated substrate/reaction vertices.

– If $\mathit{\Gamma}_{ij} \neq 0$ (i.e. there is net production or consumption of S_i reaction j), and also $\frac{\partial v_j}{\partial x_i}$ is not identically zero, i.e. the concentration of substrate i affects the rate of reaction j, then there is an undirected edge $\{S_i, R_j\}$.
– If $\mathit{\Gamma}_{ij} \neq 0$, but $\frac{\partial v_j}{\partial x_i} \equiv 0$, then the edge $\{S_i, R_j\}$ has only R-to-S direction.

The edge $\{S_i, R_j\}$ has the sign of $\mathit{\Gamma}_{ij}$ and label $|\mathit{\Gamma}_{ij}|$. Thus the labels on edges are just stoichiometries, while the signs on edges encode information on which substrates occur together on each side of a reaction. A more complete discussion of the meanings of edge-signs in terms of "activation" and "inhibition" is presented in [8]. Note that in the context of N1C reaction systems, the following features (which are reasonably common in the more general setting) do not occur: edges with only R-to-S direction; multiple edges between a vertex pair; and edges with edge-label ∞.

SR/DSR graphs can be uniquely associated with (2), (3), or (4): in the case of (3) and (4), the inflows and outflows are ignored, and the SR/DSR graph is just that derived from the associated system (2). The construction is most easily visualised via an example. Consider, first, the simple system of two reactions:

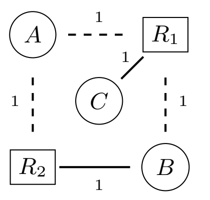
$$A + B \rightleftharpoons C, \qquad A \rightleftharpoons B \tag{5}$$

This has SR graph, shown in Figure 1, *left*. If all substrates affect the rates of reactions in which they participate then this is also the DSR graph for the reaction. If, now, the second reaction is irreversible, i.e. one can write

$$A + B \rightleftharpoons C, \qquad A \to B, \tag{6}$$

and consequently the concentration of B does not affect the rate of the second reaction[1], then the SR graph remains the same, losing information about irreversibility, but the DSR graph now appears as in Figure 1 *right*.

[1] Note that this is usually, but not always, implied by irreversibility: it is possible for the product of an irreversible reaction to influence a reaction rate.



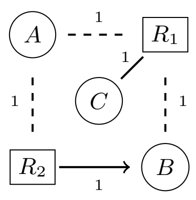


Fig. 1. *Left.* The SR (and DSR graph) for reaction system (5). Negative edges are depicted as dashed lines, while positive edges are bold lines. This convention will be followed throughout. *Right.* The DSR graph for reaction system (6), that is, when B is assumed not to affect the rate of the second reaction.

4 Paths and cycles in SR and DSR graphs

In the usual way, **cycles** in SR (DSR) graphs are minimal undirected (directed) paths from some vertex to itself. All paths have a sign, defined as the product of signs of edges in the path. Given any subgraph E, its size (or length, if it is a path) $|E|$ is the number of edges in E. Paths of length two will be called **short** paths. Any path E of even length also has a **parity**

$$P(E) = (-1)^{|E|/2}\mathrm{sign}(E).$$

A cycle C is an **e-cycle** if $P(C) = 1$, and an **o-cycle** otherwise. Given a cycle C containing edges $e_1, e_2, \ldots, e_{2r}$ such that e_i and $e_{(i \bmod 2r)+1}$ are adjacent for each $i = 1, \ldots, 2r$, define:

$$\mathrm{stoich}(C) = \left|\prod_{i=1}^{r} \mathrm{lbl}(e_{2i-1}) - \prod_{i=1}^{r} \mathrm{lbl}(e_{2i})\right| .$$

Note that this definition is independent of the starting point chosen on the cycle. A cycle with $\mathrm{stoich}(C) = 0$ is termed an **s-cycle**.

An **S-to-R path** in an SR graph is a non-self-intersecting path between an S-vertex and an R-vertex. **R-to-R paths** and **S-to-S paths** are similarly defined, though in these cases the initial and terminal vertices may coincide. Any cycle is both an R-to-R path and an S-to-S path. Two cycles have **S-to-R intersection** if each component of their intersection is an S-to-R path. This definition can be generalised to DSR graphs in a natural way, but to avoid technicalities regarding cycle orientation, the reader is referred to [8] for the details. Further notation will be presented as needed.

Returning to the family of CRNs in (1), these give SR graphs shown in Figure 2. If all reactants can influence the rates of reactions in which they participate, then these are also their DSR graphs (otherwise some edges may become directed). Each SR graph contains a single cycle, which is an e-cycle (resp. o-cycle) if n is odd (resp. even). These cycles all fail to be s-cycles because of the unique edge-label of 2.

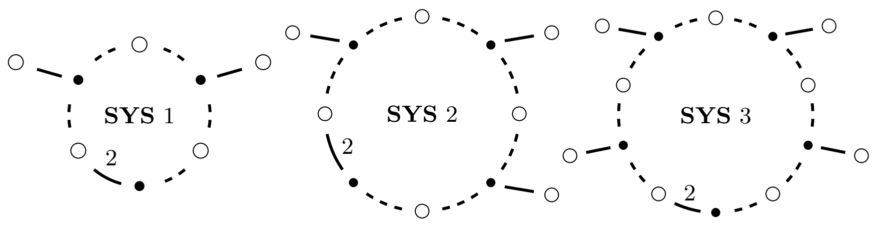


Fig. 2. The structure of the SR graphs for **SYS** 1, 2 and 3 in (1). For simplicity vertices are unlabelled, but filled circles are S-vertices while open circles are R-vertices. Unlabelled edges have edge-label 1.

5 Existing results on CRNs, injectivity and monotonicity

5.1 Injectivity and multiple equilibria

A function $f : X \to \mathbb{R}^n$ is **injective** if for any $x, y \in X$, $f(x) = f(y)$ implies $x = y$. Injectivity of a vector field on some domain is sufficient to guarantee that there can be no more than one equilibrium on this domain. Define the following easily computable condition on an SR or DSR graph:

Condition (∗): All e-cycles are s-cycles, and no two e-cycles have S-to-R intersection.

Note that if an SR/DSR graph has no e-cycles, then Condition (∗) is trivially fulfilled. A key result in [7] was:

Proposition 1. *An N1C reaction system of the form (4) with SR graph satisfying Condition (∗) is injective.*

Proof. See Theorem 1 in [7].

In [8] this result was strengthened considerably and extended beyond CRNs. In the context of CRNs with N1C kinetics it specialises to:

Proposition 2. *An N1C reaction system of the form (4) with DSR graph satisfying Condition (∗) is injective.*

Proof. See Corollary 4.2 in [8].

Proposition 2 is stronger than Proposition 1 because irreversibility is taken into account. In the case without outflows (2), attention must be restricted to some fixed stoichiometric class. The results then state that no stoichiometry class can contain more than one nondegenerate equilibrium in the interior of the positive orthant [8, 19]. (In this context, a degenerate equilibrium is defined to be an equilibrium with a zero eigenvalue and corresponding eigenvector lying in the stoichiometric subspace.) The case with partial outflows was also treated.

5.2 Monotonicity

A closed, convex, solid, pointed cone $K \subset \mathbb{R}^n$ is termed a **proper cone** [20]. The reader is referred to [20] for basic definitions related to cones. Any proper cone defines a partial order on $\mathbb{R}^n$ as follows: given two points $x, y \in \mathbb{R}^n$:

1. $x \geq y \Leftrightarrow x - y \in K$;
2. $x > y \Leftrightarrow x \geq y$ and $x \neq y$;
3. $x \gg y \Leftrightarrow x - y \in \operatorname{int} K$.

An extremal ray is a one dimensional face of a cone. A proper cone with exactly n extremal rays is termed **simplicial**. Simplicial cones have the feature that unit vectors on the extremal rays can be chosen as basis vectors for a new coordinate system. Consider some linear subspace $\mathcal{A} \subset \mathbb{R}^n$. Then any closed, convex, pointed cone $K \subset \mathcal{A}$ with nonempty interior in $\mathcal{A}$ is termed $\mathcal{A}$-proper. If, further, K has exactly $\dim(\mathcal{A})$ extremal rays, then K is termed $\mathcal{A}$-simplicial.

Consider some local semiflow ϕ defined on $X \subset \mathbb{R}^n$. Assume that there is some linear subspace $\mathcal{A} \subset \mathbb{R}^n$ with a coset $\mathcal{A}^{'}$ with nonempty intersection with X, and such that ϕ leaves $\mathcal{A}^{'} \cap X$ invariant. Suppose further that there is an $\mathcal{A}$-proper cone K such that for all $x, y \in \mathcal{A}^{'} \cap X$, $x > y \Rightarrow \phi_t(x) > \phi_t(y)$ for all values of $t \geq 0$ such that $\phi_t(x)$ and $\phi_t(y)$ are defined. Then we say that $\phi|_{\mathcal{A}' \cap X}$ **preserves** K, and that $\phi|_{\mathcal{A}' \cap X}$ is **monotone**. If, further, $x > y \Rightarrow \phi_t(x) \gg \phi_t(y)$ for all values of $t > 0$ such that $\phi_t(x)$ and $\phi_t(y)$ are defined, then $\phi|_{\mathcal{A}' \cap X}$ is **strongly monotone**. A local semiflow is monotone with respect to the nonnegative orthant if and only if the Jacobian of the vector field has nonnegative off-diagonal elements, in which case the vector field is termed **cooperative**.

Returning to (3), in the case $q = 0$, all stoichiometry classes are invariant, while if $q > 0$, there is a globally attracting stoichiometry class. Conditions for monotonicity of ϕ restricted to invariant subspaces of $\mathbb{R}^n$ were discussed extensively in [12]. Here the immediate aim is to develop graph-theoretic corollaries of one of these results, and to raise some interesting open questions.

Given a vector $y \in \mathbb{R}^n$, define

$$\mathcal{Q}_1(y) \equiv \{v \in \mathbb{R}^n \mid v_i y_i \geq 0\}.$$

A matrix Γ is **R-sorted** (resp. **S-sorted**) if any two distinct columns (resp. rows) Γ_i and Γ_j of Γ satisfy $\Gamma_i \in \mathcal{Q}_1(-\Gamma_j)$. A matrix $\Gamma^{'}$ is **R-sortable** (resp. **S-sortable**) if there exists a signature matrix D such that $\Gamma \equiv \Gamma^{'} D$ (resp. $\Gamma \equiv D\Gamma^{'}$) is well-defined, and is R-sorted (resp. S-sorted).

Proposition 3. *Consider a system of N1C reactions of the form (3) whose stoichiometric matrix Γ is R-sortable, and whose reaction vectors $\{\Gamma_k\}$ are linearly independent. Let $\mathcal{S} = \operatorname{Im}(\Gamma)$. Then there is an $\mathcal{S}$-simplicial cone K preserved by the system restricted to any invariant stoichiometry class, such that each reaction vector is collinear with an extremal ray of K.*

Proof. This is a specialisation of Corollary A7 in [12].

Systems fulfilling the assumptions of Proposition 3, cannot have periodic orbits intersecting the interior of the positive orthant which are stable on their stoichiometry class. In fact, mild additional assumptions ensure strong monotonicity guaranteeing generic convergence of bounded trajectories to equilibria [9, 10].

6 Graph-theoretic implications of Proposition 3

Some more notation is needed for the results to follow. The **S-degree** (**R-degree**) of an SR graph G is the maximum degree of its S-vertices (R-vertices). Analogous to the terminology for matrices, a subgraph E is **R-sorted** (**S-sorted**) if each R-to-R (S-to-S) path E_k in E satisfies $P(E_k) = 1$. Note that E is R-sorted if and only if each R-to-R path E_k of length 2 in E satisfies $P(E_k) = 1$.

An **R-flip** on a SR/DSR graph G is an operation which changes the signs on all edges incident on some R-vertex in G. (This is equivalent to exchanging left and right for the chemical reaction associated with the R-vertex). An **R-resigning** is a sequence of R-flips. An **S-flip** and **S-resigning** can be defined similarly. Given a set of R-vertices $\{R_k\}$ in G, the closed neighbourhood of $\{R_k\}$ will be denoted $G_{\{R_k\}}$, i.e., $G_{\{R_k\}}$ is the subgraph consisting of $\{R_k\}$ along with all edges incident on vertices of $\{R_k\}$, and all S-vertices adjacent to those in $\{R_k\}$.

Proposition 4. *Consider a system of N1C reactions of the form (3) with stoichiometric matrix Γ, and whose reaction vectors $\{\Gamma_k\}$ are linearly independent. Define $\mathcal{S} = \mathrm{Im}(\Gamma)$. Associate with the system the SR graph G. Suppose that*

1. *G has S-degree ≤ 2.*
2. *All cycles in G are e-cycles.*

Then there is an $\mathcal{S}$-simplicial cone K preserved by the system restricted to any invariant stoichiometry class, such that each reaction vector is collinear with an extremal ray of K.

The key idea of the proof is simple: if the system satisfies the conditions of Proposition 4, then the conditions of Proposition 3 are also met. In this case, the extremal vectors of the cone K define a local coordinate system on each stoichiometry class, such that the (restricted) system is cooperative in this coordinate system. This interpretation in terms of recoordinatisation is best illustrated with an example.

Consider **SYS** 1 from (1) with SR graph shown in Figure 2 *left*, which can easily be confirmed to satisfy the conditions of Proposition 4. Define the following matrices:

$$\Gamma = \begin{pmatrix} -1 & 0 & 2 \\ -1 & -1 & 0 \\ 0 & -1 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \qquad T = \begin{pmatrix} -1 & 0 & 2 \\ -1 & 1 & 0 \\ 0 & 1 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}, \qquad T^{'} = \begin{pmatrix} 1 & -2 & 2 & 0 & 0 \\ 1 & -1 & 2 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 \end{pmatrix}$$

$\mathit{\Gamma}$, the stoichiometric matrix, has rank 3, and so Proposition 4 applies. Let $x_1, \ldots, x_5$ be the concentrations of the five substrates involved, v_1, v_2, v_3 be the rates of the three reactions, and $v_{ij} \equiv \frac{\partial v_i}{\partial x_j}$. Assuming that the system is N1C means that $V \equiv [v_{ij}]$ has sign structure

$$\mathrm{sgn}(V) = \begin{pmatrix} + & + & 0 & - & 0 \\ 0 & + & + & 0 & - \\ - & 0 & + & 0 & 0 \end{pmatrix}$$

where $+$ denotes a nonnegative quantity, and $-$ denotes a nonpositive quantity. Consider now any coordinates y satisfying $x = Ty$. Note that T is a re-signed version of $\mathit{\Gamma}$. Choosing some left inverse for T, say $T^{'}$, gives $y_1 = x_1 - 2x_2 + 2x_3$, $y_2 = x_1 - x_2 + 2x_3$ and $y_3 = x_1 - x_2 + x_3$. (The choice of $T^{'}$ is not unique, but this does not affect the argument.) Calculation gives that $J = T^{'}\mathit{\Gamma} VT$ has sign structure

$$\mathrm{sgn}(J) = \begin{pmatrix} - & + & + \\ + & - & + \\ + & + & - \end{pmatrix},$$

i.e., restricting to any invariant stoichiometry class, the dynamical system for the evolution of the quantities y_1, y_2, y_3 is cooperative. Further, the evolution of $\{x_i\}$ is uniquely determined by the evolution of $\{y_i\}$ via the equation $x = Ty$.

It is time to return to the steps leading to the proof of Proposition 4. In Lemmas 1 and 2 below, G is an SR graph with S-degree ≤ 2. This implies the following: consider R-vertices v, $v^{'}$ and $v^{''}$ such that $v \neq v^{'}$ and $v \neq v^{''}$ ($v^{'} = v^{''}$ is possible). Assume there exist two distinct short paths in G, one from v to $v^{'}$ and one from v to $v^{''}$. These paths must be edge disjoint, for otherwise there must be an S-vertex lying on both A and B, and hence having degree ≥ 3.

Lemma 1. *Suppose G is a connected SR graph with S-degree ≤ 2, and has some connected, R-sorted, subgraph E containing R-vertices $v^{'}$ and $v^{''}$. Assume that there is a path C_1 of length 4 between $v^{'}$ and $v^{''}$ containing an R-vertex not in E. Then either C_1 is even or G contains an o-cycle.*

Proof. If $v^{'} = v^{''}$, then C_1 is not even, then it is itself and e-cycle. Otherwise consider any path C_2 connecting $v^{'}$ and $v^{''}$ and lying entirely in E. C_2 exists since E is connected, and $P(C_2) = 1$ since E is R-sorted. Since G has S-degree ≤ 2, and $|C_1| = 4$, C_1 and C_2 share only endpoints, $v^{'}$ and $v^{''}$, and hence together they form a cycle C. If $P(C_1) = -1$, then $P(C) = P(C_2)P(C_1) = -1$, and so C is an o-cycle. □

Lemma 2. *Suppose G is a connected SR graph with S-degree ≤ 2 which does not contain an o-cycle. Then it can be R-sorted.*

Proof. The result is trivial if G contains a single R-vertex, as it contains no short R-to-R paths. Suppose the result is true for graphs containing k R-vertices. Then it must be true for graphs containing $k+1$ R-vertices. Suppose G contains

$k+1$ R-vertices. Enumerate these R-vertices as $R_1, \ldots, R_{k+1}$ in such a way that $G_- \equiv G_{\{R_1,\ldots,R_k\}}$ is connected. This is possible since G is connected.

By the induction hypothesis, G_- can be R-sorted. Having R-sorted G_-, consider R_{k+1}. If all short paths between R_{k+1} and R-vertices in G_- have the same parity, then either they are all even and G is R-sorted; or they are all odd, and a single R-flip on R_{k+1} R-sorts G. (Note that an R-flip on R_{k+1} does not affect the parity of any R-to-R paths in G_-.) Otherwise there must be two distinct short paths of opposite sign, between R_{k+1} and R-vertices $v^{'}, v^{''} \in G_-$ ($v^{'} = v^{''}$ is possible). Since G has S-degree ≤ 2, these paths must be edge-disjoint, and together form an odd path of length 4 from $v^{'}$ to R_{k+1} to $v^{''}$. By Lemma 1, G contains an o-cycle. □

PROOF of Proposition 4. From Lemma 2, if no connected component of G contains an o-cycle then each connected component of G (and hence G itself) can be R-sorted. The fact that G can be R-sorted corresponds to choosing a signing of the stoichiometric matrix Γ such that any two columns Γ_i and Γ_j satisfy $\Gamma_i \in \mathcal{Q}_1(-\Gamma_j)$. Thus the conditions of Proposition 3 are satisfied. □

7 Examples illustrating the result and its limitations

Example 1: SYS n from Section 1. It is easy to confirm that the reactions in **SYS** n have linearly independent reaction vectors for all n . Moreover, as illustrated by Figure 2, the corresponding SR graphs contain a single cycle, which, for odd (even) n is an e-cycle (o-cycle). Thus for even n, Proposition 1 and subsequent remarks apply, ruling out the possibility of more than one positive nondegenerate equilibrium for (2) on each stoichiometry class, or in the case with outflows (4), ruling out multiple equilibria altogether; meanwhile, while for odd n, Proposition 4 can be applied to (2) or (3), implying that restricted to any invariant stoichiometry class the system is monotone, and the restricted dynamical system cannot have an attracting periodic orbit intersecting the interior of the nonnegative orthant.

Example 2: Generalised interconversion networks. Consider the following system of chemical reactions:

$$A \rightleftharpoons B, \quad A \rightleftharpoons C, \quad A \rightleftharpoons D, \quad B \rightleftharpoons C \tag{7}$$

with SR graph shown in Figure 3. Formally, such systems have R-degree ≤ 2 and have SR graphs which are S-sorted. Although Proposition 4 cannot be applied, such "interconversion networks", with the N1C assumption, in fact give rise to cooperative dynamical systems [12], and a variety of different techniques give strong convergence results, both with and without outflows [16, 11, 21].

This example highlights that there is an immediate dual to Lemma 2, and hence Proposition 4. The following lemma can be regarded as a restatement of well-known results on systems preserving orthant cones (see [10], for example, and the discussion for CRNs in [11]). Its proof is omitted as it follows closely that of Lemma 2.

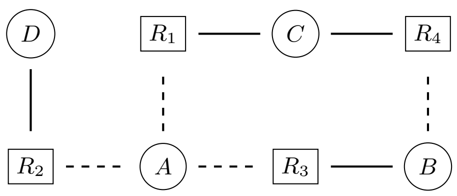


Fig. 3. The SR graph for reaction system 7. All edge labels are 1 and have been omitted. The system preserves the nonnegative orthant.

Lemma 3. *Let G be an SR graph with R-degree ≤ 2 and containing no o-cycles. Then, via an S-resigning, G can be S-sorted.*

Although the S-sorting process is formally similar to the R-sorting one, the interpretation of the result is quite different: changing the sign of the ith row of Γ and the ith column of V is equivalent to a recoordinatisation replacing concentration x_i with $-x_i$. Such recoordinatisations give rise to a cooperative system if and only if the original system is monotone with respect to an orthant cone.

Example 3: Linearly independent reaction vectors are not necessary for monotonicity. Consider the system of three reactions involving four substrates

$$A \rightleftharpoons B + C, \qquad B \rightleftharpoons D, \qquad C + D \rightleftharpoons A \tag{8}$$

with stoichiometric matrix Γ and SR graph shown in Figure 4.

$$\Gamma = \begin{pmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix}$$

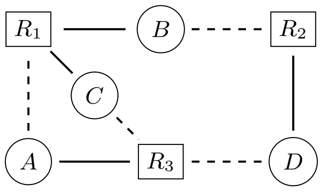


Fig. 4. The stoichiometric matrix and SR graph for reaction system 8. All edge labels are 1 and have been omitted.

Note that Γ is R-sorted, but has rank 2 as all row-sums are zero. As before, let x_i be the concentrations of the four substrates involved. Now, choose new coordinates y satisfying $x = Ty$, where

$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Note: i) T has rank 3, ii) $\mathrm{Im}(\Gamma) \subset \mathrm{Im}(T)$, and iii) regarding the columns of T as extremal vectors of a cone K, K has trivial intersection with $\mathrm{Im}(\Gamma)$. One can proceed to choose some left inverse $T^{'}$ of T, and calculate that the Jacobian $J = T^{'}\Gamma VT$ has nonnegative off-diagonal entries. In other words the y-variables define a cooperative dynamical system. The relationship between T and Γ is further discussed in the concluding section.

Note that although K has empty interior in $\mathbb{R}^4$, both K and $\mathrm{Im}(\Gamma)$ lie in the hyperplane $H = \mathrm{Im}(T)$ defined by $x_1 + x_3 = 0$. As K is H-proper, attention can be restricted to invariant cosets of H. With mild additional assumptions on the kinetics, the theory in [21] can be applied to get strong convergence results, but this is not pursued here.

Example 4a: The absence of o-cycles is not necessary for monotonicity. Consider the following system of 4 chemical reactions on 5 substrates:

$$A \rightleftharpoons B + C, \qquad B \rightleftharpoons D, \qquad C + D \rightleftharpoons A \qquad C + E \rightleftharpoons A \tag{9}$$

Define

$$\Gamma = \begin{pmatrix} -1 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \quad \text{and} \quad T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Γ, the stoichiometric matrix, has rank 3, and the system has SR graph containing both e- and o-cycles (Figure 5). Further, there are substrates participating in 3 reactions, and reactions involving 3 substrates (and so it is neither R-sortable nor S-sortable). Thus, all the conditions for the results quoted so far in this paper, and for theorems in [11], are immediately violated. However, applying theory in [12], the system is order preserving. In particular, $\mathrm{Im}(T)$ is a 4D subspace of $\mathbb{R}^5$ containing $\mathrm{Im}(\Gamma)$ (the stoichiometric subspace), and T defines a cone K which is preserved by the system restricted to cosets of $\mathrm{Im}(T)$.

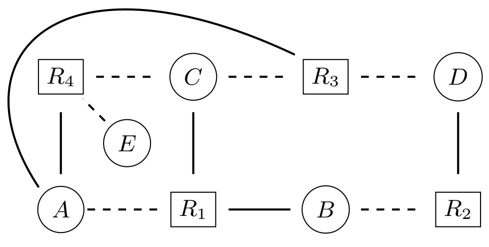


Fig. 5. The SR graph for reaction system 9. All edge labels are 1 and have been omitted.

Example 4b: The absence of o-cycles is not necessary for monotonicity. Returning to the system of reactions in (5), the system has SR graph containing an o-cycle (Figure 1, *left*). Nevertheless, the system was shown in [12] to preserve a *nonsimplicial* cone for all N1C kinetics. In fact, the further analysis in [21] showed that with mild additional assumptions this system is strongly monotone and all orbits on each stoichiometry class converge to an equilibrium which is unique on that stoichiometry class. It is worth mentioning that this example is fundamentally different from Example 4a, and that it is currently unclear how commonly reaction systems preserve orders generated by nonsimplicial cones.

8 Discussion and open questions

The results presented here provide only a glimpse of the possibilities for analysis of limit sets of CRNs using graph-theoretic – and more generally combinatorial – approaches. The literature in this area is growing rapidly, and new techniques are constantly being brought into play. Working with the weakest possible kinetic assumptions often gives rise to approaches quite different from those used in the previous study of mass-action systems. Conversely, it is possible that such approaches can be used to provide explicit restrictions on the kinetics for which a system displays some particular behaviour.

The paper highlights an interesting duality between questions of multistationarity and questions of stable periodic behaviour, a duality already implicit in discussions of interaction graphs [22–25]. Loosely, the absence of e-cycles (positive cycles) is associated with injectivity for systems described by SR graphs (I graphs); and the absence of o-cycles (negative cycles) is associated with absence of periodic attractors for systems described by SR graphs (I graphs). The connections between apparently unrelated SR and I graph results on injectivity have been clarified in [26], but there is still considerable work to be done to clarify the results on monotonicity.

One open question regards the relationship between the theory and examples presented here on monotonicity, and previous results, particularly Theorem 1 in [11], on monotonicity in “reaction coordinates”. Note that by Proposition 4.5 in [11] the “positive loop property” described there is precisely Conditions 1 and 2 in Proposition 4 here. At the same time, the requirement that the stoichiometric matrix has full rank, is not needed for monotonicity in reaction coordinates. In some cases (e.g. Example 3 above), it can be shown that this requirement is unnecessary for monotonicity too, but it is currently unclear whether this is always the case. On the other hand, as illustrated by Examples 4a and 4b, the positive loop property is not needed for monotonicity.

Consider again Examples 3 and 4a. The key fact is that their stoichiometric matrices admit factorisations $\Gamma = T_1T_2$, taking the particular forms

$$\begin{pmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{pmatrix} \qquad \text{(Example 3), and}$$

$$\begin{pmatrix} -1 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \qquad \text{(Example 4a)}.$$

In each case, the first factor, T_1, has exactly one nonzero entry in each row. On the other hand, the second factor, T_2, is S-sorted. The theory in [12] ensures that these conditions are sufficient (though not necessary) to guarantee that the system restricted to some coset of $\mathrm{Im}(T_1)$, is monotone with respect to the order defined by T_1. The dynamical implications of this factorisation result will be elaborated on in future work.

A broad open question concerns the extent to which the techniques presented here extend to systems with discrete-time, and perhaps also discrete-state space. In [6], there were shown to be close relationships, but also subtle differences, between results on persistence in the continuous-time, continuous-state context, and results on liveness in the discrete-time, discrete-state context. Even discretising only time can lead to difficulties: while the interpretation of injectivity results in the context of discrete-time, continuous-state, systems is straightforward, the dynamical implications of monotonicity can differ from the continuous-time case. For example, strongly monotone disrete-time dynamical systems may have stable k-cycles for $k \geq 2$ [27]. When the state space is discrete, an additional difficulty which may arise concerns differentiability of the associated functions, an essential requirement for the results presented here.

Finally, the work on monotonicity here has an interesting relationship with examples presented by Kunze and Siegel, for example in [28]. This connection remains to be explored and clarified.

References

1. I. R. Epstein and J. A. Pojman, editors. *An Introduction to Nonlinear Chemical Dynamics: Oscillations, Waves, Patterns, and Chaos.* Oxford University Press, New York, 1998.
2. M. Feinberg. Complex balancing in general kinetic systems. *Arch. Ration. Mech. Anal.*, 49(3):187–194, 1972.
3. M. Feinberg. Chemical reaction network structure and the stability of complex isothermal reactors - I. The deficiency zero and deficiency one theorems. *Chem. Eng. Sci.*, 42(10):2229–2268, 1987.
4. G. Craciun and M. Feinberg. Multiple equilibria in complex chemical reaction networks: II. The species-reaction graph. *SIAM J. Appl. Math.*, 66(4):1321–1338, 2006.
5. M. Mincheva and M. R. Roussel. Graph-theoretic methods for the analysis of chemical and biochemical networks, I. multistability and oscillations in ordinary differential equation models. *J. Math. Biol.*, 55:61–86, 2007.
6. D. Angeli, P. De Leenheer, and E. D. Sontag. A Petri net approach to the study of persistence in chemical reaction networks. *Math. Biosci.*, 210:598–618, 2007.

7. M. Banaji and G. Craciun. Graph-theoretic criteria for injectivity and unique equilibria in general chemical reaction systems. *Adv. in Appl. Math.*, 44:168–184, 2010.
8. M. Banaji and G. Craciun. Graph-theoretic approaches to injectivity and multiple equilibria in systems of interacting elements. *Commun. Math. Sci.*, 7(4):867–900, 2009.
9. M.W. Hirsch and H. Smith. *Handbook of Differential Equations: Ordinary Differential Equations, Vol II*, chapter Monotone Dynamical Systems, pages 239–357. Elsevier B. V., Amsterdam, 2005.
10. H. Smith. *Monotone Dynamical Systems: An introduction to the theory of competitive and cooperative systems.* American Mathematical Society, 1995.
11. D. Angeli, P. De Leenheer, and E. D. Sontag. Graph-theoretic characterizations of monotonicity of chemical reaction networks in reaction coordinates. *J. Math. Biol.*, DOI 10.1007/s00285-009-0309-0, 2009.
12. M. Banaji. Monotonicity in chemical reaction systems. *Dyn. Syst.*, 24(1):1–30, 2009.
13. G. Craciun and M. Feinberg. Multiple equilibria in complex chemical reaction networks: I. The injectivity property. *SIAM J Appl Math*, 65(5):1526–1546, 2005.
14. M. Banaji, P. Donnell, and S. Baigent. P matrix properties, injectivity and stability in chemical reaction systems. *SIAM J. Appl. Math.*, 67(6):1523–1547, 2007.
15. P. De Leenheer, D. Angeli, and E.D. Sontag. Monotone chemical reaction networks. *J. Math. Chem.*, 41(3):295–314, 2007.
16. M. Banaji and S. Baigent. Electron transfer networks. *J. Math. Chem.*, 43(4), 2008.
17. R. David and H. Alla. Autonomous and timed continous Petri nets. In *Papers from the 12th International Conference on Applications and Theory of Petri Nets: Advances in Petri Nets 1993*, volume 674 of *Lecture Notes In Computer Science*, pages 71–90. 1991.
18. F. Bause and P. S. Kritzinger. *Stochastic Petri nets.* Vieweg, 2nd edition, 2002.
19. G. Craciun and M. Feinberg. Multiple equilibria in complex chemical reaction networks: Extensions to entrapped species models. *IEEE Proc. Syst. Biol.*, 153(4):179–186, 2006.
20. A. Berman and R. Plemmons. *Nonnegative matrices in the mathematical sciences.* Academic Press, New York, 1979.
21. M. Banaji and D. Angeli. Convergence in strongly monotone systems with an increasing first integral. *SIAM J. Math. Anal.*, 42(1):334–353, 2010.
22. J.-L. Gouzé. Positive and negative circuits in dynamical systems. *J Biol Sys*, 6:11–15, 1998.
23. C. Soulé. Graphic requirements for multistationarity. *Complexus*, 1:123–133, 2003.
24. M. Kaufman, C. Soulé, and R. Thomas. A new necessary condition on interaction graphs for multistationarity. *J. Theor. Biol.*, 248(4):675–685, 2007.
25. D. Angeli, M. W. Hirsch, and E. Sontag. Attractors in coherent systems of differential equations. *J. Diff. Eq.*, 246:3058–3076, 2009.
26. M. Banaji. Graph-theoretic conditions for injectivity of functions on rectangular domains. *J. Math. Anal. Appl.*, 370:302–311, 2010.
27. J. F. Jiang and S. X. Yu. Stable cycles for attractors of strongly monotone discrete-time dynamical systems. *J. Math. Anal. Appl.*, 202:349–362, 1996.
28. H. Kunze and D. Siegel. A graph theoretic approach to strong monotonicity with respect to polyhedral cones. *Positivity*, 6:95–113, 2002.

On the Importance of the Deadlock Trap Property for Monotonic Liveness

Monika Heiner[1], Cristian Mahulea[2], Manuel Silva[2]

[1] Department of Computer Science, Brandenburg University of Technology
Postbox 10 13 44, 03013 Cottbus, Germany
monika.heiner@tu-cottbus.de
[2] Instituto de Investigación en Ingeniería de Aragón (I3A),
Universidad de Zaragoza, Maria de Luna 1, E-50018 Zaragoza, Spain
cmahulea@unizar.es, silva@unizar.es

Abstract. In Petri net systems, liveness is an important property capturing the idea of no transition (action) becoming non-fireable (unattainable). Additionally, in some situations it is particularly interesting to check if the net system is (marking) monotonically live, i.e., it remains live for any marking greater than the initial one. In this paper, we discuss structural conditions preserving liveness under arbitrary marking increase. It is proved that the deadlock trap property (DTP) is a necessary condition for liveness monotonicity of ordinary nets, and necessary and sufficient for some subclasses. We illustrate also how the result can be used to study liveness monotonicity for non-ordinary nets using a simulation preserving the firing language. Finally, we apply these conditions to several case studies of biomolecular networks.

1 Motivation

Petri nets are a natural choice to represent biomolecular networks. Various types of Petri nets may be useful – qualitative, deterministically timed, stochastic, continuous or hybrid ones, depending on the available information and the kind of properties to be analysed. Accordingly, the integrative framework demonstrated by several case studies in [GHR$^+$08], [HGD08], [HDG10] applies a family of related Petri net models, sharing structure, but differing in their kind of kinetic information.

A key notion of the promoted strategy of biomodel engineering is the *level concept*, which has been introduced in the Petri net framework in [GHL07]. Here, a token stands for a specific amount of mass, defined by the total mass divided by the number of levels. Thus, increasing the token number to represent a certain amount of mass means to increase the resolution of accuracy.

This procedure silently assumes some kind of behaviour preservation while the marking is increased (typically multiplied by a factor) to represent a finer granularity of the mass flowing through the network. However, as it is well-known in Petri net theory, liveness is not monotonic with respect to (w.r.t.) the initial marking for general Petri nets. Thus, there is no reason to generally assume that there is no significant change in the possible behaviour by marking increase. Contrary, under liveness monotonicity w.r.t. the initial marking we can expect continuization (fluidization) to be reasonable. However, only a particular kind of monotonicity seems to be needed for continuization: *homothetic liveness*, i.e., liveness preservation while multiplying the initial marking by k [RTS99], [SR02].

At structural level, (monotonic) liveness can be considered using transformation (reduction) rules [Ber86], [Sil85], [Mur89], [Sta90], the classical analysis for ordinary nets based on the Deadlock Trap Property (DTP) [Mur89], [Sta90], or the results of *Rank Theorems*, which are directly applicable to non-ordinary nets [TS96], [RTS98]. In this paper, we concentrate on the DTP, which will initially be used for ordinary net models, and later extended to non-ordinary ones.

This paper is organized as follows. We start off with recalling relevant notions and results of Petri net theory. Afterwards we introduce the considered subject by looking briefly at two examples, before turning to our main result yielding a necessary condition for monotonic liveness. We demonstrate the usefulness of our results for the analysis of biomolecular networks by a variety of case studies. We conclude with an outlook on open issues.

2 Preliminaries

We assume basic knowledge of the standard notions of place/transition Petri nets, see e.g. [DHP+93], [HGD08], [DA10]. To be self-contained we recall the fundamental notions relevant for our paper.

Definition 1 (Petri net, syntax).
A Petri net *is a tuple $\mathcal{N} = \langle P, T, \boldsymbol{Pre}, \boldsymbol{Post} \rangle$, and a* Petri net system *is a tuple $\Sigma = \langle \mathcal{N}, \boldsymbol{m}_0 \rangle$, where*

- *P and T are finite, non-empty, and disjoint sets. P is the set of* places*. T is the set of* transitions*.*
- *$\boldsymbol{Pre}, \boldsymbol{Post} \in \mathbb{N}^{|P|\times|T|}$ are the pre- and post-matrices, where $|\cdot|$ is the cardinality of a set, i.e., its number of elements. For a place $p_i \in P$ and a transition $t_j \in T$, $Pre(p_i, t_j)$ is the weight of the arc connecting p_i to t_j (0 if there is no arc), while $Post(p_i, t_j)$ is the weight of the arc connecting t_j to p_i.*
- *$\boldsymbol{m}_0 \in \mathbb{N}^{|P|}_{\geq 0}$ gives the* initial marking*.*
- *$\boldsymbol{m}(p)$ yields the number of tokens on place p in the marking $\boldsymbol{m}$. A place p with $\boldsymbol{m}(p) = 0$ is called* empty (unmarked) *in $\boldsymbol{m}$, otherwise it is called* marked *(non-empty). A set of places is called empty if all its places are empty, otherwise marked.*
- *The* preset *and* postset *of a node $x \in P \cup T$ are denoted by ${}^\bullet x$ and $x^\bullet$. They represent the input and output transitions of a place x, or the input and output places of a transition x. More specifically, if $t_j \in T$, ${}^\bullet t_j = \{p_i \in P | Pre(p_i, t_j) > 0\}$ and $t_j{}^\bullet = \{p_i \in P | Post(p_i, t_j) > 0\}$. Similarly, if $p_i \in P$, ${}^\bullet p_i = \{t_j \in T | Post(p_i, t_j) > 0\}$ and $p_i{}^\bullet = \{t_j \in T | Pre(p_i, t_j) > 0\}$.*
  We extend both notions to a set of nodes $X \subseteq P \cup T$ and define the set of all prenodes ${}^\bullet X := \bigcup_{x \in X} {}^\bullet x$, and the set of all postnodes $X^\bullet := \bigcup_{x \in X} x^\bullet$.

– *A node* $x \in P \cup T$ *is called* source node*, if* $^\bullet x = \emptyset$*, and* sink node *if* $x^\bullet = \emptyset$*. A* boundary node *is either a sink or a source node (but not both, because we assume a connected net).*

Definition 2 (Petri net, behaviour). *Let* $\langle \mathcal{N}, \boldsymbol{m}_0 \rangle$ *be a net system.*

– *A transition* t *is* enabled *at marking* $\boldsymbol{m}$*, written as* $\boldsymbol{m}[t\rangle$*, if* $\forall p \in {}^\bullet t : \boldsymbol{m}(p) \geq Pre(p,t)$*, else* disabled*.*
– *A transition* t*, enabled in* $\boldsymbol{m}$*, may* fire *(occur), leading to a new marking* $\boldsymbol{m'}$*, written as* $\boldsymbol{m}[t\rangle\boldsymbol{m'}$*, with* $\forall p \in P : \boldsymbol{m'}(p) = \boldsymbol{m}(p) - Pre(p,t) + Post(p,t)$*.*
– *The set of all markings* reachable *from a marking* $\boldsymbol{m}_0$*, written as* $[\boldsymbol{m}_0\rangle$*, is the smallest set such that* $\boldsymbol{m}_0 \in [\boldsymbol{m}_0\rangle$*,* $\boldsymbol{m} \in [\boldsymbol{m}_0\rangle \wedge \boldsymbol{m}[t\rangle\boldsymbol{m'} \Rightarrow \boldsymbol{m'} \in [\boldsymbol{m}_0\rangle$*.*
– *The* reachability graph (RG) *is a directed graph with* $[\boldsymbol{m}_0\rangle$ *as set of nodes, and the labelled arcs denote the reachability relation* $\boldsymbol{m}[t\rangle\boldsymbol{m'}$*.*

Definition 3 (Behavioural properties). *Let* $\langle \mathcal{N}, \boldsymbol{m}_0 \rangle$ *be a net system.*

– *A place* p *is* k-bounded *(bounded for short) if there is a positive integer number* k*, serving as an upper bound for the number of tokens on this place in all reachable markings of the Petri net:* $\exists\, k \in \mathbb{N}_0 : \forall \boldsymbol{m} \in [\boldsymbol{m}_0\rangle : \boldsymbol{m}(p) \leq k$ *.*
– *A Petri net system is* k-bounded *(bounded for short) if all its places are* k-bounded.
– *A transition* t *is* dead at marking $\boldsymbol{m}$ *if it is not enabled in any marking* $\boldsymbol{m'}$ *reachable from* $\boldsymbol{m}$*:* $\nexists\, \boldsymbol{m'} \in [\boldsymbol{m}\rangle : \boldsymbol{m'}[t\rangle$*.*
– *A transition* t *is* live *if it is not dead in any marking reachable from* $\boldsymbol{m}_0$*.*
– *A marking* $\boldsymbol{m}$ *is* dead *if there is no transition which is enabled in* $\boldsymbol{m}$*.*
– *A Petri net system is* deadlock-free (weakly live) *if there are no reachable dead markings.*
– *A Petri net system is* live (strongly live) *if each transition is live.*

Definition 4 (Net structures). *Let* $\mathcal{N} = \langle P, T, \boldsymbol{Pre}, \boldsymbol{Post} \rangle$ *be a Petri net.* $\mathcal{N}$ *is*

– Homogeneous (HOM) *if* $\forall p \in P : t, t' \in p^\bullet \Rightarrow Pre(p,t) = Pre(p,t')$*;*
– Ordinary (ORD) *if* $\forall p \in P$ *and* $\forall t \in T$*,* $Pre(p,t) \leq 1$ *and* $Post(p,t) \leq 1$*;*
– Extended Simple (ES) *(sometimes also called* asymmetric choice*) if it is ORD and* $\forall\, p, q \in P : p^\bullet \cap q^\bullet = \emptyset \vee p^\bullet \subseteq q^\bullet \vee q^\bullet \subseteq p^\bullet$*;*
– Extended Free Choice (EFC) *if it is ORD and* $\forall\, p, q \in P : p^\bullet \cap q^\bullet = \emptyset \vee p^\bullet = q^\bullet$*.*

Definition 5 (DTP). *Let* $\mathcal{N} = \langle P, T, \boldsymbol{Pre}, \boldsymbol{Post} \rangle$ *be a Petri net.*

– *A* siphon *(structural deadlock, co-trap) is a non-empty set of places* $D \subseteq P$ *with* $^\bullet D \subseteq D^\bullet$ *.*
– *A* trap *is a non-empty set of places* $Q \subseteq P$ *with* $Q^\bullet \subseteq {}^\bullet Q$*.*
– *A* minimal siphon (trap) *is a siphon (trap) not including a siphon (trap) as a proper subset.*
– *A* bad siphon *is a siphon, which does not include a trap.*

– *An* empty siphon (trap) *is a siphon (trap), not containing a token.*
– *The* Deadlock Trap Property (DTP) *asks for every siphon to include an initially marked trap, i.e., marked at* $\boldsymbol{m}_0$*.*

The DTP can be reformulated as: minimal siphons are not bad and the maximal traps included are initially marked.

Definition 6 (Semiflows). *Let* $\mathcal{N} = \langle P, T, \boldsymbol{Pre}, \boldsymbol{Post} \rangle$ *be a net.*

– *The* token flow matrix *(or* incidence matrix *if the net is pure, i.e., self-loop free) is a matrix* $\boldsymbol{C} = \boldsymbol{Post} - \boldsymbol{Pre}$*.*
– *A* place vector *is a vector* $\boldsymbol{y} \in \mathbb{Z}^{|P|}$*; a* transition vector *is a vector* $\boldsymbol{x} \in \mathbb{Z}^{|T|}$.
– *A* P-semiflow *is a place vector* $\boldsymbol{y}$ *with* $\boldsymbol{y} \cdot \boldsymbol{C} = \boldsymbol{0}$*,* $\boldsymbol{y} \geq \boldsymbol{0}$*,* $\boldsymbol{y} \neq \boldsymbol{0}$*;*
a T-semiflow *is a transition vector* $\boldsymbol{x}$ *with* $\boldsymbol{C} \cdot \boldsymbol{x} = \boldsymbol{0}$*,* $\boldsymbol{x} \geq \boldsymbol{0}$*,* $\boldsymbol{x} \neq \boldsymbol{0}$*.*
– *The* support of a semiflow $\boldsymbol{x}$*, written as* $supp(\boldsymbol{x})$*, is the set of nodes corresponding to the non-zero entries of* $\boldsymbol{x}$*.*
– *A net is* conservative *if every place belongs to the support of a P-semiflow.*
– *A net is* consistent *if every transition belongs to the support of a T-semiflow.*
– *In a* minimal semiflow $\boldsymbol{x}$*,* $supp\,(\boldsymbol{x})$ *does not contain the support of any other semiflow* $\boldsymbol{z}$*, i.e.,* $\nexists$ *semiflow* $\boldsymbol{z} : supp\,(\boldsymbol{z}) \subset supp\,(\boldsymbol{x})$*, and the greatest common divisor of* $\boldsymbol{x}$ *is 1.*
– *A* mono-T-semiflow net *(MTS net) is a consistent and conservative net that has exactly one minimal T-semiflow.*

For convenience, we give vectors (markings, semiflows) in a short-hand notation by enumerating only the non-zero entries. Finally, we recall some well-known related propositions (see for example [Mur89], [Sta90]), which might be useful for the reasoning we pursue in this paper.

Proposition 1 (Basics).

1. *An empty siphon remains empty forever. A marked trap remains marked for ever.*
2. *If* R *and* R' *are siphons (traps), then* $R \cup R'$ *is also a siphon (trap).*
3. *A minimal siphon (trap) is a P-strongly-connected component, i.e., its places are strongly connected.*
4. *A deadlocked Petri net system has an empty siphon.*
5. *Each siphon of a live net system is initially marked.*
6. *If there is a bad siphon, the DTP does not hold.*
7. *A source place* p *establishes a bad siphon* $D = \{p\}$ *on its own, and a sink place* q *a trap* $Q = \{q\}$*.*
8. *If each transition has a pre-place, then* $P^{\bullet} = T$*, and if each transition has a post-place, then* ${}^{\bullet}P = T$*. Thus, in a net without boundary transitions, the whole set of places is a siphon as well as a trap (however, not necessarily minimal ones).*
9. *For a P-semiflow* $\boldsymbol{x}$ *it holds* ${}^{\bullet}supp(\boldsymbol{x}) = supp(\boldsymbol{x})^{\bullet}$*. Thus, the support of a P-semiflow is siphon and trap as well (however, generally not vice versa).*

Proposition 2 (DTP and behavioural properties).

1. *An ordinary Petri net without siphons is live.*
2. *If $\mathcal{N}$ is ordinary and the DTP holds for $\boldsymbol{m}_0$, then $\langle \mathcal{N}, \boldsymbol{m}_0 \rangle$ is deadlock-free.*
3. *If $\mathcal{N}$ is ES and the DTP holds for $\boldsymbol{m}_0$, then $\langle \mathcal{N}, \boldsymbol{m}_0 \rangle$ is live.*
4. *Let $\mathcal{N}$ be an EFC net. $\langle \mathcal{N}, \boldsymbol{m}_0 \rangle$ is live iff the DTP holds.*

We conclude this section with a proposition from [CCS91], which might be less known.

Proposition 3 (MTS net and behavioural properties). *Liveness and deadlock-freeness coincide in mono-T-semiflow net systems.*

3 Monotonic Liveness

If a property holds for a Petri net $\mathcal{N}$ with the marking $\boldsymbol{m}_0$, and it also holds in $\mathcal{N}$ for any $\boldsymbol{m} \geq \boldsymbol{m}_0$, then it is said to be *monotonic* in the system $\langle \mathcal{N}, \boldsymbol{m}_0 \rangle$. In this paper we are especially interested in monotonic liveness.

Definition 7 (Monotonic liveness).
Let $\langle \mathcal{N}, \boldsymbol{m}_0 \rangle$ be a Petri net system. It is called monotonically live, if being live for $\boldsymbol{m}_0$, it remains live for any $\boldsymbol{m} \geq \boldsymbol{m}_0$.

We are looking for conditions, at best structural conditions, preserving liveness under arbitrary marking increase. To illustrate the problem, let's consider a classical example [Sta90], [SR02].

Example 1. The net $\mathcal{N}$ in Figure 1 is ES, conservative, consistent, and covered by one T-semiflow. It is live for the given initial marking $\boldsymbol{m}_1 = (2p_1, p_4)$. Adding a token to place p_5 yields the initial marking $\boldsymbol{m}_2 = (2p_1, p_4, p_5)$ and the net system remains live for $\boldsymbol{m}_2 \geq \boldsymbol{m}_1$. However, adding a token to p_4 yields the initial marking $\boldsymbol{m}_3 = (2p_1, 2p_4)$ and the net behaviour now contains finite firing sequences, i.e., it can run into a deadlock (dead state). Thus, the net system is not live for $\boldsymbol{m}_3 \geq \boldsymbol{m}_1$. It is not monotonically live.

How to distinguish both cases? The net has two (minimal) bad siphons $D_1 = \{p_1, p_2\}$ and $D_2 = \{p_1, p_3\}$. There is no chance to prevent these siphons from getting empty for arbitrary markings. D_1 can potentially be emptied by firing $t_2 \in D_1{}^\bullet \setminus {}^\bullet D_1$, and D_2 by firing $t_1 \in D_2{}^\bullet \setminus {}^\bullet D_2$. The latter case destroyed the liveness for $\boldsymbol{m}_3$ as it will equally occur for all initial markings allowing transition sequences containing one of the troublemakers, in this example t_1 and t_2, sufficiently often. $\square$

One lesson learnt from the previous example is, a net does not have to make use of the additional tokens. Thus, all behaviour (set of transition sequences), which is possible for $\boldsymbol{m}$ is still possible for $\boldsymbol{m}'$, with $\boldsymbol{m} \leq \boldsymbol{m}'$. However, new tokens may allow for additional system behaviour, which is actually well-known in Petri net theory, see Proposition 4.

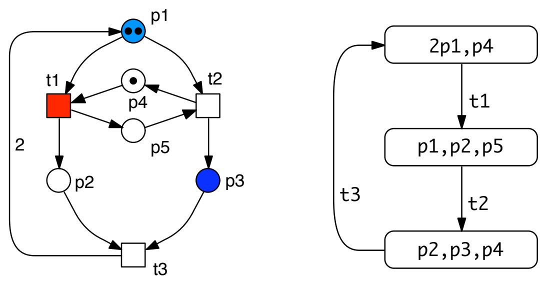


Fig. 1. A mono-T-semiflow and ES Petri net $\mathcal{N}$ and its reachability graph for the marking $\boldsymbol{m}_1 = (2p_1, p_4)$, generating the language $\mathcal{L}_{\mathcal{N}}(\boldsymbol{m}_1) = (t_1t_2t_3)^*\{\varepsilon, t_1, t_1t_2\}$. The siphon $\{p_1, p_3\}$ does not contain a trap, i.e., it is a bad siphon. If the initial marking is increased, it can potentially become empty by firing of t_1.

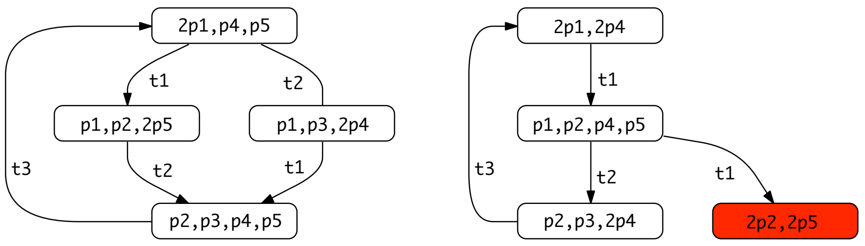


Fig. 2. Two other reachability graphs for the net $\mathcal{N}$ in Figure 1 for the initial markings $\boldsymbol{m}_2 = (2p_1, p_4, p_5)$ and $\boldsymbol{m}_3 = (2p_1, 2p_4)$; both are greater than $\boldsymbol{m}_1$. Obviously, $\mathcal{L}_{\mathcal{N}}(\boldsymbol{m}_1) \subset \mathcal{L}_{\mathcal{N}}(\boldsymbol{m}_2)$, $\mathcal{L}_{\mathcal{N}}(\boldsymbol{m}_1) \subset \mathcal{L}_{\mathcal{N}}(\boldsymbol{m}_3)$, but $\langle \mathcal{N}, \boldsymbol{m}_3 \rangle$ is not live while $\langle \mathcal{N}, \boldsymbol{m}_1 \rangle$ is live.

Proposition 4. *For any net $\mathcal{N}$ and two markings $\boldsymbol{m}$ and $\boldsymbol{m}'$, with $\boldsymbol{m} \leq \boldsymbol{m}'$, it holds $\mathcal{L}_{\mathcal{N}}(\boldsymbol{m}) \subseteq \mathcal{L}_{\mathcal{N}}(\boldsymbol{m}')$ [BRA83]; nevertheless, $\langle \mathcal{N}, \boldsymbol{m} \rangle$ may be live while $\langle \mathcal{N}, \boldsymbol{m}' \rangle$ not.*

Example 1 is a mono-T-semiflow net, i.e., a net, where liveness and deadlock-freeness coincide (see Proposition 3). We look briefly at Example 2 to understand that this does not generally hold if there are several T-semiflows breathing life into the net.

Example 2. The net $\mathcal{N}$ in Figure 3 is a slight extension of Example 1. It is ES, conservative, consistent and covered by two T-semiflows: $\boldsymbol{x}_1 = (t_1, t_2, t_3), \boldsymbol{x}_2 = (t_4, t_5)$. It is live for the initial marking $\boldsymbol{m}_1 = (2p_1, p_4)$.

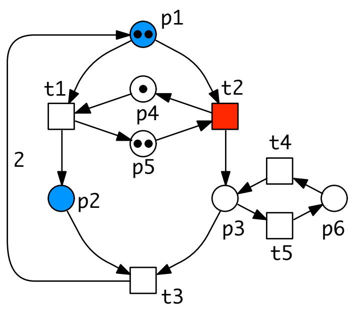


Fig. 3. An ES Petri net which is not mono-T-semiflow. It is live for the initial marking $\boldsymbol{m}_1 = (2p_1, p_4)$. The siphon $\{p_1, p_2\}$ is bad. So it can potentially become empty by firing t_2 sufficiently often. This happens for the initial marking $\boldsymbol{m}_2 = (2p_1, p_4, 2p_5)$, making the net non-live, however keeping it deadlock-free (observe that $\{p_3, p_6\}$ behaves as a trap if the firing of t_3 is blocked forever).

The net has the following minimal siphons $D_1 = \{p_1, p_2\}$, $D_2 = \{p_1, p_3, p_6\}$, and $D_3 = \{p_4, p_5\}$; the first two are bad siphons. With the initial marking $\boldsymbol{m}_2 = (2p_1, p_4, 2p_5)$, D_1 can become empty by firing twice $t_2 \in D_1{}^\bullet \setminus {}^\bullet D_1$, which destroys the liveness, without causing a dead state. The transitions t_4, t_5 are live, the others not. Thus, the net system is not live, but deadlock-free. □

The loss of liveness is not necessarily monotonic itself; i.e., a net may be live for $\boldsymbol{m}_1$, non-live for a marking $\boldsymbol{m}_2$ with $\boldsymbol{m}_2 \geq \boldsymbol{m}_1$, and live again for a marking $\boldsymbol{m}_3$ with $\boldsymbol{m}_3 \geq \boldsymbol{m}_2$ (which works for all examples in this paper). Liveness may also be lost by marking multiples (*homothetic markings*). Examples 1 and 2 are homothetically live, Example 3 in Section 5 not.

4 Monotonic Liveness of Ordinary Nets

Let us turn to liveness criteria suitable for our objective looking at *ordinary* nets first. Liveness criteria not relying on the marking obviously ensure monotonic liveness. Unfortunately, there are only a few.

First of all, there are some structural reduction rules, see, e.g., [Sil85], [Ber86], [Mur89], [SR99]. To give a sample, the following reduction rule is easy to accept: a source transition is live, and all its post-places are unbounded. The transition and its post-places can be deleted (for analysis purposes); the reduction can be iterated as many times it is applicable. Sometimes, this kind of reasoning allows to decide liveness (for examples, see Section 6).

Besides structural reduction we have the DTP, which in most cases does depend on the marking, but it is obviously monotonic w.r.t. the marking: if each siphon contains a marked trap at $\boldsymbol{m}$, then – of course – it contains a marked trap at $\boldsymbol{m}' \geq \boldsymbol{m}$. Thus, the DTP-related conclusions on behavioural properties in Proposition 2 are monotonic as well:

Proposition 5 (Monotonic DTP).

1. *An ordinary net without siphons is monotonically live.*
2. *An ordinary net system which holds the DTP is monotonically deadlock-free.*
3. *A live ES net system which holds the DTP is monotonically live.*
4. *An EFC net system is monotonically live iff the DTP holds.*

Proposition 5.1 can be considered as a special case of the DTP. Then, there must be source transitions (see Proposition 1.8), and the net is not strongly connected and not bounded.

Lemma 1. *Let be $\mathcal{N}$ an ordinary Petri net. If $\mathcal{N}$ is monotonically live, then there are no bad siphons.*

Proof. We will prove its reverse – if there exist a bad siphon, then the net system is not monotonically live – by contradiction. Let P_S be a bad siphon. Then there exist troublemaking transitions $\Theta_i \in P_S{}^\bullet \setminus {}^\bullet P_S$. There must be such transitions, because otherwise $P_S{}^\bullet = {}^\bullet P_S$, and then the siphon P_S would be a trap as well.

Since the net system is monotonically live, the marking of the places $P \setminus P_S$ can be increased in such a way that it will never restrict the firing of the transitions $P_S{}^\bullet$, i.e., the transitions depending on the siphon. Therefore, we can consider the subnet restricted to P_S in isolation.

We will show that the subsystem restricted to P_S can be emptied eventually by increasing the marking, hence cannot be monotonically live. Obviously we can assume that P_S is a minimal siphon. We consider two cases.

(1) The siphon has no forks (t_j is a *fork* if $|t_j{}^\bullet| > 1$). Based on the P-strongly-conectedness (see Proposition 1.3), there exists at least one path from each place $p \in P_S$ to one of the troublemakers Θ_i. Moving a token from p to ${}^\bullet\Theta_i$ does not increase the marking of any other place of P_S not belonging to the considered path. Obviously, this path can contain joins (t_j is a *join* if $|{}^\bullet t| > 1$), but we can add any tokens that are missing in the input places of the join. Firing the join, the marking of the places in the siphon is not increased. Using this process we can move the tokens from any $p \in P_S$ to some ${}^\bullet\Theta_i$, and by firing Θ_i when it is enabled, P_S can be emptied. Thus, the net system can not be monotonically live.

(2) On the contrary, let us assume that there exists at least one fork t_j and let $p_1, p_2 \in t_j{}^\bullet$ be its output places. For the same reason as discussed in case (1), there exists a directed path from both places to one or several troublemakers. If all paths from p_1 to any troublemaker Θ_i contain t_j, then they form a trap. This is impossible because siphons are assumed to be bad. By symmetry, in the case in which the paths from p_2 to troublemakers contain t_j, there exists a trap as well.

Finally, let us assume that there exists a path from p_1 to a troublemaker Θ_i and one path from p_2 to a troublemakers Θ_k, none of them containing t_j.

On both paths the same kind of reasoning can be applied (in an iterative way if several forks appear). Therefore, the siphon can be emptied even if firing t_j increases the tokens in P_S. □

Lemma 1 helps to preclude monotonic liveness for Examples 1 and 2 as well as for all other non-monotonically live examples we are aware of.

Theorem 1. *Let be $\mathcal{N}$ an ordinary Petri net. If $\langle \mathcal{N}, \boldsymbol{m}_0 \rangle$ is monotonically live, then the DTP holds.*

Proof. The structural check of the DTP can have three possible outcomes.

1. If there are no siphons, then the DTP holds trivially and the net is monotonically live (see Proposition 5.1).
2. If there are bad siphons, then the DTP does not hold for any initial marking and the net is not monotonically live (see Lemma 1).
3. If each siphon includes a trap, then the *maximal* trap P_T in every minimal siphon P_S has to be initially marked to fulfill the DTP. Because we assume liveness of the net system, there has to be at least one token in each minimal siphon (see Proposition 1.5). Let us assume that a token is not in P_T, but in a place $p \in P_S \setminus P_T$. If there exists at least one path without forks from p to a troublemaking transition $\Theta_i \in P_S{}^\bullet \setminus {}^\bullet P_S$ not containing any transition belonging to the trap, ${}^\bullet P_T$, then p can be emptied using the same reasoning as used in the proof of Lemma 1, case (1). Therefore the net can not be live. If the path from p to a troublemaking transition $\Theta_i \in P_S{}^\bullet \setminus {}^\bullet P_S$ contains a fork, then the output places of the fork will be marked when p is emptied, and the paths from the output places of the forks to the output should be considered separately.
   Finally, if *all* paths from p to the troublemaking transitions contain at least one transition ${}^\bullet P_T$, then the trap P_T is not maximal since P_T together with all places belonging to the above mentioned paths (including all non-minimal ones) from p to transitions ${}^\bullet P_T$ are also a trap. □

According to Theorem 1, the DTP establishes a necessary condition for monotonic liveness, which complements Proposition 5.3.

Corollary 1. *A live ES net system is monotonically live iff the DTP holds.*

Moreover, for those systems for which deadlock-freeness is equivalent to liveness, the DTP is a sufficient criteria for liveness monotonicity. This leads, for example, to the following theorem:

Theorem 2. *Let be $\mathcal{N}$ an ordinary mono-T-semiflow Petri net which for $\boldsymbol{m}_0$ fulfills the DTP. Then the system $\langle \mathcal{N}, \boldsymbol{m} \rangle$ is live for any $\boldsymbol{m} \geq \boldsymbol{m}_0$.*

Proof. It follows from Proposition 5.2 (DTP and deadlock-freeness monotonicity) and Proposition 3 (equivalence of liveness and deadlock freeness in mono-T-semiflow net systems). □

Therefore, the DTP is a sufficient criterion for monotonic liveness of ordinary mono-T-semiflow net systems as well. In summary, while the DTP is in general neither necessary nor sufficient for liveness, it turns out to be the case to keep alive ordinary ES nets or ordinary mono-T-semiflow nets under any marking increase.

5 Monotonic Liveness of Non-ordinary Nets

It is well-known that non-ordinary nets can be simulated under interleaving semantics by ordinary ones [Sil85] (see Figure 4 for an example). Let us look on the net structures we get by this simulation to learn how far the results for ordinary nets of Section 4 can be uplifted to non-ordinary nets.

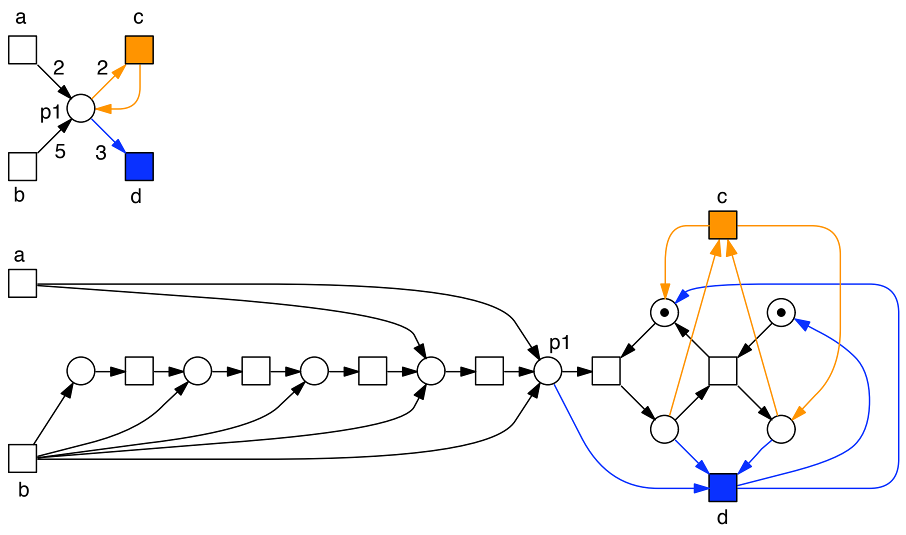


Fig. 4. A general principle to *simulate* a non-ordinary net system by an ordinary net system (here, the firing language of the second net projected on $\{a, b, c, d\}$ is always equal to that of the first) [Sil85].

Example 3. We take a non-ordinary net from [SR02] and consider its simulation by an ordinary net, which we construct according to the general principle demonstrated in Figure 4.

The two net systems in Figure 5 are conservative, consistent, and live for the given initial marking. The ordinary net on the right hand side is not ES, and it has two minimal bad siphons $\{q_1, p_1, p_{1b}, p_{1c}\}$, $\{p_2, p_1, p_{1b}, p_{1c}\}$. Thus, according to Lemma 1, it is not monotonically live. Because our simulation preserves the projection of the firing language, in particular, preserves monotonicity of liveness. Thus, we conclude that the model on the left hand side is not monotonically live. Indeed, both nets are not live for any initial marking with an even number of tokens in p_1, but live for infinitely many other markings greater than or equal to $(1, 1)$. □

As a consequence of firing simulation by the ordinary net systems of the non-ordinary ones (preserving always the markings of the places involved in the head of the tail and complement, here q_1 and q_{1co}), liveness monotonicity can be studied on the ordinary simulation.

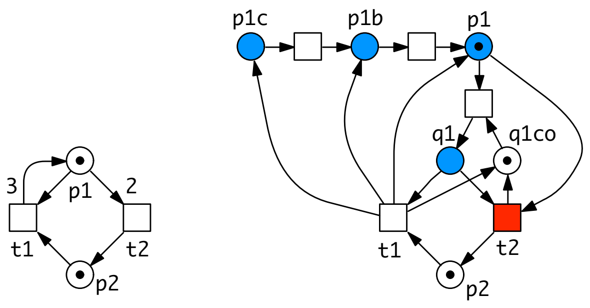


Fig. 5. A non-ordinary Petri net system and its simulation by an ordinary one. Both systems are non-live for any initial marking with an even number of tokens in p_1, and live for any other odd marking. Note that the markings of q_1 and q_{1co} should not be increased in order to keep the language simulation in the right hand model. The net system on the right has a bad siphon $\{q_1, p_1, p_{1b}, p_{1c}\}$ that can potentially become empty by firing t_2 sufficiently often.

6 Applications

We consider a variety of test cases of our benchmark repository to demonstrate the helpfulness of the DTP for biomolecular networks. The following list sketches some basic characteristics. The essential analysis results are summarized in Table 1. All models hold the DTP, they are consistent and (supposed to be) live. For non-ordinary nets, the DTP refers to its simulation by an ordinary one.

1. **Apoptosis** (size: 37 places, 45 transitions, 89 arcs) is a signal transduction network, which governs complex mechanisms to control and execute genetically programmed cell death in mammalian cells. Disturbances in the apoptotic processes may lead to various diseases. This essential part of normal physiology for most metazoan species is not really well understood; thus

there exist many model versions. The validation by Petri net invariants of the model considered here is discussed in [HKW04], [HK04].

2. **RKIP** (size: 11 places, 11 transitions, 14 arcs) models the core of the influence of the Raf-1 Kinase Inhibitor Protein (RKIP) on the Extracellular signal Regulated Kinase (ERK) signalling pathway. It is one of the standard examples used in the systems biology community. It has been introduced in [CSK+03]; the corresponding qualitative, stochastic, continuous Petri nets are scrutinized in [GH06], [HDG10].
3. **Biosensor** (size: 6 places, 10 transitions, 21 arcs) is a gene expression network extended by metabolic activity. The model is a general template of a biosensor, which can be instantiated to be adapted to specic pollutants. It is considered as qualitative, stochastic, and continuous Petri net in [GHR+08] to demonstrate a model-driven design of a self-powering electrochemical biosensor.
4. **Hypoxia** (size: 14 places, 19 transitions, 56 arcs) is one of the well-studied molecular pathways activated under hypoxia condition. It models the Hypoxia Induced Factor (HIF) pathway responsible for regulating oxygen-sensitive gene expression. The version considered here is discussed in [YWS+07]; the corresponding qualitative and continuous Petri nets are used in [HS10] to determine the core network.
5. **Lac operon** (size: 11 places, 17 transitions, 41 arcs) is a classical example of prokaryotic gene regulation. We re-use the simplified model discussed in [Wil06]. Its corresponding stochastic Petri net is considered in [HLGM09].
6. **G/PPP** (size: 26 places, 32 transitions, 76 arcs) is a simplified model of the combined glycolysis (G) and pentose phosphate pathway (PPP) in erythrocytes (red blood cells). It belongs to the classical examples of biochemistry textbooks, see e.g. [BTS02], and thus of systems biology as well. The model was first discussed using Petri net technologies in [Red94]. Its validation by Petri net invariants is shown in [HK04], and a more exhaustive qualitative analysis in [KH08].
7. **MAPK** (size: 22 places, 30 transitions, 90 arcs) models the signalling pathway of the mitogen-activated protein kinase cascade, published in [LBS00]. It is a three-stage double phosphorylation cascade; each phosphorylation/dephosphorylation step applies the mass action kinetics pattern. The corresponding qualitative, stochastic, and continuous Petri net are scrutinized in [GHL07], [HGD08].
8. **CC – Circadian clock** (size: 14 places, 16 transitions, 58 arcs) refers to the central time signals of a roughly 24-hour cycle in living entities. Circadian rhythms are used by a wide range of organisms to anticipate daily changes in the environment. The model published in [BL00] demonstrates that circadian network can oscillate reliably in the presence of stochastic biomolecular noise and when cellular conditions are altered. It is also available as PRISM model on the PRISM website (http://www.prismmodelchecker.org). Its corresponding stochastic Petri net belongs to the benchmark suite used in [SH09]. We consider here a version with inhibitor arcs modelled by co-places.

9. **Halo** (size: 37 places, 38 transitions, 138 arcs) is a cellular signaling and regulation network, describing the phototaxis in the halobacterium salinarum [NMOG03]. It models the sophisticated survival strategy, which the halobacterium developed for harsh conditions (high temperature, high salt). A light sensing system and flagellar motor switching allows the cells to swim to those places of their habitat where the best light conditions are available. The model is the result of prolonged investigations by experimentally working scientists [Mar10].
10. **Pheromone** (size: 42 places, 48 transitions, 119 arcs) is a signal transduction network of the well understood mating pheromone response pathway in *Saccharomyces cerevisiae.* The qualitative Petri net in [SHK06] extends a former ODE model [KK04]. The Petri net was validated by Petri net invariants and a partitioning of the transition set.
11. **Potato** (size: 17 places, 25 transitions, 78 arcs) describes the main carbon metabolism, the sucrose-to-starch breakdown in Solanum tuberosum (potato) tubers. The qualitative Petri net model was developed in cooperation with experimentally working scientists, experienced in ODE modelling. Its validation by Petri net invariants is discussed in [HK04], and a more detailed pathway exploration in [KJH05].

Table 1. Some biomolecular case studies; all of them hold the DTP, are consistent and live.

#	case study	multiplicities	net class	bounded	liveness shown by
1	apoptosis	ORD	ES	no	Proposition 2.1
2	RKIP	ORD	ES	yes	Proposition 2.3
3	biosensor	ORD	ES	no	Proposition 2.3
4	hypoxia	ORD	not ES	no	structural reduction
5	lac operon	HOM	not ES	no	structural reduction
6	G/PPP	HOM	not ES	no	structural reduction
7	MAPK	ORD	not ES	yes	dynamic analysis (RG)
8	CC	HOM	not ES	yes	dynamic analysis (RG)
9	halo	not HOM	not ES	yes	dynamic analysis (RG)
10	pheromone	HOM	not ES	no	by reasoning
11	potato	not HOM	not ES	no	by reasoning

Contrary, the model of signal transduction events involved in the angiogenesis processes, which is discussed in [NMC$^+$09] as a stochastic and continuous Petri net model (size: 39 places, 64 transitions, 185 arcs) is to a large extent covered by a (non-minimal) bad siphon. Thus, even if the net is live for a certain marking $\boldsymbol{m}$, there is always a larger marking $\boldsymbol{m'}$, which will allow to remove all tokens from the bad siphon. Consequently, an arbitrary marking increase will not preserve liveness.

7 Tools

The Petri nets for the case studies have been constructed using Snoopy [RMH10], a tool to design and animate or simulate hierarchical graphs, among them qualitative, stochastic and continuous Petri nets as used in the case studies in Section 6. Snoopy provides export to various analysis tools as well as import and export of the Systems Biology Markup Language (SBML).

The qualitative analyses have been made with the Petri net analysis tool Charlie [Fra09], complemented by the structural reduction rules supported by the Integrated Net Analyser INA [SR99].

8 Conclusions

We have discussed the problem of monotonic liveness, with one of the motivations originating from bio-model engineering. We have presented a new result showing the necessity of the DTP for monotonic liveness.

Moreover, we immediately know – thanks to the well-known propositions of the DTP – that ordinary ES nets are monotonically iff the DTP holds. Furthermore, we know – because the DTP monotonically ensures deadlock freeness – that for any net class, in which liveness and deadlock freeness coincide, monotonic liveness is characterized by the DTP. We have shown one instance for this case: the mono-T-semiflow nets (MTS).

We have demonstrated the usefulness of our results by applying them to a variety of biomolecular networks.

One of the remaining open issues is: what are sufficient conditions for monotonic liveness for more general net structures? While none of our test cases is an MTS net, this line might be worth being explored more carefully, e.g. by looking at FRT nets (Freely Related T-Semiflows) [CS92] and extensions.

Acknowledgements. This work has been partially supported by CICYT - FEDER grant DPI2006-15390 and by the European Community's Seventh Framework Programme under project DISC (Grant Agreement n. INFSO-ICT-224498). The work of M. Heiner was supported in part by the grant DPI2006-15390 for a stay with the Group of Discrete Event Systems Engineering (GISED); the main ideas of this paper have been conceived during this period.

References

Ber86. G. Berthelot. Checking properties of nets using transformations. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 1986.

BL00. N. Barkai and S. Leibler. Biological rhythms: Circadian clocks limited by noise. *Nature*, 403(6767):267–268, 2000.

BRA83. G. W. BRAMS. Reseaux de Petri. Theorie et pratique (2 tomes). *Masson*, 1983.

BTS02. J.M. Berg, J.L. Tymoczko, and L. Stryer. *Biochemistry, 5th ed.* WH Freeman and Company, New York, 2002.

CCS91. J. Campos, G. Chiola, and M. Silva. Ergodicity and Throughput Bounds of Petri Nets with Unique Consistent Firing Count Vector. *IEEE Transactions on Software Engineering*, 17:117–125, 1991.

CS92. J. Campos and M. Silva. Structural techniques and performance bounds of stochastic Petri net models. In *Advances in Petri Nets 1992, volume 609 of Lecture Notes in Computer Science*, pages 352–391. Springer-Verlag, 1992.

CSK$^+$03. K.-H. Cho, S.-Y. Shin, H.-W. Kim, O. Wolkenhauer, B. McFerran, and W. Kolch. Mathematical modeling of the influence of RKIP on the ERK signaling pathway. In *Proc. CMSB*, pages 127–141. Springer, LNCS 2602, 2003.

DA10. R. David and H. Alla. *Discrete, Continuous, and Hybrid Petri Nets.* Springer, 2010.

DHP$^+$93. F. DiCesare, G. Harhalakis, J.M. Proth, M. Silva, and F.B. Vernadat. *Practice of Petri nets in manufacturing.* Chapman & Hall, 1993.

Fra09. A. Franzke. *Charlie 2.0 - a multi-threaded Petri net analyzer.* Diploma Thesis, Brandenburg University of Technology at Cottbus, CS Dep., 2009.

GH06. D. Gilbert and M. Heiner. From Petri nets to differential equations - an integrative approach for biochemical network analysis. In *Proc. ICATPN 2006*, pages 181–200. LNCS 4024, Springer, 2006.

GHL07. D. Gilbert, M. Heiner, and S. Lehrack. A unifying framework for modelling and analysing biochemical pathways using Petri nets. In *Proc. CMSB*, pages 200–216. LNCS/LNBI 4695, Springer, 2007.

GHR$^+$08. D. Gilbert, M. Heiner, S. Rosser, R. Fulton, Xu Gu, and M. Trybiło. A Case Study in Model-driven Synthetic Biology. In *Proc. 2nd IFIP Conference on Biologically Inspired Collaborative Computing (BICC), IFIP WCC 2008, Milano*, pages 163–175, 2008.

HDG10. M. Heiner, R. Donaldson, and D. Gilbert. *Petri Nets for Systems Biology, in Iyengar, M.S. (ed.), Symbolic Systems Biology: Theory and Methods.* Jones and Bartlett Publishers, Inc., in Press, 2010.

HGD08. M. Heiner, D. Gilbert, and R. Donaldson. Petri nets in systems and synthetic biology. In *Schools on Formal Methods (SFM)*, pages 215–264. LNCS 5016, Springer, 2008.

HK04. M. Heiner and I. Koch. Petri Net Based Model Validation in Systems Biology. In *Proc. ICATPN, LNCS 3099*, pages 216–237. Springer, 2004.

HKW04. M. Heiner, I. Koch, and J. Will. Model Validation of Biological Pathways Using Petri Nets - Demonstrated for Apoptosis. *BioSystems*, 75:15–28, 2004.

HLGM09. M. Heiner, S. Lehrack, D. Gilbert, and W. Marwan. Extended Stochastic Petri Nets for Model-Based Design of Wetlab Experiments. *Transactions on Computational Systems Biology XI*, pages 138–163, 2009.

HS10. M. Heiner and K. Sriram. Structural analysis to determine the core of hypoxia response network. *PLoS ONE*, 5(1):e8600, 01 2010.

KH08. I. Koch and M. Heiner. *Petri Nets, in B.H. Junker and F. Schreiber (eds.), Biological Network Analysis*, chapter 7, pages 139–179. Wiley Book Series on Bioinformatics, 2008.

KJH05. I. Koch, B.H. Junker, and M. Heiner. Application of Petri Net Theory for Modeling and Validation of the Sucrose Breakdown Pathway in the Potato Tuber. *Bioinformatics*, 21(7):1219–1226, 2005.

KK04. B. Kofahl and E. Klipp. Modelling the dynamics of the yeast pheromone pathway. *Yeast*, 21(10):831–850, 2004.

LBS00. A. Levchenko, J. Bruck, and P.W. Sternberg. Scaffold proteins may biphasically affect the levels of mitogen-activated protein kinase signaling and reduce its threshold properties. *Proc Natl Acad Sci USA*, 97(11):5818–5823, 2000.

Mar10. W. Marwan. Phototaxis in halo bacterium as a case study for extended stochastic Petri nets. *Private Communication*, 2010.

Mur89. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proc.of the IEEE 77*, 4:541–580, 1989.

NMC$^+$09. L Napione, D Manini, F Cordero, A Horvath, A Picco, M De Pierro, S Pavan, M Sereno, A Veglio, F Bussolino, and G Balbo. On the Use of Stochastic Petri Nets in the Analysis of Signal Transduction Pathways for Angiogenesis Process. In *Proc. CMSB*, pages 281–295. Springer, LNCS/LNBI 5688, 2009.

NMOG03. T. Nutsch, W. Marwan, D. Oesterhelt, and E.D. Gilles. Signal processing and flagellar motor switching during phototaxis of Halobacterium salinarum. *Genome research*, 13(11):2406–2412, 2003.

Red94. V. N. Reddy. Modeling Biological Pathways: A Discrete Event Systems Approch. Master thesis, University of Maryland, 1994.

RMH10. C. Rohr, W. Marwan, and M. Heiner. Snoopy - a unifying Petri net framework to investigate biomolecular networks. *Bioinformatics*, 26(7):974–975, 2010.

RTS98. L. Recalde, E. Teruel, and M. Silva. On linear algebraic techniques for liveness analysis of P/T systems. *Journal of Circuits, Systems, and Computers*, 8(1):223–265, 1998.

RTS99. L. Recalde, E. Teruel, and M. Silva. Autonomous continuous P/T systems. In *Proc. ICATPN*, pages 107–126. Springer, LNCS 1689, 1999.

SH09. M. Schwarick and M. Heiner. CSL model checking of biochemical networks with Interval Decision Diagrams. In *Proc. CMSB*, pages 296–312. Springer, LNCS/LNBI 5688, 2009.

SHK06. A. Sackmann, M. Heiner, and I. Koch. Application of Petri Net Based Analysis Techniques to Signal Transduction Pathways. *BMC Bioinformatics*, 7:482, 2006.

Sil85. M. Silva. *Las Redes de Petri: en la Automática y la Informática.* Editorial AC, 1985.

SR99. P.H. Starke and S. Roch. *INA - The Intergrated Net Analyzer.* Humboldt University Berlin, www.informatik.hu-berlin.de/∼starke/ina.html, 1999.

SR02. M. Silva and L. Recalde. Petri nets and integrality relaxations: A view of continuous Petri net models. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 32(4):314–327, 2002.

Sta90. P.H. Starke. *Analysis of Petri Net Models (in German).* B. G. Teubner, Stuttgart, Stuttgart, 1990.

TS96. E. Teruel and M. Silva. Structure theory of equal conflict systems. *Theoretical Computer Science*, 153:271–300, 1996.

Wil06. D.J. Wilkinson. *Stochastic Modelling for System Biology.* CRC Press, New York, 1st Edition, 2006.

YWS$^+$07. Y. Yu, G. Wang, R. Simha, W. Peng, F. Turano, and C. Zeng. Pathway switching explains the sharp response characteristic of hypoxia response network. *PLoS Comput Biol*, 3(8):e171, 2007.

Modelling Gradients Using Petri Nets

Laura M.F. Bertens[1], Jetty Kleijn[2], Maciej Koutny[3], and Fons J. Verbeek[1]

[1] Imaging and Bioinformatics group, LIACS, Leiden University
Leiden, 2300 RA The Netherlands
lbertens@liacs.nl, fverbeek@liacs.nl
[2] LIACS, Leiden University
Leiden, 2300 RA The Netherlands
kleijn@liacs.nl
[3] School of Computing Science, Newcastle University
Newcastle upon Tyne, United Kingdom
maciej.koutny@ncl.ac.uk

Abstract. Motivated by the graded posteriorization during the AP axis development in the frog *Xenopus laevis*, we propose an abstract Petri net model for the formation of a gradient of proteins in a chain of cells.

Keywords: gradient formation, planar signalling, Petri net model

1 Introduction

Petri nets have been shown to be very promising for molecular and cellular biology, in particular for metabolic, signalling and gene-regulatory networks (see e.g. [1, 2, 6, 9, 10, 14, 20, 30, 31]). In this paper we propose Petri nets as an abstract modelling tool for higher level developmental processes in the organism, e.g., on tissue and organ level, taking cells as central elements.

Currently, we are working on a case study: the embryonic development of the anterior-posterior i.e., head-to-tail axis (AP axis) in the model organism *Xenopus laevis*, the African clawed frog. The development of this model embryo has been studied thoroughly and a huge amount of literature is available to draw from when building and validating the model, see references in [3, 15]. Moreover, this case study comprises several different subprocesses, found in many biological processes, that require modelling solutions. The aim of our project is to eventually model the entire process of AP axis development. Hence we envisage a final model consisting of several building blocks, most of which describing generic biological processes. Each of the subprocesses poses modelling challenges which, when solved, may lead to templates for similar developmental processes, incorporating multiple levels of both spatial and temporal information, also in other organisms. Petri nets are particularly useful in modelling such biological processes, due to their intuitive graphical component, which resembles biological diagrams, and their ability to model concurrency. Our case study appears to be very well suited to explore new ways in which Petri nets can be applied to developmental biology.

In this paper we present a fundamental approach to modelling a particular subprocess: the *formation of a morphogen gradient*, which helps instigate the differentiation of the cells along the developing axis. This subprocess is a good starting point, since it is relatively simple conceptually, in comparison to the other subprocesses in the case study. In early development, gradients are crucial ([36]) and finding a modelling solution for the generic process of gradient formation will not only serve the modelling of this case study, but will also be useful for the modelling of other developmental processes. By staying very close to the biological sequence of events in gradient formation, rather than focusing on a concrete outcome, the model should be generally applicable and robust.

Throughout this paper the emphasis will be on abstraction and modelling decisions, as opposed to implementation of specific biological data; we present a basic Petri net modelling gradient formation, which serves as a proof of concept for our approach. In the remainder of this paper we outline the biological background of gradient formation in general and in this particular case study. Subsequently we describe our modelling decisions and we present the model. In the last section the possibilities of the model and future work are discussed.

2 PT-nets with activator arcs

For a general introduction to Petri nets we refer to [27]. In this paper, we use PT-nets with activator arcs ([17]), and a maximally concurrent execution rule [5].

Petri nets are defined by an underlying structure consisting of *places* and *transitions*. These basic elements are connected by directed, *weighted arcs*. In the Petri net model considered in this paper, there are moreover *activator arcs* connecting places to transitions. In modelling, places are usually the passive elements, representing local states, and transitions the active elements. Here, global states, referred to as *markings*, are defined as mappings assigning to each place a natural number (of *tokens* corresponding to available resources).

A *PTA-net*, is a tuple $N = (P, T, W, Act, m_0)$ such that:

- P and T are finite disjoint sets, of the *places* and *transitions* of N, resp.
- $W : (T \times P) \cup (P \times T) \to \mathbb{N}$ is the *weight function* of N.
- $Act \subseteq P \times T$ is the set of *activator* arcs of N.
- $m_0 : P \to \mathbb{N}$ is the *initial marking* of N.

In diagrams, places are drawn as circles, and transitions as boxes. Activator arcs are indicated by black-dot arrowheads. If $W(x, y) \geq 1$, then (x, y) is an *arc* leading from x to y; it is annotated with its weight if this is greater than one. A marking m is represented by drawing in each place p exactly $m(p)$ tokens as small black dots. We assume that each transition t has at least one input place (there is at least one place p such that $W(p, t) \geq 1$).

When a single transition t occurs ('fires') at a marking, it takes tokens from its input places and adds tokens to its output places (with the number of tokens consumed/produced given by the weights of the relevant arcs). Moreover, if there is an activator arc $(p,t) \in Act$, then transition t can only be executed at the given marking if p contains at least one token, without the implication of tokens in p being consumed or produced when t occurs. Thus, the difference with a *self-loop*, i.e., an arc from p to t and vice versa, is that the activator arc only tests for the presence of tokens in p.

We define the executions of N in the more general terms of simultaneously occurring transitions. A *step* is a multiset of transitions $U : T \rightarrow \mathbb{N}$. Thus $U(t)$ specifies how many times transition t occurs in U. (Note that if we exclude the empty multiset, single transitions can be considered as minimal steps.) Step U is *enabled* (to occur) at a marking m if m assigns enough tokens to each place for all occurrences of transitions in U and, moreover, all places tested through an activator arc by a transition in U, contain at least one token.

Formally, step U is enabled at marking m of N if, for all $p \in P$:

– $m(p) \geq \sum_{t \in T} U(t) \cdot W(p,t)$
– $m(p) \geq 1$ whenever there is a transition t such that $U(t) \geq 1$ and $(p,t) \in Act$.

If U is enabled at m, it can be *executed* leading to the marking m' obtained from m throught the accumulated effect of all transition occurrences in U:

– $m'(p) = \sum_{t \in T} U(t) \cdot (W(t,p) - W(p,t))$ for all $p \in P$.

Finally, a step U is said to be *max-enabled* at m if it is enabled at m and there is no step U' that strictly contains U (meaning that $U' \neq U$ and $U(t) \leq U'(t)$ for all transitions t) and which is also enabled at m. We denote this by $m[U\rangle m'$. A (max-enabled) *step sequence* is then a sequence $\sigma = U_1 \dots U_n$ of non-empty steps U_i such that $m_0 \, [U_1\rangle \, m_1 \, \cdots \, m_{n-1} \, [U_n\rangle \, m_n$, for some markings $m_1, \dots, m_n$ of N. Then m_n is said to be a *reachable* marking of N (under the maximally concurrent step semantics).

To conclude this preliminary section, we elaborate a bit on the choice of this particular net model. First, it should be observed that it follows from the above definitions that the semantics allows *auto-concurrency*, the phenomenon that a transition may be executed concurrently with itself. This approach makes it possible to use transitions for a faithful modeling of natural events like the independent (non-sequential) occurrence in vast numbers of a biochemical reaction in a living cell. Note that the degree of auto-concurrency of a transition can easily be controlled by a dedicated place with a fixed, say k, number of tokens connected by a self-loop with that transition implying that never more than k copies of that transition can fire simultaneously.

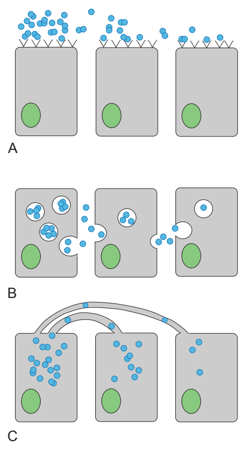
Activator arcs were introduced in [16] as a means of *testing* for the presence of at least one token in a place, and so they are similar to other kinds of net features designed for the same reason. We mentioned already self-loops by which the presence of a token in a place can be tested only by a single transition (which 'takes and returns' the token) and not simultaneously by an arbitrary number of transition occurrences in a step. Two other mechanisms which do allow such multiple testing are *context* arcs [25] and *read (or test)* arcs [34]. Both, however, display important differences when compared with activator arcs. A context arc testing for the presence of a token in place p by transition t indicates that after a step in which t participates has been executed, p must still contain a token which precludes the occurrence in the same step of transitions that have p as an output place. A read arc is also different, but less demanding in that there must exist a way to execute sequentially (i.e., one-by-one) all transition occurrences in the step, without violating the read arc specification. In both cases, one can easily see that activator arcs are most permissive since they only check for the presence of a token *before* the step is executed (this is often referred to as *a priori* testing). We feel that *a priori* testing is more appropriate for biological applications as the 'lookahead' implied by the other two kinds of test arcs is hard to imagine in reality.

Finally, we rely in this paper on *maximal* concurrency in the steps that are executed which reflects the idea that execution of transitions is never delayed. This may also be viewed as a version of time-dependent Petri nets where all transitions have a firing duration of 1. However, the maximal concurrency we apply in this paper does not derive from Petri nets with time, but rather from Petri nets with *localities* [19] leading to a *locally maximal* semantics. This semantics is what we plan to use to model other aspects of the development as well. Here one may think of e.g., the locally synchronous occurrence (in pulses) of reactions in individual compartments of a cell.

3 Biological background and modelling decisions

In biology, the term gradient is used to describe a gradual and directed change in concentration of a morphogen through a group of cells, e.g., a tissue. Morphogens are signalling molecules that cause cells in different places in the body to adopt different fates and thereby help establish embryonic axes. Morphogens are produced in a localized source of a tissue, the source cell(s), and emanate from this region, forming a concentration gradient ([13, 32]). A morphogen gradient has an immediate effect on the differentiation of the cells along it; cells are able to 'read' their position along the gradient and determine their developmental fate accordingly. They have a range of possible responses and the morphogen concentration dictates which response will be exhibited ([13, 32]). In establishing their developmental fate, cells take into account the morphogen concentration. When the morphogen concentration over the entire gradient is increased (or decreased), the cells should accordingly change their response to that corresponding to a higher (or lower) level of morphogen.

The mechanisms by which the morphogen travels through a cell layer have been the topic of some debate and are not yet fully understood. Three mechanisms have been described, shown schematically in Figure 1: (A) diffusion through the extracellular matrix ([8, 11, 23]), either passively, like a drop of ink in water ([11]), or facilitated by receptors on the cell surface which guide the morphogens along ([8]), as shown in the figure; (B) sequential internalization of the morphogen molecules in vesicles in the cells, a process called endocytosis, and subsequent re-emission ([7, 8, 32]); (C) direct contact between the cells by



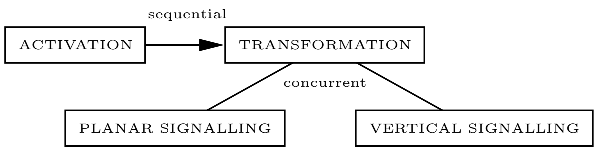


Fig. 1. Left: three possible mechanisms for gradient formation: diffusion (A), endocytosis and subsequent re-emission (B) and transport through cytonemes (C). Right: an overview of the process of AP axis formation in *Xenopus laevis*

means of tentacle-like threads of cytoplasm, called cytonemes, connecting the cells ([13]). These mechanisms are not necessarily mutually exclusive and some studies conclude that a combination of mechanisms underlies the formation of a gradient. It is important to note that both diffusion and endocytosis take place between neighbouring cells, while cytonemes connect all cells directly to the source. This makes it very different from a modelling perspective, as will be discussed below.

Unfortunately, knowledge of the exact concentrations and shapes of most gradients is limited. This is mainly due to the transient nature of morphogen gradients and the low concentrations at which they are effective, both of which make it difficult to visualize the morphogens ([11]). Many morphogens are rapidly degraded or prevented from binding to receptors by antagonistic proteins ([11]). Much of the information on gradients is therefore obtained indirectly, by observing their effect, i.e., the responses of the cells along it ([11]).

AP axis formation in *Xenopus laevis*: a case study. The AP axis formation in *Xenopus laevis* takes place during the early embryonic stage of gastrulation and ensures the development of anterior structures near the head and posterior structures towards the tail. The process can be seen as divided into two steps, which take place sequentially ([26]), cf. Figure 1. The first step is activation; a group of cells in the outer cell layer of the embryo, the ectoderm, change their developmental identity and form a rectangular strip of tissue, called the neurectoderm ([3, 15]). It is this strip of cells in which the AP axis will ultimately be established, leading to gradual posteriorization of cells nearer to the tail-end of the embryo.

During the second step, transformation, the axis is formed in the neurectoderm by means of two mechanisms: vertical and planar signalling between neighbouring cells ([3, 15]). Here we focus on the second. Planar signalling occurs within the neurectoderm in a direction parallel to the future axis (and is therefore called 'planar'). Concentration gradients of several morphogens are formed in the neurectoderm along the future AP axis. Source cells on the posterior end of the neurectoderm produce the morphogens, which then get distributed throughout the tissue. Individual cells sense their position along these gradients and take on a more or less posterior fate according to the concentration of these posteriorizing molecules ([12, 15]), thereby establishing the formation of an AP axis. In the planar signalling of our case study three types of signalling molecules play a role: retinoic acid (RA), fibroblast growth factors (Fgf) and Wnts ([22, 29, 35]). All of these are produced at the posterior end of the embryo and together these promote posterior cell fates, while inhibiting anterior fates. Although it is clear that all three types are important in axis formation, it is not yet fully understood how these proteins interact in establishing cell fate. A general and abstract modelling approach, focusing on the underlying common process of gradient formation, makes it possible to later add specific data on any of the morphogens in particular or on combinations of these.

Modelling decisions. We have chosen cells as the elementary units in our model to be represented as places in a Petri net. Earlier studies ([4, 21, 24]) have successfully modelled cell-to-cell signalling, starting from a lower biological level, using places to represent genes and proteins. Although this allows a high level of detail, it also complicates the net and makes it difficult to identify single cells. In our approach the cellular level represents the intermediate level between the subcellular levels, on which the morphogen signalling between cells takes place, and the tissue/organ level, where whole cell layers may move.

Tokens are used to represent a certain level of concentration (see [10]) within the overall gradient of the morphogen system, without differentiation between morphogens or their quantities. As mentioned before, in most cases no quantitative data are available, since morphogen gradients are often transient and difficult to visualize ([13]). As in [21], our approach is therefore partially qualitative and partially quantitative. The significance of tokens in a place is not purely qualitative; not only the presence or absence but also the exact number of tokens determine the course of events. However, the numbers of tokens do not represent actual numbers of molecules, making the model semi-qualitative. Our Petri net model can, however, also be used to model specific morphogens, incorporating quantitative data, by assigning exact concentrations to the tokens and thereby making the model completely quantitative. For certain morphogens quantitative data exist, for instance for the gradient of Fgf8 in zebrafish, which can be seen to spread extracellularly through the processes of diffusion, endocytosis and degradation ([28]). Also, for some gradients found in biological processes, experimental data have enabled to deduce mathematical expressions, describing the quantitative morphogen concentrations ([33]). When modelling these gradients, these formulas can be incorporated in the parameters of our Petri net model.

Neighbourhood communication. It is our aim to develop a faithful model for gradient formation. Rather than having the net simply distribute the proper pre-computed amount of tokens over the places representing the cells, the actual transport of morphogen between cells can be read off from the Petri net model during execution. Consequently, when building the model we have to specify explicitly which process of gradient formation is to be modelled. Here we choose to model morphogens moving between neighbouring cells, i.e., the Petri net will implement a mechanism similar to diffusion or endocytosis and subsequent re-emission, but not transport through cytonemes (since this does not take place between neighbouring cells). However, we foresee no problems in the abstract implementation of the latter process. The difference between diffusion and endocytosis is apparent on a lower biological level and could be modeled by subnets. Furthermore, often the ratio of the concentrations between neighbouring cells is not known due to lack of quantitative data, and it may vary depending on the gradient considered. Therefore we have a parameter ρ in our model to represent this ratio and to determine the amount of tokens to be transported between places during the simulation of gradient formation. Since we do not distinguish the molecular mechanisms of diffusion, endocytosis and degradation of morphogens in this model, ρ represents the final ratio of morphogens between neighbouring cells and morphogen degradation is implicit. To model explicitly both the production of morphogens in the local source cells and the degradation in the target cells, subnets could be added. This should make the source and sink mechanisms of gradient formation transparent and allow the user to experiment with different configurations.

Implementation. In the organism, gradient ratios arise passively as a consequence of physical laws. However, to accurately reflect the biological process of gradient formation underlying the spread of morphogens from cell to cell, our formal model has to compute the number of tokens passed on based on the ratio ρ. Hence the model includes explicit separate computational units for the necessary calculations. In particular, these parts of the net control the transport of tokens between places. In this way a close relation to the biological process can be maintained in one part of the net, with the underlying computations performed by a subnet in the background. At all times, the marking of the net will be consistent with biological observations of (the effect of) the gradient, i.e., the ratio is maintained and places corresponding to cells further away from the source will never have more tokens than places (cells) closer to it. Another important feature of the model is the use of concurrent steps rather than individually occurring transitions. Cells only react to their environment and have no knowledge of other cells than their immediate neighbours. Non-adjacent cells can be simultaneously involved in the transport of morphogens. This leads to an execution mode consisting of concurrent *steps*. Moreover, these steps are *maxi-*

mal to reflect that also in the net model morphogens are moved to a next cell as soon as possible.

4 Gradients and Petri Nets

Following the ideas outlined in the previous section, we will propose a formal model for the formation of a gradient.

Our assumptions regarding the biological process of gradient formation are as follows. Given is a segment of k adjacent cells with the i-th cell immediate neighbour of the $(i+1)$-th cell. Morphogens can be transported only between immediate neighbours. Morphogens move from cells with higher concentration to neighbours with lower concentration, as long as their concentration ratio does not exceed a given gradient ratio $0 < \rho < 1$. We assume that ρ is a rational number, i.e., $\rho = \frac{N}{M}$, where $M > N \geq 1$. Initially, the first cell x_1 contains a quantity (has concentration level) K of a morphogen. These assumptions lead to the following modelling problem.

Given are $k \geq 1$ places $x_1, \ldots, x_k$, representing a segment of k cells with place x_i corresponding to the i-th cell. In the initial marking m_0, the first place x_1 contains K tokens and there are no tokens in the other places.
In the net modelling the mechanism of gradient formation, we need to shift tokens from x_1 in the direction of the last place x_k. Places and/or transitions may be added, but in such a way that for any reachable marking m the following hold.

1. The number of tokens in the x_i's remains constant, i.e.,

   $m(x_1) + \cdots + m(x_k) = K$ **token preservation**

2. The tokens are distributed monotonically along the sequence of k places, i.e.,

   $m(x_1) \geq \ldots \geq m(x_k)$ **monotonicity**

3. The ratio of the numbers of tokens in two neighbouring places does not exceed ρ, i.e., for every $1 \leq i < k$ with $m(x_i) \geq 1$:

   $\frac{m(x_{i+1})}{m(x_i)} \leq \rho$ **ratio**

4. Shifting continues until moving even one token would violate the above, i.e., if no tokens are shifted after marking m was reached, then for every $1 \leq i < k$ with $m(x_i) > 1$:

   $\frac{m(x_{i+1})+1}{m(x_i)-1} > \rho$ **termination**

Moreover, the relative position of a place within the sequence plays no role. In particular, the mechanism should be easily scalable and **insensitive** to the specific values of k and K. □

If we look at the above formulation of properties (2) and (3) — monotonicity and preservation of the gradient ratio — and recall that $\rho = \frac{N}{M}$ and $M > N$, it is easy to observe that these two properties are together equivalent to stating that, for every $1 \le i < k$, $N \cdot m(x_i) - M \cdot m(x_{i+1}) \ge 0$. We will call a marking m satisfying this inequality *consistent* and denote $\alpha_i \overset{\mathrm{df}}{=} N \cdot m(x_i) - M \cdot m(x_{i+1})$, for every $1 \le i < k$. Note that the initial marking is consistent.

Similarly, if we look at the above formulation of properties (2) and (4) — monotonicity and termination — it is easy to observe that together they are equivalent to the statement that, for every $1 \le i < k$, $N \cdot m(x_i) - M \cdot m(x_{i+1}) < M + N$. We will call a consistent marking m satisfying this inequality *stable.* Note that for a given ρ, k and K, there may be more than one stable marking. For example, if $\rho = \frac{1}{2}$, $k = 5$ and $K = 111$, then the following are two different stable markings:

x_1	x_2	x_3	x_4	x_5
59	29	14	6	3

x_1	x_2	x_3	x_4	x_5
58	29	14	7	3

We are now ready to propose a generic solution for the above problem.
For a given consistent marking m and each $1 \le i < k$, move β_i tokens from x_i to x_{i+1} where $\beta_i \le \left\lfloor \frac{\alpha_i}{M+N} \right\rfloor$, and at least one β_i must be non-zero if at least one of the values $\left\lfloor \frac{\alpha_i}{M+N} \right\rfloor$ is non-zero. We denote the resulting marking by $m_{\beta_1 \ldots \beta_{k-1}}$.

An intuitive reason for proposing such a mechanism for shifting tokens is that the number of tokens in x_i that are 'balanced' by tokens in x_{i+1} is $\frac{M}{N} \cdot m(x_{i+1})$, because each token in x_{i+1} is equivalent to $\frac{M}{N}$ tokens in x_i. Hence there are $m(x_i) - \frac{M}{N} \cdot m(x_{i+1})$ unbalanced tokens in x_i. The 'portion' of each unbalanced token that could be safely transferred to x_{i+1} is $\frac{N}{M+N}$. Hence in total we may safely transfer $\left\lfloor \frac{N}{M+N} \cdot (m(x_i) - \frac{M}{N} \cdot m(x_{i+1})) \right\rfloor$ tokens, which is precisely $\left\lfloor \frac{\alpha_i}{M+N} \right\rfloor$ tokens. Clearly, some of the numbers $\beta_1, \ldots, \beta_{k-1}$ can be zero, and by the condition above, all β_i's are zeros if and only if the marking is stable:

Proposition 1. $\beta_1 = \cdots = \beta_{k-1} = 0$ *if and only if m is stable.*

Crucially, by the mechanism proposed consistent markings are always transformed into consistent markings.

Proposition 2. *If m is a consistent marking then $m_{\beta_1 \ldots \beta_{k-1}}$ is also consistent.*

According to the above, any number of tokens not exceeding $\left\lfloor \frac{\alpha_i}{M+N} \right\rfloor$ can be moved *simultaneously* from x_i to x_{i+1} (for every $i < k$), and consistency will be preserved. Clearly, the new consistent marking is different from the previous one if and only if, for at least one i, we have $\beta_i \ge 1$. The idea now is to keep changing the marking on $x_1, \ldots, x_k$ until a marking m has been reached such that $\left\lfloor \frac{\alpha_i}{M+N} \right\rfloor = 0$, for all $1 \le i < k$, which is equivalent to $\alpha_i < M + N$, for all $1 \le i < k$. In other words, this m is a stable marking. Since tokens cannot be shifted forever, this procedure will always terminate in a stable marking (formally, we can show this by considering a weighted distance to the end of the chain of the K tokens; it never increases and always decreases in a non-stable state).

Looking now from the point of view of a Petri net implementation of the proposed mechanism, what we are after is a net N_{shift} comprising the places $x_1, \ldots, x_k$ and such that if m is a marking of N_{shift} whose projection on these k places is consistent, then a step U can occur at m if

- it moves at most $\left\lfloor \frac{\alpha_i}{M+N} \right\rfloor$ tokens from x_i to x_{i+1}, for all $1 \leq i < k$;
- at least one token is moved from x_i to x_{i+1} for at least one $1 \leq i < k$, unless the projection of m onto $x_1, \ldots, x_k$ is stable.

In fact, in the proposed implementation, we will be preceding the 'token-shifting' with a 'pre-processing' stage which seems to be unavoidable unless one uses some kind of arcs with complex weights depending on the current net marking.

Implementation. In the implementation of the proposed shifting mechanism, as many tokens as possible should be shifted from one neighbour to the next. That means that, at each stage we have, for every $1 \leq i < k$, $\beta_i = \left\lfloor \frac{\alpha_i}{M+N} \right\rfloor$. Moreover, tokens are shifted from a place without any assumptions whether new tokens will come to that place from its other neighbour. Thus we need to provide a Petri net structure capable of 'calculating' the value of expressions like

$$\left\lfloor \frac{N \cdot m(x_i) - M \cdot m(x_{i+1})}{M+N} \right\rfloor .$$

Our proposed gradient forming mechanism distinguishes three phases: I, II and III. An auxiliary net N_{3phase}, shown in Figure 2(b), is used to schedule the transitions implementing the calculations. It controls these transitions via the places w^I and w^{II} and activator arcs. For the full picture of the system one should combine the figures for all pairs (x_i, x_{i+1}) with a single copy of the net in Figure 2(b). Note that all places with identical label (in particular w^I, w^{II}, and w^{III}) should be identified. That other parts of the encompassing net model do not interfere with the calculations carried out during phases I and II can be ensured by connecting the relevant transitions with the place w^{III} using activator arcs.

For every $1 \leq i < k$, transition t_i is intended to shift tokens from x_i to x_{i+1} (phase III). To achieve this, we use two disjoint sets of new, auxiliary places, $x'_1, \ldots, x'_k$ and $x''_1, \ldots, x''_k$. These places are initially empty. The idea is to fill x'_i with $N \cdot m(x_i)$ tokens and x''_{i+1} with $M \cdot m(x_{i+1})$ tokens (phase I). The latter are used for the removal of $M \cdot m(x_{i+1})$ tokens from x'_i (phase II). After this, there are α_i tokens remaining in x'_i. Finally, for each group of $N + M$ tokens in x'_i, one token is shifted from x_i to x_{i+1}. The construction (for x_i and x_{i+1}) is shown in Figure 2(a).

The overall mechanism operates in cycles of three consecutive, maximally concurrent steps such that for every $1 \leq i < k$:

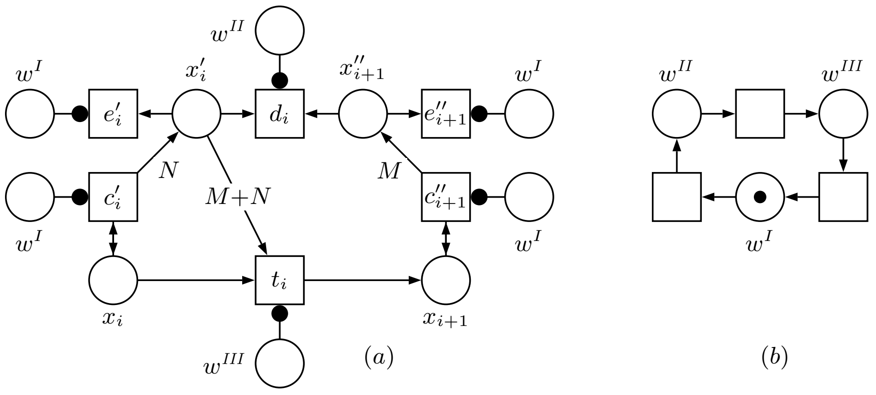


Fig. 2. (a) The main part of the construction for the solution (note that e_{i+1}'' is introduced for later use when one might want to remove or add tokens to the x_i's from 'outside'; in the standard (consistent) situation it is never activated as after phase 2, place x_{i+1}'' is empty.); and (b) the subnet N_{3phase} enforcing the three phases.

I. Transition c_i', inserts (in $m(x_i)$ auto-concurrent occurences) $N{\cdot}m(x_i)$ tokens into x_i'. In the same step, transition c_{i+1}'', inserts (in $m(x_i)$ auto-concurrent occurences) $M{\cdot}m(x_{i+1})$ tokens into x_{i+1}''. Simultaneously, transitions e_i' and e_{i+1}'' empty x_i' and x_{i+1}'' of any residual tokens left from the previous cycle.

II. Next, transition d_i (in $M{\cdot}m(x_{i+1})$ auto-concurrent occurences) empties x_{i+1}'' and leaves in x_i' the difference $\alpha_i = N{\cdot}m(x_i) - M{\cdot}m(x_{i+1})$.

III. In the third step, the occurrences of transition t_i transfer $\beta_i = \left\lfloor \frac{\alpha_i}{M+N} \right\rfloor$ tokens from x_i to x_{i+1}.

Proposition 3. *Each cycle results in transferring β_i tokens from x_i to x_{i+1}.*

Note that in this implementation with the control net N_{3phase}, neighbouring pairs are either all involved in calculations (step 1 and 2 of the cycle) or tokens are transferred between neighbours (step 3). During the whole operation of the adjustment process (except for the transfer phase), the token counts in the places x_i representing the cells are unchanged and they can be accessed for reading by other transitions (and thus influence neighbouring cells). In other words, calculations are *orthogonal* to the basic operation of the net (the gradient formation).

As an example, let us consider the case when $\rho = \frac{1}{2}$, $k = 4$ and $K = 100$. Then executing the constructed net in a maximally concurrent manner leads to the following sequence of markings on the x_i after each cycle and eventually to a stable marking:

x_1	100	67	67	60	60	57	57	56	56	55	54
x_2	0	33	22	29	25	28	26	27	26	25	26
x_3	0	0	11	8	12	10	12	12	13	12	12
x_4	0	0	0	3	3	5	5	5	5	6	6

The next example shows what happens if we start from a (non-initial) consistent marking (again $\rho = \frac{1}{2}$):

x_1	200	167	156	...
x_2	50	67	67	...
x_3	0	16	22	...
x_4	0	0	5	...

The construction works without any problems, if we start with a consistent marking. In case $0 > \alpha_i$ for some i, then transition t_i is not executed, but the transitions t_{i-1} and t_{i+1} may still be executed and lead to an adjustment of the marking causing t_i to become active in the next cycle. A further observation is that adding (or removing) tokens at some point, will trigger a re-adjustment process which tries to re-establish the correct ratios between the markings of adjacent places x_i. This process is unpredictable, but to deal with that case we have included transition e''_{i+1} which in the standard (consistent) situation is never activated since then, after phase 2, place x''_{i+1} is empty.

An important characteristics of the proposed solution is that it is purely *local* and does not assume anything about the number of tokens which may appear in the x_i's nor the length of the chain. In other words, it is truly generic. What's more it also works if M and N are different for different pairs of neighbouring places, i.e., if rather than a uniform gradient ratio ρ there is a ratio ρ_i for each pair of neighbours x_i and x_{i+1}.

Another feature of our solution is the maximal concurrency semantics intended to reflect the idea of morphogens (simultaneously) moving from cell to neighbouring cell whenever that is possible. The preliminary sequential semantics model we developed (but not reproduced here) is more complicated as it also needs *inhibitor arcs* which test for *absence* of tokens (to decide whether or not tokens should still be shifted). Moreover, one needs to decide that x_i either receives or sends tokens at each stage. In a step model it can both receive and send. Also, with the maximal concurrency semantics, the number of states of the model is dramatically reduced.
The auxiliary net N_{3phase} is used to partly sequentialize the behaviour in order to separate the pre-processing phases from the actual shifting phase. This net could also have been made local to the subnet in Figure 2(a), with different copies of it assigned to different localities. This would have given the additional possibility of controlling the degree of synchronisation between different parts of the gradient model by using a locally maximal step semantics.

Finally, we would like to point out that the activator arcs in our implementation are used only to control the calculation and can actually be avoided in case there would be a limit on the number of tokens in each place x_i at any time. (Then the activator arcs can be eliminated basically by having separate copies of N_{3phase} for each $1 \leq i < k$, transfer around sufficiently many tokens in a bundle, and replace activator arcs by self-loops). This assumption corresponds to having (or knowing) some capacity bound on the concentration levels of morphogens in a cell and so may be biologically sound!

5 Conclusion

Starting from gradient formation in the AP axis development in the model organism *Xenopus laevis*, we have presented a novel approach to using Petri nets in developmental biology by focusing on the cellular rather than subcellular levels and abstracting from concrete proteins and genes. This has led to a parameterized Petri net model for the general process of gradient formation through diffusion and endocytosis.

Assumptions regarding gradient formation have been formulated based on essential features of this process as reported in the literature. These assumptions underlie the precise requirements given that should be satisfied by an abstract Petri net model of gradient formation. A crucial point here is the consistency that is maintained during the execution of the model. Hence the realization of the gradient is faithfully reflected. Moreover the close relationship between biological process and evolution of the formal model makes it possible to apply existing Petri net techniques to analyse what happens during gradient formation. In particular cause-effect relations should be properly reflected in the process semantics of the modelling Petri net ([17, 18]).

Another main contribution of this approach is its generic nature, leading to a model that is scalable and applicable to a plethora of specific gradients. Also scalability is a consequence of the faithful reflection of the biological process. Since the final token (morphogen) distribution is not directly computed from the initial amount of morphogen and the length of the chain of cells, but rather simulates the communication between neighbouring cells, the length plays no role in the occurrence of the steps. The model as presented here represents a one-morphogen system without relying on quantitative data, but exact values could be assigned to ratio and individual tokens. Moreover, it provides a basis for simulation of simultaneous gradient formation (different morphogens with different experimental initial markings) and for inhibiting/activating interactions between them. Simulation with actual biological data to validate the model should be a next step. In addition, we will focus our attention on the extension of this still rather basic model to more dimensions, e.g., rather than having just a single line of cells, we consider the spread of morphogens from a source throughout a tissue plane or volume.

In [4, 21], Petri nets are used to model developmental processes in a way similar to our approach when it comes to the semi-qualitative use of tokens and the use of maximal concurrency. In these papers however, the focus is on subcellular levels. Petri net places are used to represent genes and gene products, where in our approach cells, as basic units in a tissue, are modelled by places. Having cells as basic units should prove to be a useful intermediate position convenient for 'zooming in and out' between subcellular and tissue level. It is our aim to model more subprocesses of the AP axis formation. For instance the different molecular processes underlying diffusion and endocytosis could be modeled in subnets, allowing the user to compare the different effects of these mechanisms. Also the degradation of morphogens could be modeled by a subnet, making the entire process more explicit. The choice of cells as main elements is expected to be particularly suitable not only in this 'vertical' linking processes, but also for the 'horizontal' connections between processes taking place on the same cellular level. Having molecules or genes as places would result in specific net models for certain processes, as would taking tissue structures, such as the neurectoderm. Cells however can play a role in different processes simultaneously.

The next step in the modelling of the AP axis development in *Xenopus laevis* will focus on the vertical signalling (see Figure 1). This process occurs concurrently with the planar signalling of gradient formation and involves the same cells. This will challenge us to explore further the possibilities of Petri nets as a model for concurrent and independent processes in high level developmental biology.

Acknowledgments We are grateful to all referees for their remarks and suggestions which helped us to improve the paper. We would like to express also our appreciation to A.Durston and H.Jansen for discussing *Xenopus* biology.

References

1. R.Banks: Qualitatively Modelling Genetic Regulatory Networks: Petri Net Techniques and Tools. Ph.D. Thesis, Newcastle University, 2009
2. R.Banks, V.Khomenko, L.J.Steggles: A Case for Using Signal Transition Graphs for Analysing and Refining Genetic Networks. ENTCS 227, 2009, 3–19
3. N.Bardine: Hox in Frogs. PhD Thesis, University of Leiden, 2008
4. N.Bonzanni et. al: Executing Multicellular Differentiation: Quantitative Predictive Modelling of *C.elegans* Vulval Development. Bioinformatics 25, 2009
5. H.-D.Burkhard: On Priorities of Parallelism: Petri Nets under the Maximum Firing Strategy. LNCS 146, 1983, 86–97
6. C.Chaouiya: Petri Net Modelling of Biological Networks. Briefings in Bioinformatics 8, 2007, 210–219.
7. E.Entchev, M.Gonzalez-Gaitan: Morphogen Gradient Formation and Vesicular Trafficking. Traffic 3, 2002, 98–109
8. J.A.Fischer, S.H.Eun, B.T.Doolan: Endoytosis, Endosome Trafficking, and the Regulation of *Drosophila* Development. The Annual Review of Cell and Developmental Biology 22, 2006, 181–206
9. D.Gilbert, M.Heiner: From Petri Nets to Differential Equations - an Integrative Approach for Biochemical Network Analysis. LNCS 4024, 2006, 181–200
10. D.Gilbert, M.Heiner, S.Lehrack: A Unifying Framework for Modelling and Analysing Biochemical Pathways Using Petri Nets. LNBI 4695, 2007, 200–216
11. J.B.Gurdon et. al: Activin Signalling and Response to a Morphogen Gradient. Nature 371, 1994, 487–492
12. J.B.Gurdon et. al: Single Cells Can Sense their Position in a Morphogen Gradient. Development 126, 1999, 5309–5317
13. J.B.Gurdon, P.Bourillot: Morphogen Gradient Interpretation. Nature 413, 2001, 797–803
14. M.Heiner, D.Gilbert, R.Donaldson: Petri Nets for Systems and Synthetic Biology. LNCS 5016, 2008, 215–264
15. H.J.Jansen: Anterior-posterior Axis Formation in *Xenopus laevis*. PhD Thesis, University of Leiden, 2009

16. R.Janicki, M.Koutny: Semantics of Inhibitor Nets. Information and Computation 12, 1995, 1–16
17. J.Kleijn and M.Koutny: Processes of Petri Nets with Range Testing. Fundamenta Informaticae 80, 2007, 199–219
18. J.Kleijn and M.Koutny: Processes of Membrane Systems with Promoters and Inhibitors. Theoretical Computer Science 404, 2008, 112–126
19. J.Kleijn, M.Koutny, G.Rozenberg: Towards a Petri Net Semantics for Membrane Systems. LNCS 3850, 2006, 292–309
20. I.Koch, B.H.Junker, M.Heiner: Application of Petri Net Theory for Modelling and Validation of the Sucrose Breakdown Pathway in the Potato Tuber. Bioinformatics 21, 2004, 1219–1226
21. E.Krepska et. al: Design Issues for Qualitative Modelling of Biological Cells with Petri Nets. LNBI 5054, 2008, 48–62
22. T.Kudoh, S.W.Wilson, I.B.Dawid: Distinct roles for Fgf, Wnt and Retinoic Acid in Posteriorizing the Neural Ectoderm. Development 129, 2002, 4335–4346
23. A.D.Lander, Q.Nie, F.Y.M.Wan: Do Morphogen Gradients Arise by Diffusion? Developmental cell 2, 2002, 785–796
24. H.Matsuno et. al: Boundary Formation by Notch Signaling in Drosophila Multicellular Systems: Experimental Observations and Gene Network Modeling by Genomic Object Net. Pacific Symposium on Biocomputing 8, 2003, 152–163
25. U.Montanari, F.Rossi: Contextual Nets. Acta Informatica 32, 1995, 545–596
26. P.D.Nieuwkoop: Activation and Organisation of the Central Nervous System in Amphibians. III. Synthesis of a New Working Hypothesis. Journal of Experimental Zoology 120, 1952, 83–108
27. W.Reisig, G.Rozenberg (Eds.): Lectures on Petri Nets I and II. LNCS 1491 and 1492, 1998
28. S.Scholpp, M.Brand: Endocytosis Controls Spreading and Effective Signaling Range of Fgf8 Protein. Current Biology 14, 2004, 1834–1841
29. J.Shiotsuguet et. al: Multiple Points of Interaction between Retinoic Acid and Fgf Signaling During Embryonic Axis Formation. Development 131, 2004, 2653–2667
30. L.J.Steggles, R.Banks, O.Shaw et al.: Qualitatively Modelling and Analysing Genetic Regulatory Networks: a Petri Net Approach. Bioinformatics 23, 2006, 363–343
31. C.Talcott, D.L.Dill: Multiple Representations of Biological Processes. Transactions on Computational Systems Biology, 2006
32. A.A.Teleman, M.Strigini, S.M.Cohen: Shaping Morphogen Gradients. Cell 105, 2001, 559–562
33. C.J.Tomlin, J.D.Axelrod: Biology by Numbers: Mathematical Modelling in Developmental Biology. Nature Reviews Genetics 8, 2007, 331–340
34. W.Vogler: Partial Order Semantics and Read Arcs. Theoretical Computer Science 286, 2002, 33–63
35. R.J.White et. al: Complex Regulation of *cyp26a1* Creates a Robust Retinoic Acid Gradient in the Zebrafish Embryo. Plos biology 5, 2007, 2522–2533
36. L.Wolpert: Principles of Development. Oxford University Press, 2002

Comparison of approximate kinetics for unireactant enzymes: Michaelis-Menten against the equivalent server

Alessio Angius[1], Gianfranco Balbo[1], Francesca Cordero[1,2], András Horváth[1], and Daniele Manini[1]

[1] Department of Computer Science, University of Torino, Torino, Italy
[2] Department of Clinical and Biological Sciences, University of Torino, Torino, Italy

Abstract. Mathematical models are widely used to create complex biochemical models. Model reduction in order to limit the complexity of a system is an important topic in the analysis of the model. A way to lower the complexity is to identify simple and recurrent sets of reactions and to substitute them with one or more reactions in such a way that the important properties are preserved but the analysis is easier.
In this paper we consider the typical recurrent reaction scheme $E + S \rightleftharpoons ES \longrightarrow E + P$ which describes the mechanism that an enzyme, E, binds a substrate, S, and the resulting substrate-bound enzyme, ES, gives rise to the generation of the product, P. If the initial quantities and the reaction rates are known, the temporal behaviour of all the quantities involved in the above reactions can be described exactly by a set of differential equations. It is often the case however that, as not all necessary information is available, only approximate analysis can be carried out. The most well-known approximate approach for the enzyme mechanism is provided by the kinetics of Michaelis-Menten. We propose, based on the concept of the flow-equivalent server which is used in Petri nets to model reduction, an alternative approximate kinetics for the analysis of enzymatic reactions. We evaluate the goodness of the proposed approximation with respect to both the exact analysis and the approximate kinetics of Michaelis and Menten. We show that the proposed new approximate kinetics can be used and gives satisfactory approximation not only in the standard deterministic setting but also in the case when the behaviour is modeled by a stochastic process.

1 Introduction

Mathematical models are widely used to describe biological pathways because, as it is phrased in [1], they "offer great advantages for integrating and evaluating information, generating prediction and focusing experimental directions". In the last few years, high-throughput techniques have increased steadily, leading to

Cordero is the recipient of a research fellowship supported by grants from Regione Piemonte, University of Torino and MIUR. Horváth is supported by MIUR PRIN 2008.

the production of a huge volume of data used to derive the complex texture behind the biological/biochemical mechanisms, and creating in this way the structure needed for mathematical modelling. Indeed, many models based on the combination and the integration of various elements in order to investigate their relationships and behaviour have been devised which become more complex with the growth of available data. The complexity is reflected in the number of dynamic state variables and parameters, as well as in the form of the kinetic rate expressions.

Such complexity leads to difficulties both from the point of view of defining the model as the parametrisation becomes unfeasible and for what concerns the analysis of the model. It is often the case hence that in order to have a model which is feasilble for the analysis simplifications must be performed.

In this paper we focus our attention on the simplified, approximate treatment of a set of reactions that very often appears as building blocks of complex models. We consider the reactions

$$E + S \underset{k_{-1}}{\overset{k_1}{\rightleftarrows}} ES \xrightarrow{k_2} E + P \tag{1}$$

describing that the enzyme, E, attaches reversibly to the substrate, S, forming the substrate-bound enzyme ES which gives rise then to the product P releasing the enzyme. This and similar enzymatic reactions are widely studied in biology. The most common approximate approach to deal with them is provided by the Michaelis-Menten (MM) kinetics (called also Michaelis-Menten-Henri kinetics) which, based on quasi-steady-state assumptions, connects the speed of producing P directly to the concentration of E and P, omitting the explicit modeling of ES.

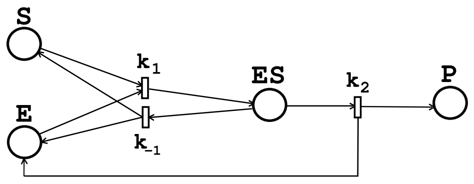


Fig. 1. Petri net representation of the reactions given in (1)

System of enzymatic reactions can be described by Petri nets [2] (Figure 1 shows the Petri net corresponding to the reactions given in (1)) and then analysed by methods developed for this formalism. We propose for the reactions in (1) an alternative to the approximate Michaelis-Menten kinetics. This new approximate kinetics is based on a concept widely used in the analysis of Petri nets and models described by other formalisms like queueing networks and process algebras. This concept is called the flow equivalent server [3]. The application of this concept, similarly to the Michaelis-Menten kinetics, leads to a simplified set of reactions in which the intermediate complex ES is not modeled explicitly. The difference is, however, that, since the application of the flow equivalent server (FES) is based on assumptions that are different and less strict than those used by the Michaelis-Menten kinetics, the resulting approximation is more robust.

The concept of flow equivalent server has already been used in [4] where a complex signal transduction model was considered. In that paper we have shown that this concept can be applied not only to the small set of reactions given in (1) but also to bigger submodels. This leads to a simplified model which has less parameters and whose analysis is not as heavy as that of the complete one. For the model presented in [4] it was shown that the quantitative temporal behaviour of the simplified model coincides satisfactorily with that of the complete model and that important qualitative properties are maintained as well. In this paper our goal is to study in detail the goodness of the FES based approximation for the reactions in (1) and to compare it to the widely-used approximate kinetics of Michaelis, Menten and Henri.

The paper is organised as follows. Section 2 provides the necessary background, Section 3 describes the concept of the flow equivalent server and Section 4 presents the results of the comparison between the approximation approaches. We conclude with a discussion and an outlook on future works in Section 5.

2 Background

In 1901 Henri [5] proposed a partly reversible reaction scheme to describe the enzymatic process. According to this scheme the enzyme E and the substrate S form, through a reversible reaction, the enzyme-substrate complex ES. This complex can then give rise to the product P through an irreversible reaction during which the enzyme is freed and can bind again to other molecules of the substrate. This scheme is summarised in (1) where k_1 is the rate of the binding of E and S, k_{-1} is the rate of the unbinding of ES into E and S and k_2 is the rate at which ES decays to the product P freeing the enzyme E.

There are two typical approaches to associate a quantitative temporal behaviour to the reactions in (1). The first results in a deterministic representation while the other in a stochastic one. In the following we give a brief idea of both approaches. For a detailed description see, for example, [6, 7].

The deterministic approach describes the temporal behaviour of a reaction with a set of ordinary differential equations (ODE). For the reactions in (1) we have

$$\begin{aligned}
\frac{d[E]}{dt} &= -k_1[E][S] + (k_{-1} + k_2)[ES] \\
\frac{d[S]}{dt} &= -k_1[E][S] + k_{-1}[ES] \\
\frac{d[ES]}{dt} &= k_1[E][S] - (k_{-1} + k_2)[ES] \\
\frac{d[P]}{dt} &= k_2[ES]
\end{aligned} \tag{2}$$

where $[X]$ is the concentration of molecule X at time t. These equations state that the rate at which the concentration of a given molecule changes equals the difference between the rate at which it is formed and the rate at which it is utilised. The four equations can be solved numerically to yield the concentration of E, S, ES and P at any time t if both the initial concentration levels ($[S]_0$, $[E]_0$, $[ES]_0$, $[P]_0$) and the reaction rates (k_1, k_{-1}, k_2) are known. In the deterministic approach the concentrations of the molecules are described by continuous quantities.

In the stochastic approach a continuous time Markov chain (CTMC) is used to describe the process. Each state of the chain is described by a vector of integers in which the entries give the quantities of the molecules, which, accordingly, assume discrete values. These discrete values are resulting either directly from molecule count or from discretization of continuous values. Reactions are modeled by transitions between the states. For example, from state $|x_1, x_2, x_3, x_4|$ where x_1, x_2, x_3 and x_4 are the quantities of the molecules E, S, ES and P, respectively, there is a transition to state $|x_1 - 1, x_2 - 1, x_3 + 1, x_4|$ with rate $k_1 x_1 x_2$ which corresponds to the binding of one molecule E with one molecule S to form one molecule of ES. It is easy to see that even for small models the corresponding CTMC can have a huge state space whose transition rate structure is non-homogeneous. Exact analytical treatment of these chains is often unfeasible and in most cases simulation is the only method that can be used for their analysis.

2.1 Michaelis-Menten approximate kinetics

Under some assumptions, the temporal, quantitative dynamics of the mechanism described by the reactions in (1) can be summarised as follows. Initially we have a certain amount of substrate, denoted by $[S]_0$, and enzyme, denoted by $[E]_0$, and no complex ES ($[ES]_0 = 0$). Assuming that k_2 is significantly smaller than k_1 and k_{-1}, a brief transient period occurs during which the amount of the complex ES quickly increases up to a “plateau” level where it remains stable for a long period of time. As the ratio of $[S]_0/[E]_0$ increases, the time needed to reach the condition $d[ES]/dt \approx 0$ decreases and the period during which $d[ES]/dt \approx 0$ increases. In this period we have approximately

$$\frac{d[ES]}{dt} = k_1[E][S] - [ES](k_{-1} + k_2) = 0$$

from which, considering that the total amount of enzyme is conserved, i.e. $[E] + [ES] = [E]_0$, the quantity of ES can be expressed as

$$[ES] = \frac{[E]_0[S]}{\frac{k_{-1}+k_2}{k_1} + [S]} = \frac{[E]_0[S]}{k_M + [S]} \tag{3}$$

where the term $k_M = \frac{k_{-1}+k_2}{k_1}$ is called the Michaelis-Menten constant. Applying (3), the speed of the production of P can be approximated by

$$v_{MM} = \frac{k_2[E]_0[S]}{[S] + k_M} \tag{4}$$

Accordingly, after the "plateau" level of ES is reached, the kinetic parameters k_1, k_{-1} and k_2 together with $[S]$ and the initial total quantity of the enzyme, $[E]_0$, determine the overall rate of the production of P.

Applying the approximate kinetics of Michaelis and Menten, the differential equations describing the reactions become

$$\begin{aligned}\frac{d[E]}{dt} &= 0 \\ \frac{d[S]}{dt} &= -\frac{k_2[E][S]}{[S]+k_M} \\ \frac{d[P]}{dt} &= \frac{k_2[E][S]}{[S]+k_M}\end{aligned} \tag{5}$$

3 Approximate kinetics by flow equivalent server

In this section we derive an alternative approximate kinetics for the analysis of enzymatic reactions, based on the concept of the flow equivalent server. This technique was originally proposed in the context of the steady-state solution of queueing networks [3, 8, 9] and can be adapted to our purposes with a proper interpretation of the assumptions on which it is based. The idea behind this concept is to consider the reactions given in (1) as a fragment of a large biological system in which substrate S is produced by an "up-stream" portion of the system and product P is used "down-stream" within the same system. The goal of the flow equivalent method is to consider the flow of moles that move from the substrate to the product, in the presence of an enzyme that catalyse this phenomenon, and to evaluate its intensity in order to define the overall speed of a "composite" reaction that captures this situation in an abstract manner.

Figure 2 depicts the Petri net corresponding to the reactions of (1) organised in order to make explicit the relationship between the substrate S and the product P, via the enzyme E, enclosing in a dashed box the elements of the system whose dynamics we want to mimic with the composite transition. This

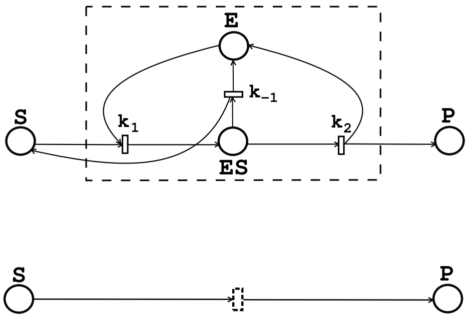


Fig. 2. Petri net of the reactions in (1) organised for computation of the flow equivalent-transition (above) and its approximation (below)

picture makes evident the fact that the speed of the composite transition must depend not only on the speeds of the transitions included in the box but also on the quantities present in the box, namely, the total amount of enzyme. Assuming to know the kinetic constants of the reactions inside the box and the quantity of the enzyme, the speed of the composite transition also depends on the amount $[S]$ that participates in the reactions and that may change during the evolution of the whole system. Following this point of view, it is possible to conceive a characterisation of the speed of the composite transition that is conditioned on the quantity of S. The flow equivalent approach accounts for this observation by computing the intensity of the flow of moles that reaches place P assuming that the total amount of S remains constant. Technically, this is obtained by short-circuiting the output and input places of the sub-net (introducing an immediate transition [10] that connects place P with place S) and by computing the throughput along the short-circuit which will be conditioned on the initial amount of S and that will thus be computed for all the possible values of S. In general, this amounts to the construction of a table that looks like that depicted in Figure 3, where S_1, S_2, ..., S_n represent different values of the amount of substrate S for which the speeds of the composite reaction $v_{FES}(S_1)$, $v_{FES}(S_2)$, ..., $v_{FES}(S_n)$ are computed, given that k_1, k_{-1}, k_2, and $\#E$ are assumed to be the values of the kinetics constant of the reactions in the box and of the amount of enzyme E.

In practice, this corresponds to the construction and to the (steady state) solution of the continuous time Markov chain (CTMC) that corresponds to the sub-model in isolation. Providing the speed of the composite transition in the tabular form highlighted by Figure 3 is convenient for cases where the domain of the function is "small", but may be impractical in many common situations. Despite the computational complexity of the approach, we must notice that the equilibrium assumption of the flow equivalent method is used only to obtain an approximate characterisation of the throughput for different sets of initial conditions and does not mean that the equivalent speed can only be used for steady state analysis.

The concept of flow equivalent server described above is used traditionally in a stochastic setting. However, it can be applied in a deterministic setting as well using arguments that are summarized by the following points. The complexity of the approach in the stochastic setting becomes prohibitive when the amount of the substrate S becomes very large. On the other hand, this is the case in

Given $k_1, k_{-1}, k_2,$ and $\#E$	
S_1	$v_{FES}(S_1)$
S_2	$v_{FES}(S_2)$
$\cdots$	$\cdots$
S_n	$v_{FES}(S_n)$

Fig. 3. Flow Equivalent Server characterisation

which the stochastic (or at least the average) behaviour of the model is conveniently captured by a set of ODE, i.e., by a deterministic model. Moreover, in the case of our model, the equilibrium solution of the set of differential equations corresponding to the short-circuited model is simple enough to obtain an analytic expression for the speed of the composite transition as it is described in the following.

We assume that the initial condition is $[E]_0 = M_1$, $[S]_0 = M_2$, $[ES]_0 = 0$, and $[P]_0 = 0$. We will denote the steady state measures of the compounds by $[E], [S], [ES]$ and $[P]$. In the short-circuited version of the reactions given in (1), moles transformed in P are immediately moved back to S and consequently its steady state measure is zero (i.e., $[P] = 0$). The steady state measures of the other compounds can be determined by considering

– the fact that in steady state the rate of change of the quantities of the different compounds is zero, i.e., we have

$$\begin{aligned}\frac{d[E](t)}{dt} &= 0 = -k_1[E][S] + k_{-1}[ES] + k_2[ES] \\ \frac{d[S](t)}{dt} &= 0 = -k_1[E][S] + k_{-1}[ES] + k_2[ES] \\ \frac{d[ES](t)}{dt} &= 0 = +k_1[E][S] - k_{-1}[ES] - k_2[ES]\end{aligned} \tag{6}$$

which are three dependent equations;

– and the following equations expressing conservation of mass

$$[E] + [ES] = M_1, \quad [S] + [ES] = M_2 \tag{7}$$

In (6) and (7) we have three independent equations for three unknowns. There are two solutions but only one of them guarantees positivity of the unknowns. The speed of producing P is given by the steady state quantity of ES multiplied by k_2. This speed is

$$v_{FES} = \frac{k_2\left([E] + [S] + k_M - \sqrt{([E] - [S])^2 + 2k_M([E] + [S]) + k_M^2}\right)}{2} \tag{8}$$

Accordingly, the set of ordinary differential equations describing the reactions given in (1) becomes

$$\begin{aligned}\frac{d[E]}{dt} &= 0 \\ \frac{d[S]}{dt} &= -\frac{k_2\left([E] + [S] + k_M - \sqrt{([E] - [S])^2 + 2k_M([E] + [S]) + k_M^2}\right)}{2} \\ \frac{d[P]}{dt} &= \frac{k_2\left([E] + [S] + k_M - \sqrt{([E] - [S])^2 + 2k_M([E] + [S]) + k_M^2}\right)}{2}\end{aligned} \tag{9}$$

which explicitly reflects the assumption of the conservation of E and the observation that substrate S is transformed into product P.

4 Numerical illustration

In this section, we first compare in Section 4.1 the MM and FES approximate kinetics from the point of view of the speed they assign to the production of P as function of the reaction rates (k_1, k_{-1}, k_2) and the concentration of the enzyme and the substrate $([E], [S])$. Subsequently, in Sections 4.2 and 4.3 we compare the quantitative behaviour of the approximations to that of the full model in the deterministic and in the stochastic setting, respectively.

It is easy to check that as the quantity of the substrate tends to infinity the two approximate kinetics lead to the the same speed of production. In both cases for the maximum speed of production we have

$$v_{\max} = \lim_{[S]\to\infty} v_{MM} = \lim_{[S]\to\infty} v_{FES} = k_2[E] \tag{10}$$

Another situation in which the two approximate kinetics show perfect correspondence is when the quantity of the enzyme is very low. This can be shown formally by observing that

$$\lim_{[E]\to 0} \frac{v_{MM}}{v_{FES}} = 1 \tag{11}$$

4.1 Production speeds

A typical way of illustrating the approximate Michaelis-Menten kinetics is to plot the production speed against the quantity of the substrate. Figure 4 gives such illustrations comparing the speeds given by the two approximate kinetics. Reaction rate k_2 is either 0.1, 1 or 10 and reaction rates k_1 and k_{-1} are varied in order to cover different situations for what concerns the ratio k_1/k_{-1}. Two different values of $[E]$ are considered. The limit behaviours expressed by (10) and (11) can be easily verified in the figures. On the left sides of the figure it can be observed that for small values of $[E]$ the two approximations are almost identical for all considered values of the reaction rates, thus in agreement with the trend conveyed by (11). It can also be seen that for larger values of $[E]$ the two approximations are rather different and the difference is somewhat increasing as k_2 increases, and becomes more significant for higher values of k_1/k_{-1}. In all cases the curves become closer to each other when the amount of $[S]$ increases.

4.2 Deterministic setting

In this section we compare the different kinetics in the deterministic setting. Once the initial quantities and the reaction rates are defined, the systems of differential equations given in (2),(5) and (9) can be numerically integrated and this provides the temporal behaviour of the involved quantities, used as references for the comparisons.

For the first experiments we choose such parameters with which the two approximate kinetics result in different speeds of production. Based on Figure 4 this is achieved whenever the quantity of the enzyme is comparable to the quantity

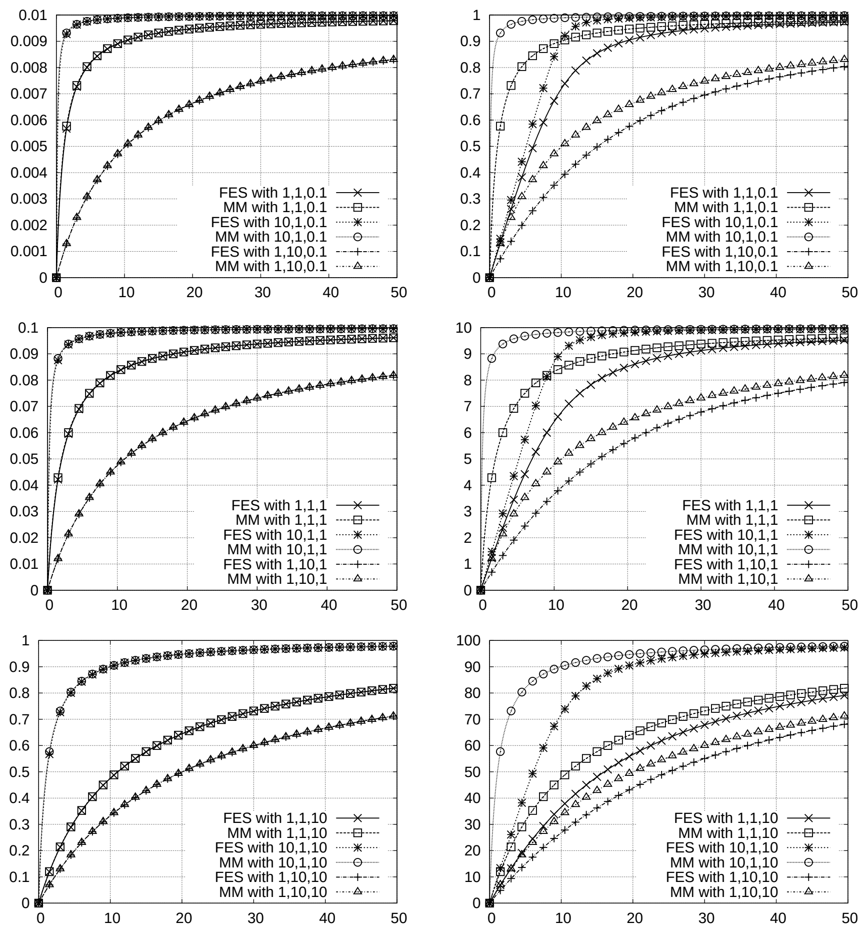


Fig. 4. Production speed as function of substrate quantity with $[E] = 0.1$ for the figures on the left side and with $[E] = 10$ on the right side; reaction rates are given in the legend in order k_1, k_{-1} and k_2

of the substrate. Accordingly, we set $[E]_0 = [S]_0 = 10$. For the full model $[ES]_0$ needs to be set too, and we choose $[ES]_0 = 0$. This choice does not help the approximations. They assume that the total enzyme concentration $[E]_0 + [ES]_0$ is immediately distributed between $[E]$ and $[ES]$, thus making possible an immediate (consistent) production of P. On the contrary, in the full model the production of $[ES]$ takes time and thus the speed of the production of P must start from 0, growing to a high value only later. Figures 5 and 6 depict the quantity of the product and the speed of its production as functions of time for two different sets of reaction rates. In both figures the kinetics based on flow equivalence provides precise approximation of the production of P. The Michaelis-Menten kinetics instead fails to follow the full model, but this is not surprising as the derivation of this kinetics assumes small amount of enzymes. It can also be seen that high values of k_1/k_{-1} (Figure 6) lead to worst approximation in case of Michaelis-Menten kinetics. On the right hand side of the figures one can observe that for the full model the speed of producing P is 0 at the beginning and then it increases fast to the speed foreseen by the FES approximation.

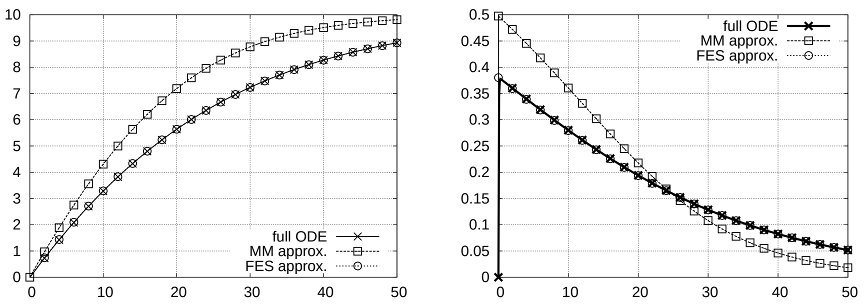


Fig. 5. Quantity of product (left) and speed of production (right) as function of time with $k_1 = 1$, $k_{-1} = 10$, $k_2 = 0.1$, $[E] = 10$ and initial quantity of substrate equals 10

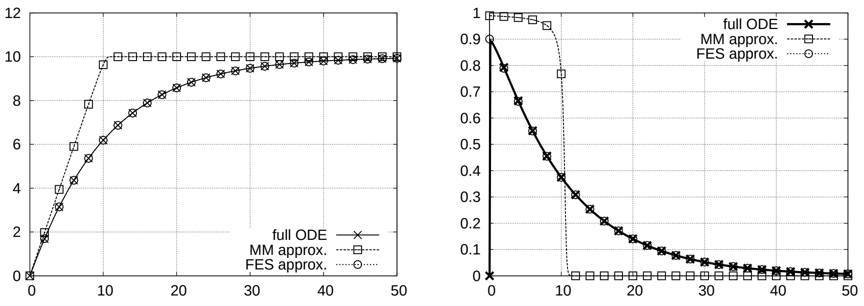


Fig. 6. Quantity of product (left) and speed of production (right) as function of time with $k_1 = 10$, $k_{-1} = 1$, $k_2 = 0.1$, $[E] = 10$ and initial quantity of substrate equals 10

A second set of experiments is illustrated in Figures 7 and 8. We choose sets of parameters with which the speed of production of the MM and FES approximations are similar. In these cases both approximations are close to the reference behaviour. Still, it can be seen that for high values of k_1/k_{-1} (Figure 8) the approximation provided by the Michaelis-Menten kinetics is slightly less precise.

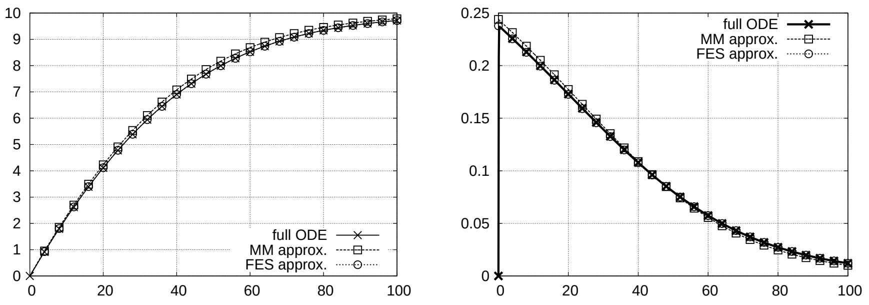


Fig. 7. Quantity of product (left) and speed of production (right) as function of time with $k_1 = 1$, $k_{-1} = 10$, $k_2 = 0.5$, $[E] = 1$ and initial quantity of substrate equals 10

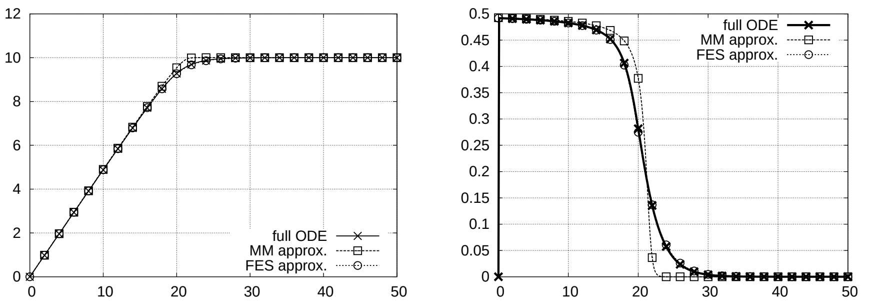


Fig. 8. Quantity of product (left) and speed of production (right) as function of time with $k_1 = 10$, $k_{-1} = 1$, $k_2 = 0.5$, $[E] = 1$ and initial quantity of substrate equals 10

In the following we turn our attention to the cases in which both the approximations are less reliable. In Figure 9 we plotted the case $k_1 = 0.1$, $k_{-1} = 0.1$, $k_2 = 0.1$, $[E] = 1$, $[S]_0 = 1$ and $[ES]_0 = 0$. As mentioned earlier, with $[ES]_0 = 0$ the initial production speed in the original model is 0 while it is immediately high in the approximate kinetics. With low values of k_1 and k_{-1}, the time taken by the system to reach the quasi-steady-state situation assumed by the approximate kinetics is quite long. For this reason there is a longer initial period in which P is produced by the approximations at a “wrong” speed. Furthermore, decreasing k_1 and k_{-1} would lead to a longer period in which the approximate kinetics are not precise (see Figure 9).

Another way of “disturbing” the approximations is to dynamically change the quantity of the substrate in the system. In the original model, because of the intermediate step yielding ES, the speed of producing P changes only after some delay. On the contrary, the approximations react immediately. The harsher the change in the quantity of the substrate the larger is the difference between the original model and the approximations. This phenomenon is reflected in the

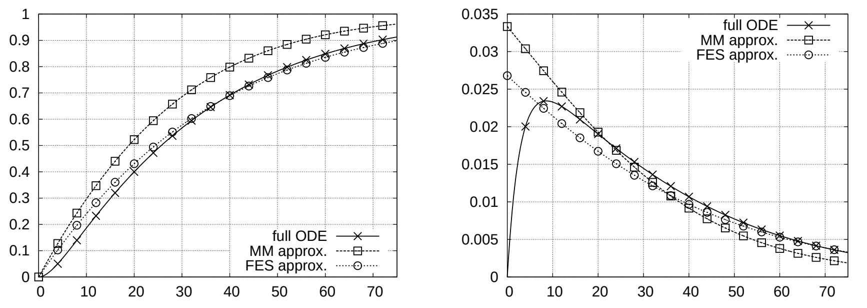


Fig. 9. Quantity of product (left) and speed of production (right) as function of time with $k_1 = 0.1$, $k_{-1} = 0.1$, $k_2 = 0.1$, $[E] = 1$ and initial quantity of substrate equals 1

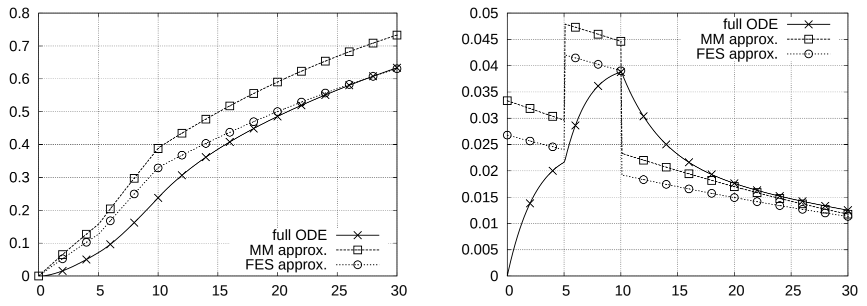


Fig. 10. Quantity of product (left) and speed of production (right) as function of time with $k_1 = 0.1$, $k_{-1} = 0.1$, $k_2 = 0.1$, $[E] = 1$, initial quantity of substrate equals 1 and adding substrate to the system according to (12)

model by adding the following term to the differential equation that describes the quantity of the substrate:

$$10(U(t-5) - U(t-5.1)) - 10(U(t-10) - U(t-10.1)) \tag{12}$$

where U denotes the unit-step function. The effect of (12) is to add 1 unit of substrate to the system in the time interval $[5, 5.1]$ and to take away 1 unit of substrate from it in the time interval $[10, 10.1]$. The resulting behaviour is depicted in Figure 10. The approximations change the speed of producing P right after the change in the quantity of the substrate while the original model reacts to the changes in a gradual manner. Naturally, if the quantity of the substrate undergoes several harsh changes then the MM and the FES kinetics can result in bad approximation of the full model.

4.3 Stochastic setting

In the following we compare the different kinetics in the stochastic setting, by analysing the corresponding CTMCs. In particular, we determine by means of simulation the average and the variance of the quantity of the product as function of time. The simulations were carried out in Dizzy [11].

The reaction rates for the first set of experiments are $k_1 = k_{-1} = k_2 = 1$. As in the previous section, this choice allows to test a situation where the speed of the two approximations are different. For the same reason, we choose the same initial quantity for the enzyme and the substrate $[E]_0 = [S]_0 = 1$. In the stochastic setting the discretization step, denoted by δ, has to be chosen as well. This choice has a strong impact because as the granularity with which the concentrations are modeled is increased, the behaviour of the CTMC tends to the deterministic behaviour of the corresponding ODE. Figures 11 and 12 depict the average and the variance of the quantity of the product with $\delta = 0.01$ and $\delta = 0.001$, respectively. In both figures the approximate kinetics based on flow equivalence gives good approximation of the original average behaviour while the Michaelis-Menten approximation results in too fast production of P. On the right side on the figures one can observe that also the variance is approximated better by the FES approximation.

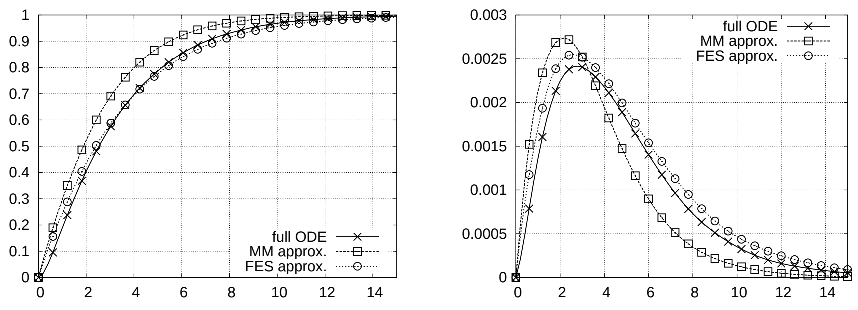


Fig. 11. The average (left) and the variance (right) of the quantity of the product as function of time with $k_1 = 1$, $k_{-1} = 1$, $k_2 = 1$, $[E]_0 = [S]_0 = 1$ and $\delta = 0.01$

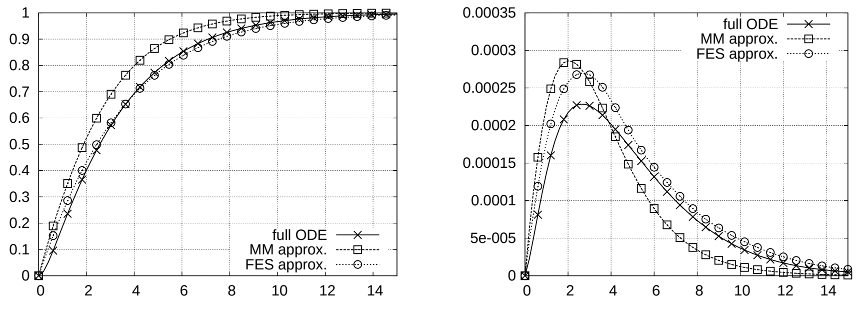


Fig. 12. The average (left) and the variance (right) of the quantity of the product as function of time with $k_1 = 1$, $k_{-1} = 1$, $k_2 = 1$, $[E]_0 = [S]_0 = 1$ and $\delta = 0.001$

For the second set of experiments we set $k_1 = 10$ and $k_{-1} = k_2 = 1$ and as initial states we choose again $[E]_0 = [S]_0 = 1$. In this case too, as it was shown in Figure 4, the speeds of production of P as predicted by the MM and FES approximations are quite different. Figures 13 and 14 depict the resulting behaviour for two different values of δ. As in case of the deterministic setting, the Michaelis-Menten approximation suffers from the increased k_1/k_{-1} ratio and becomes less precise than before. The FES based approach still results in good approximation for both the average and the variance of the production.

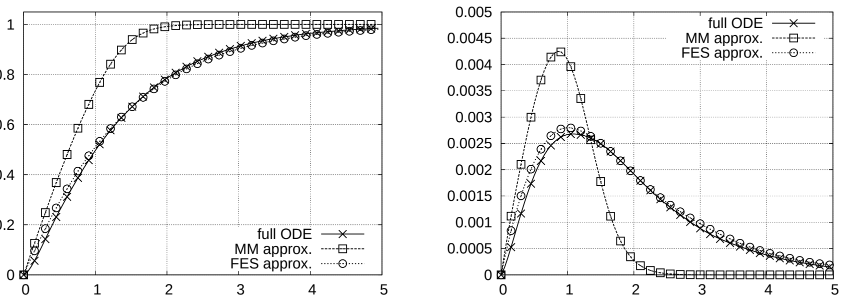


Fig. 13. The average (left) and the variance (right) of the quantity of the product as function of time with $k_1 = 10$, $k_{-1} = 1$, $k_2 = 1$, $[E]_0 = [S]_0 = 1$ and $\delta = 0.01$

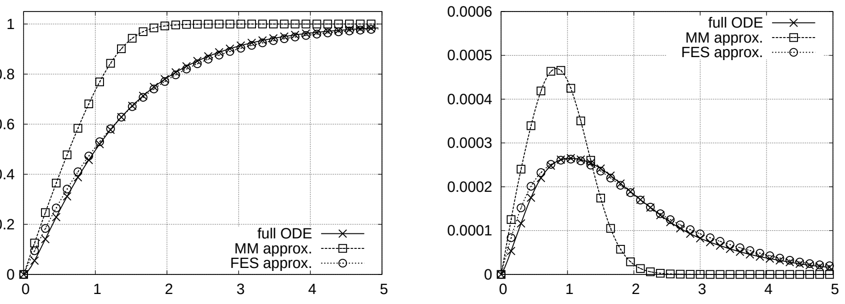


Fig. 14. The average (left) and the variance (right) of the quantity of the product as function of time with $k_1 = 10$, $k_{-1} = 1$, $k_2 = 1$, $[E]_0 = [S]_0 = 1$ and $\delta = 0.001$

5 Conclusion

In this paper we have considered the approximate treatment of the basic enzymatic reactions $E+S \rightleftharpoons ES \longrightarrow E+P$. In particular, an approximate kinetics, based on the concept of flow equivalent server, has been proposed for its analysis. This FES approximate kinetics has been compared to both the exact model and to the most common approximate treatment, namely, the Michaelis-Menten kinetics. We have shown that the FES kinetics is more robust than the one of Michaelis-Menten.

The FES approximation for the basic enzymatic reactions is computationally convenient due to the fact that it has been possible to find an analytic expression for the speed of the composite reaction in this case. While it is very unlikely for this to be true in the case of more complex kinetics, the method is very general and we will study it further within this context to see if it is possible to find other functional expressions for the speed of the composite reaction. One direction of research will be computing the flow equivalent characterization of the kinetics for a number of specific parameter sets and then of constructing the functional representations via interpolation.

References

1. Fisher, J., Henzinger, T.A.: Executable cell biology. Nature Biotechnology **25** (November 2007) 1239–1249
2. Reddy, V., Mavrovouniotis, M., Liebman, M.: Qualitative analysis of biochemical reaction systems. Comput. Biol. Med. **26** (1996) 9–24
3. Chandy, K.M., Herzog, U., Woo, L.S.: Parametric analysis of queueing networks. IBM Journal of R. & D. **19**(1) (January 1975) 36–42
4. Napione, L., Manini, D., Cordero, F., Horváth, A., Picco, A., Pierro, M.D., Pavan, S., Sereno, M., Veglio, A., Bussolino, F., Balbo, G.: On the use of stochastic Petri nets in the analysis of signal transduction pathways for angiogenesis process. In: Proc. of The 7th Conference on Computational Methods in Systems Biology (CMSB 2009). Volume 5688 of Lecture Notes in Bioinformatics., Bologna, Italy (Sept 2009) 281–295
5. Henri, V.: Recherches sur la loi de l'action de la sucrase. Compt. Rend. **133** (1901) 891–894
6. Voit, E.O.: Computational Analysis of Biochemical Systems. Cambridge University Press (2000)
7. Gillespie, D.: A rigorous derivation of the master chemical equation. Physica **188** (1992) 404–425
8. Denning, P.J., Buzen, J.P.: The operational analysis of queueing network models. ACM Comput. Surv. **10**(3) (1978) 225–261
9. Stewart, W.J.: Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling. Princeton Univ. Press (2009)
10. Ajmone Marsan, M., Balbo, G., Conte, G.: A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. ACM Transactions on Computer Systems **2**(1) (May 1984)
11. Ramsey, S., Orrell, D., Bolouri, H.: Dizzy: Stochastic simulation of large-scale genetic regulatory networks. J. Bioinformatics Comput. Biol. **3**(2) (2005) 415–436

Colored Petri nets to model and simulate biological systems

Fei Liu and Monika Heiner

Department of Computer Science, Brandenburg University of Technology
Postbox 10 13 44, 03013 Cottbus, Germany
{liu, monika.heiner}@informatik.tu-cottbus.de

Abstract. Petri nets have become an effective formalism to model biological systems. However, attempts to simulate biological systems by low-level Petri nets are restricted to relatively small models, and they tend to grow quickly for modeling complex systems, which makes it more difficult to manage and understand the nets. Motivated by this, we propose a colored Petri net-based framework for modeling, simulating, and analyzing complex biological systems. We give the definitions of biochemically interpreted colored qualitative Petri nets (QPN^C) and colored stochastic Petri nets (SPN^C) and describe their functionalities and features implemented in the Petri net tool Snoopy. We use two examples, the cooperative ligand binding and the repressilator, to demonstrate how to construct and simulate QPN^C and SPN^C models, respectively.

1 Motivation

With the rapid growth of data being generated in the biological field, it has become necessary to organize the data into coherent models that describe system behavior, which are subsequently used for simulation, analysis or prediction. A large variety of modeling approaches has already been applied to modeling a wide array of biological systems (see [HK09] for a review). Among them, Petri nets are especially suitable for representing and modeling the concurrent, asynchronous, and dynamic behavior of biological systems, which were first introduced to the qualitative analysis of the biochemical reaction systems by Reddy et al. [RML93]. Motivated by the qualitative analysis of Petri nets, many applications of Petri nets (e.g. stochastic Petri nets, timed Petri nets, continuous Petri nets, and hybrid Petri nets, etc.) have been developed for modeling and simulating biological systems [GH06]. Since biological processes are inherently stochastic, stochastic Petri nets have recently become a modeling paradigm for capturing their complex dynamics, which can help to understand the behavior of complex biological systems by integrating detailed biochemical data and providing quantitative analysis results, see e.g. [JP98], [NOG+05], [PRA05].

Petri nets provide a formal and clear representation of biological systems based on their firm mathematical foundation for the analysis of biochemical properties. However, low-level Petri nets do not scale. So attempts to simulate biological systems by low-level Petri nets have been mainly restricted so far to relatively small models. They tend to grow quickly for modeling complex systems, which makes it more difficult to manage and understand the nets, thus increasing the risk of modeling errors [Mur07]. Two known modeling concepts improving the situation are hierarchy and color. Hierarchical structuring has been discussed a lot, see e.g. [MWW09], while the color has gained little attention so far. Thus, we investigate how to apply colored Petri nets to modeling and analyzing biological systems. To do so, we not only provide compact and readable representations of complex biological systems, but also do not lose the analysis capabilities of low-level Petri nets, which can still be supported by automatic unfolding. Moreover, another attractive advantage of colored Petri nets for a biological modeler is that they provide the possibility to easily increase the size of a model consisting of many similar subnets just by adding colors.

In this paper, we propose a colored Petri net-based framework for modeling, simulating, and analyzing biological systems. We are developing tools to support this new framework. Two prototypes for colored qualitative Petri nets (QPN^C) and colored stochastic Petri nets (SPN^C) have been implemented in Snoopy, a tool for modeling and animating/simulating hierarchical graph-based formalisms [Sno10]. We will describe these two prototypes and some applications.

This paper is organized as follows. Section 2 outlines the colored Petri net-based framework for modeling, simulating and analyzing biological systems and gives the definitions of QPN^C and SPN^C. Section 3 discusses the functionalities and features for QPN^C and SPN^C, which have been implemented in Snoopy. Section 4 shows how to construct basic colored Petri net components, and gives two examples to demonstrate QPN^C and SPN^C, respectively. Section 5 summarizes related work. Finally, conclusions and outlook are given.

2 Colored Petri net-based framework

In this section, we propose a colored Petri net-based framework for modeling, and simulating/analyzing biological systems, illustrated in Fig. 1, which extends the Petri net-based framework for modeling, and simulating/analyzing biological systems introduced in [GHL07], i.e., the new proposed framework is in fact the colored version of the existing framework. Both of these frameworks unify the qualitative, stochastic and continuous Petri net paradigms, but the colored version provides more compact and readable representations of complex biological systems.

The new framework relates three modeling paradigms: QPN^C, SPN^C, and colored continuous Petri nets (CPN^C), just like the Petri net-based framework that relates qualitative Petri nets (QPN), stochastic Petri nets (SPN) and continuous Petri nets (CPN). QPN^C is an abstraction of SPN^C and CPN^C, while SPN^C and CPN^C are mutually related by approximation. The user can refer to [GHL07] to take a closer look at the detailed relationship between these three paradigms. In the following, we will describe QPN^C and SPN^C in detail, but not CPN^C as we have not investigated CPN^C yet.

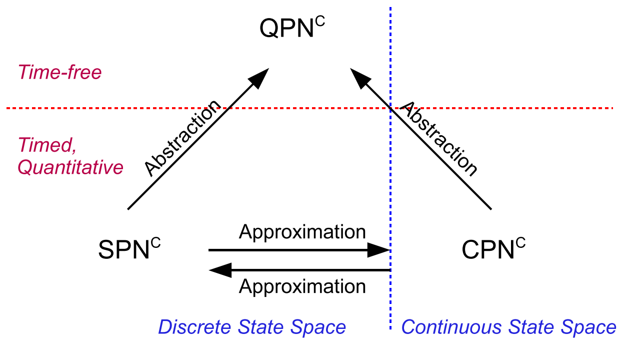


Fig. 1. Colored Petri net-based framework for modeling, and simulating/analyzing biological systems.

2.1 Colored qualitative Petri Nets (QPN^C)

We assume basic knowledge of the standard notions of qualitative place/transition Petri nets, see e.g. [Mur89], [HGD08]. In the following, we will briefly describe QPN, and then give the definition of QPN^C.

QPN is the basic Petri net class, which consists of places, transitions, and arcs. QPN does not associate a time with transitions or the sojourn of tokens at places, and thus is time-free [GHL07]. QPN^C is a colored extension of QPN.

Colored Petri nets were first proposed by Jensen [Jen81], which combine Petri nets with capabilities of programming languages to describe data types and operations, thus providing a flexible way to create compact and parameterizable models. In colored Petri nets, tokens are distinguished by the "color", rather than having only the "black" one. Besides, arc expressions, an extended version of arc weights, specify which tokens can flow over the arcs, and guards that are in fact Boolean expressions define additional constraints on the enabling of the transitions [JKW07]. In the following, we give the definition of the QPN^C based on the definition of colored Petri nets by Jensen [JKW07]. Here we denote by EXP the set of expressions that comply with a predefined syntax, which are used as arc expressions, guards, etc.

Definition 1. *A QPN^C is a tuple $< P, T, F, \sum, C, g, f, m_0 >$, where:*

- *P is a finite, non-empty set of places.*
- *T is a finite, non-empty set of transitions.*
- *F is a finite set of directed arcs, such that $F \subseteq (P \times T) \cup (T \times P)$.*
- *$\sum$ is a finite, non-empty set of types, also called color sets.*
- *$C : P \to \sum$ is a color function that assigns to each place $p \in P$ a color set $C(p) \in \sum$.*
- *$g : T \to EXP$ is a guard function that assigns to each transition $t \in T$ a guard expression that has the Boolean type.*

– *$f : F \to EXP$ is an arc function that assigns to each arc $a \in F$ an arc expression that has a multiset type $C(p)_{MS}$, where p is the place connected to the arc a, and $C(p)_{MS}$ is the multiset on the color set $C(p)$.*
– *$m_0 : P \to EXP$ is an initialization function that assigns to each place $p \in P$ an initialization expression that has a multiset type $C(p)_{MS}$.*

QPN^C is a colored extension of the qualitative place/transition net extended by different kinds of arcs, e.g., inhibitor arc and read arc [HRR+08]. These kinds of arcs are not explicitly denoted in the definition above.

2.2 Colored Stochastic Petri Nets (SPN^C)

In this section, we will briefly recall stochastic Petri nets (SPN) and their extensions, and then introduce colored stochastic Petri nets (SPN^C).

SPN are an extension of qualitative place/transition Petri nets. As with a qualitative Petri net, a stochastic Petri net maintains a discrete number of tokens on its places. But contrary to the time-free case, a firing rate (waiting time) is associated with each transition, which is a random variable, defined by an exponential probability distribution. The semantics of a stochastic Petri net is described by a continuous time Markov chain (CTMC). The CTMC of a stochastic Petri net without parallel transitions is isomorphic to the reachability graph of the underlying qualitative Petri net, while the arcs between the system states are now labelled by the transition rates [HLG+09].

There are quite a number of various extensions based on the fundamental stochastic Petri net class SPN, see e.g. [MBC+95], [Ger01]. For example, generalized stochastic Petri nets ($GSPN$) are stochastic Petri nets (SPN) extended by inhibitor arcs and immediate transitions. Deterministic and stochastic Petri nets ($DSPN$) are generalized stochastic Petri nets ($GSPN$) extended by deterministic transitions [HLG+09].

While SPN and its extensions offer enormous modeling power, managing large-scale Petri net models is difficult due to the fact that tokens are indistinguishable. To alleviate this limitation, the SPN^C is presented to uplift biochemically interpreted extended stochastic Petri nets introduced in [HLG+09] to a colored version. As in the QPN^C, in the SPN^C, tokens are distinguished by the "color", and arc expressions and guards have the same meaning. Before expressions are evaluated to values, the variables in the expressions must get assigned values, which is called binding. A binding of a transtion $t \in T$ exactly corresponds to a transition instance, denoted by $t(b)$, i.e., each binding will become an uncolored transition after unfolding. The set of all bindings for a transition t constitutes the set of all the instances of transition t, denoted by $TI(t)$. The set of all instances for all transitions T of a net is denoted by $TI(T)$. In contrast, each color $c \in C(p)$ for a place $p \in P$ exactly corresponds to a place instance, denoted by $p(c)$, i.e., each color will become an uncolored place after unfolding. We let $PI(p)$ denote all the instances of a place p and $PI(P)$ all the instances of all places P of a net. In the following, we give the definition of SPN^C based on QPN^C.

Definition 2. *A biochemically interpreted colored stochastic Petri net SPN^C is a tuple $< P, T, F, \sum, C, g, f, v, l, m_0 >$, where:*

- *$< P, T, F, \sum, C, g, f, m_0 >$ is a QPN^C.*
- *T is refined as the union of three disjoint transition sets, i.e. $T := T_{stoch} \cup T_{im} \cup T_{timed}$ with:*
  - *T_{stoch}, the set of stochastic transitions with exponentially distributed waiting time,*
  - *T_{im}, the set of immediate transitions with waiting time zero, and*
  - *T_{timed}, the set of transitions with deterministic waiting time.*
- *F is refined as the union of two disjoint arc sets, i.e., $F := F_N \cup F_I$ with:*
  - *$F_N \subseteq (P \times T) \cup (T \times P)$ is the set of directed standard arcs,*
  - *$F_I \subseteq P \times T$ is the set of directed inhibitor arcs.*
- *$v : TI(T_{stoch}) \rightarrow H$ is a function that assigns a stochastic hazard function $h(t(b))$ to each transition instance $t(b) \in TI(t)$ of each transition $t \in T_{stoch}$, whereby $H := \bigcup_{t(b) \in TI(T)} \{h_{t(b)} | h_{t(b)} : \mathbb{N}_0^{|^\bullet t(b)|} \rightarrow \mathbb{R}^+\}$ is the set of all stochastic hazard functions, and $v(t(b)) = h(t(b))$ for all transitions $t \in T_{stoch}$.*
- *$l : TI(T_{timed}) \rightarrow \mathbb{R}^+$ assigns a non-negative deterministic waiting time to each transition instance $t(b) \in TI(t)$ of each deterministic transition $t \in T_{timed}$.*

Please note, the stochastic hazard function in SPN^C is defined for each transition instance of each colored transition. The domain of $h(t(b))$ is restricted to the set of preplace instances of $t(b)$, denoted by $^\bullet t(b)$ with $^\bullet t(b) := \{p(c) \in PI(P) | f(p(c), t(b)) \neq 0\}$. For sake of simplicity, such features as read arcs and scheduled transitions are not explicitly mentioned in the definition above. For the semantics of SPN^C refer to [HLG+09].

Colored Petri nets, such as QPN^C and SPN^C, allow to build more compact and parametric representations of biological systems by, e.g., folding similar subnets which are then distinguished by colors. Therefore, it is possible to concisely represent complex systems that would have required a huge low-level Petri net. This provides an effective way to model and simulate very complex biological systems which would have been difficult with other modeling approaches.

3 Colored Petri net implementation in Snoopy

Snoopy is a generic and adaptive tool for modeling and animating/simulating hierarchical graph-based formalisms. Snoopy runs on Windows, Linux, and Mac operating systems. It is available free of charge for non-commercial use, and can be obtained from our website [Sno10]. However QPN^C and SPN^C are still prototypes and thus not included in the official release so far.

Snoopy provides the following functionalities for QPN^C and SPN^C:

- Rich data types for color set definition, consisting of dot, integer, string, Boolean, enumeration, index, product and union. The user can use these data types to define distinguishable tokens.

- Colored Petri net models as drawn as usual, and automatic syntax checking of declarations and expressions.
- Automatic animation, and single-step animation by manually choosing a binding. Thus, the user can run animation automatically or control the animation manually.
- Simulation is done on an automatically unfolded Petri net, and simulation results for colored or uncolored places/transitions are given together or separately. This functionality only applies to SPN^C.
- Several simulation algorithms to simulate SPN^C, including the Gillespie stochastic simulation algorithm (SSA) [Gil77].
- QPN^C and SPN^C are exported to different net formalisms, and thus can be analyzed by different tools such as CHARLIE [Fra09] and IDD-CSL [SH09].

In addition, there are some functionalities and features that are especially helpful for modeling biological systems, which are described as follows.

- Concise specification of initial markings. In a biological model, there are often large quantities of species to be modeled. So the initial markings may be set in many different ways.
  - Specifying the color and its corresponding tokens as usual.
  - Specifying a set of colors with the same number of tokens.
  - Using a predicate to choose a set of colors and then specifying the number of tokens.
  - Using the $all()$ function to specify a specific number of tokens for all colors.
- Specifying a rate function for each instance of a colored transition. For a transition, we may define different rate functions for different transition binding instances, and we use predicates to reach this goal.
- Supporting several extended arc types, such as inhibitor arc, read arc (often also called test arcs), equal arc, reset arc, and modifier arc, which are popular add-ons enhancing modeling comfort [HRR+08].
- Supporting extended transitions. Snoopy supports stochastic transitions with freestyle rate functions and rate functions of some predefined patterns as well as several deterministically timed transition types: immediate firing, deterministic firing delay, and scheduled firing (see [HLG+09] for details).

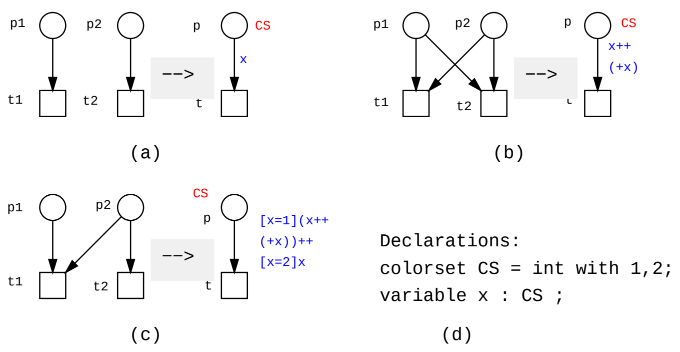
All these functionalities and features for QPN^C and SPN^C facilitate the modeling and simulation of biological systems. As a result, we not only can obtain a more compact and readable model for a complex biological system, but also do not lose simulation or analysis capabilities compared with low-level Petri nets.

4 Constructing colored Petri net models

In this section, we will demonstrate how to construct a colored Petri net model using Snoopy. We first show how to construct basic colored Petri net components, and then present two examples to illustrate QPN^C and SPN^C, respectively.

4.1 Constructing basic colored Petri net components

The key step in the design of a colored Petri net is to construct basic colored Petri net units, through which we can obtain the whole colored Petri net model step by step. This process is also called folding. In the following we will introduce some folding ways to construct basic colored Petri net components, which are illustrated in Fig. 2.



```
Declarations:
colorset CS = int with 1,2;
variable x : CS ;
```

Fig. 2. Basic colored Petri net components.

Fig. 2(a) shows the folding of two isolated subnets with the same structure. For this simple case, we can define a color set containing two colors. For example, we define the color set as "CS" with two integer colors: 1 and 2 (see Fig. 2(d)). We then assign the color set "CS" to the place. We define the arc expression as x, where x is a variable of the type "CS". Thus, we get a basic colored Petri net component, illustrated on the right hand of Fig. 2(a).

In Fig. 2(b), the net to be folded is extended by two extra arcs from p2 (p1) to t1 (t2), respectively. To fold it, we use the same color set, and just modify the arc expression to $x + +(+x)$, where the "+" in the $(+x)$ is the successor operator, which returns the successor of x in an ordered finite color set. If x is the last color, then it returns the first color. The "++" is the multiset addition operator.

In Fig. 2(c), the net to be folded gets one extra arc from p2 to t1. To fold it, we use the same color set, and just modify the arc expression to $[x = 1](x + +(+x)) + +[x = 2]x$, meaning: if $x = 1$, then there are two arcs connecting p with t, while if $x = 2$, then there is only one arc connecting p with t.

In summary, the following rules apply when folding two similar nets to a colored Petri net. If the two subnets share the same structure, we just have to define a color set and set arc expressions without predicates. If the subnets are similar, but not the same in structure, we may need to define arc expressions with predicates or guards. However, in either case, if we want to continue to add other similar nets, what we should do is usually to add new colors, and slightly change arc expressions or guards. Using these basic colored Petri net components, we can construct the whole colored Petri net model step by step.

In the next two sections, we will give two simple examples to demonstrate the application of colored Petri nets. The first example is to demonstrate QPN^C, and the second one is to demonstrate SPN^C.

4.2 Cooperative ligand binding

We consider an example of the binding of oxygen to the four subunits of a hemoglobin heterotetramer. The hemoglobin heterotetramer in the high and low affinity state binds to none, one, two, three or four oxygen molecules. Each of the ten states is represented by a place and oxygen feeds into the transitions that sequentially connect the respective places. The qualitative Petri net model is illustrated in Fig. 3 (taken from [MWW09]).

Using the folding ways demonstrated above we obtain for Fig. 3 a QPN^C model (Fig. 4), and further a more compact QPN^C model (Fig. 5). From Fig. 4, we can see that the colored Petri net model reduces the size of the corresponding low-level Petri net model. Moreover, comparing Fig. 4 with Fig. 5, we can also see that we can build colored Petri net model with different level of structural details, which is especially helpful for modeling complex biological systems. After automatic unfolding, these two colored models yield exactly the same Petri net model as given in Fig. 3, i.e., the colored models and the uncolored model are equivalent. The declarations for these two QPN^C models of the cooperative ligand binding are given in Table 1.

From these two colored nets, we can also see that the folding operation does reduce the size of the net description for the prize of more complicated inscriptions. The graphic complexity is reduced, but the annotations of nodes and edges creates a new challenge. This is not unexpected since a more concise write-up must rely on more complex components. Therefore, it is necessary to build a colored Petri net model at a suitable level of structural details.

Table 1. Declarations for the QPN^C models of the cooperative ligand binding.

Declarations
colorset Dot = dot;
colorset HbO2 = int with 0-4;
colorset Level = enum with H,L;
colorset P = product with HbO2 × Level;
variable x: HbO2;
variable y: Level;

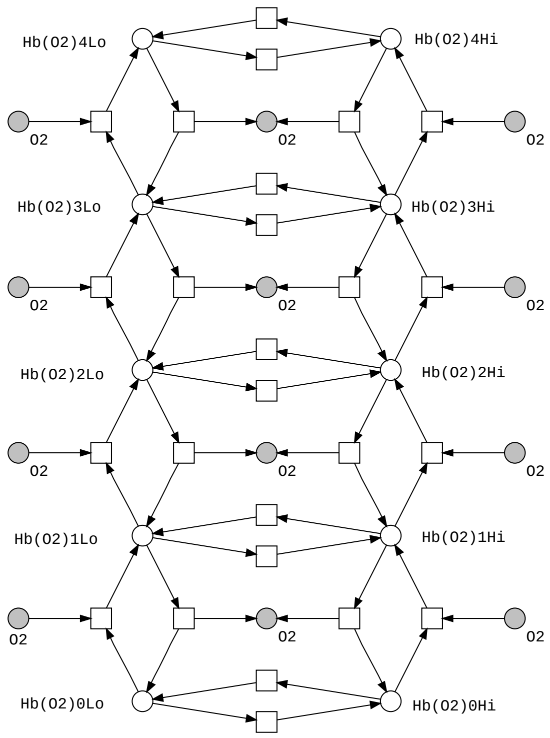


Fig. 3. Cooperative binding of oxygen to hemoglobin represented as a Petri net model. For clarity, oxygen is represented in the form of multiple copies (logical places) of one place.

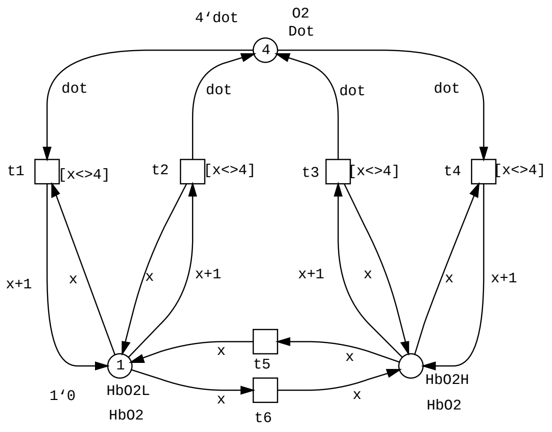


Fig. 4. QPN^C model for the cooperative binding of oxygen to hemoglobin, given as a low-level Petri net in Fig. 3. For declarations of color sets and variables, see Table 1.

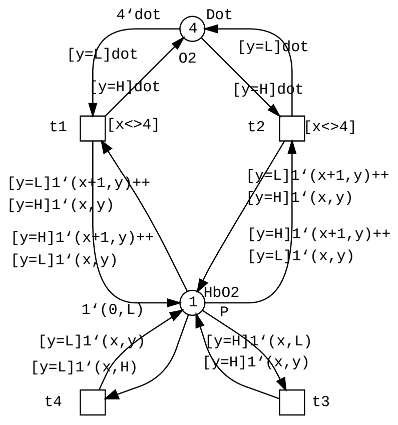


Fig. 5. QPN^C model for the cooperative binding of oxygen to hemoglobin, given as a low-level Petri net in Fig. 3. For declarations of color sets and variables, see Table 1.

4.3 Repressilator

In this section, we will demonstrate the SPN^C using an example of a synthetic circuit - the repressilator, which is an engineered synthetic system encoded on a plasmid, and designed to exhibit oscillations [EL00]. The repressilator system is a regulatory cycle of three genes, for example, denoted by g_a, g_b and g_c, where each gene represses its successor, namely, g_a inhibits g_b, g_b inhibits g_c, and g_c inhibits g_a. This negative regulation is realized by the repressors, p_a, p_b and p_c, generated by the genes g_a, g_b and g_c respectively [LB07].

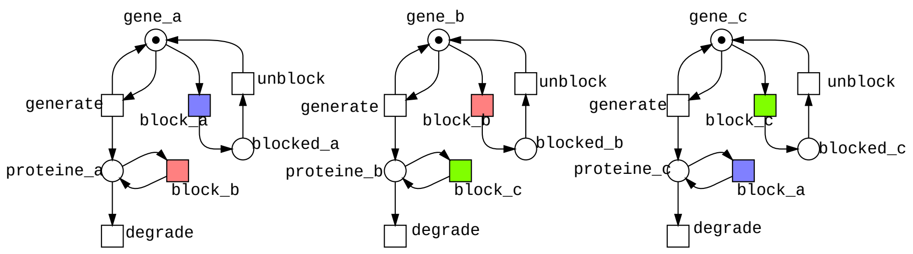


Fig. 6. Stochastic Petri net model for the repressilator. The highlighted transitions are logical transitions.

As our purpose is to demonstrate the SPN^C, we only consider a relatively simple model of the repressilator, which was built as a stochastic π-machine in [BCP08]. Based on that model, we build a stochastic Petri net model (Fig. 6), and further a SPN^C model for the repressilator (shown on the left hand of Fig.

7). This colored model when unfolded yields the same uncolored Petri net model in Fig. 6.

Table 2 gives the declarations for this SPN^C model. There are three colors, a, b, and c to distinguish three similar components in Fig. 6. The predecessor operator "-" in the arc expression $-x$ returns the predecessor of x in an ordered finite color set. If x is the first color, then it returns the last color.

As described above, the SPN^C will be automatically unfolded to a stochastic Petri net, and can be simulated with different simulation algorithms. On the right hand of Fig. 7 a snapshot of a simulation run result is given. The rate functions are given in Table 3 (coming from [PC07]). The SPN^C model exhibits the same behavior compared with that in [PC07].

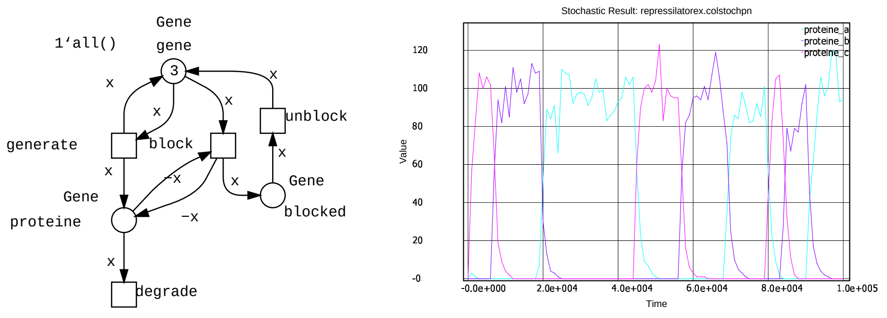


Fig. 7. SPN^C model of the low-level Petri net given in Fig. 6, and one simulation run plot for the repressilator. For rate functions, see Table 3.

Table 2. Declarations for the SPN^C model of the repressilator.

Declarations
colorset Gene = enum with a,b,c;
variable x: Gene;

From Fig. 7, we can see that the SPN^C model reduces the size of the original stochastic Petri net model to one third. More importantly, when other similar subnets have to be added, the model structure does not need to be modified and what has to be done is only to add extra colors.

For example, we consider the generalized repressilator with an arbitrary number n of genes in the loop that is presented in [MHE+06]. To build its SPN^C model, we just need to modify the color set as n colors, and do not need to

Table 3. Rate functions for the SPN^C model of the repressilator.

Transition	Rate function
generate	$0.1 * gene$
block	$1.0 * proteine$
unblock	$0.0001 * blocked$
degrade	$0.001 * proteine$

modify anything else. For example, Fig. 8 gives the conceptual graph of the generalized repressilator with $n = 9$ (on the left hand), and one simulation plot (on the right hand), whose rate functions are the same as in Table 3. Please note, the SPN^C model for the generalized repressilator is the same as the one for the three-gene repressilator, and the only difference is that we define the color set as n colors rather than 3 colors. This demonstrates a big advantage of color Petri nets, that is, to increase the colors means to increase the size of the net.

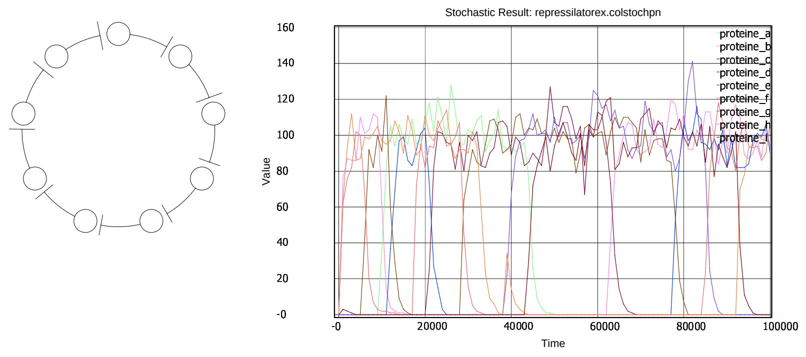


Fig. 8. Conceptual graph and one simulation run plot for the repressilator with 9 genes.

5 Related work

Heiner et al. modeled metabolic pathways with high-level Petri nets using the software packages Design/CPN [Des01]. Colors discriminate metabolites, and thus they got a number of valuable insights by combining symbolic analysis and simulation for colored metabolic steady state system models [HKV01]. Genrich et al. discussed the steps to establish and tune high-level net models, and modeled metabolic pathways of the glycolysis and citric acid cycle with colored Petri nets using also Design/CPN. By assigning enzymatic reaction rates to the transitions, they implemented the simulation and quantitative study of networks of metabolic processes [GKV01]. Bahi-Jaber et al. investigated the application of colored stochastic Petri nets to epidemic models using a very simple model [BP03]. Although this study had no tool support, it really demonstrated the advantages of colored stochastic Petri nets. Runge described a systematic semi-automatic procedure, exploiting the place/transition net's T-invariants to construct an equivalent bounded and live coloured net. As case study, an extended glycolysis was used [Run04]. However, he only considered modeling and qualitative analysis of biological model based on CPN tools [JKW07]. Lee et al. built a colored Petri net model for the signal transduction system stimulated by epidermal growth factor (EGF) based on CPN tools, in which they use the conservation and kinetic equations to quantitatively examine the dynamic behavior of the EGF signaling pathway [LYL+06]. Tubner et al. used the UML class diagram to understand the static structure of molecules involved in the TLR4 pathway, and then modeled and simulated the TLR4 pathway to get the behavior of the system with colored Petri nets based on CPN tools [TMK+06]. In their model, they did not consider any time information.

In summary, the existing studies usually resort to Design/CPN or its successor CPN tools to realize the modeling and analysis of biological systems. But CPN tools are not designed for modeling and analyzing biological systems. So it is not suitable in many aspects, like rate function definition and simulative analysis by stochastic simulation algorithms.

In contrast, in Snoopy we provide specific functionalities and features to support editing, simulating, and analyzing of biological models based on colored Petri nets, as shown in Section 3.

6 Conclusion and outlook

In this paper we have described our on-going work of a colored Petri net-based framework to model, simulate, and analyze complex biological systems. This framework consists of three parts: QPN^C, SPN^C, and CPN^C, and only the first two parts have been described in this paper. Their definitions are given, and functionalities and features implemented in Snoopy are described, followed by two examples to demonstrate their application. The colored Petri nets allow a more concise representation of biological systems, making it possible and convenient to construct and analyze large-scale biological models.

We are working on improvements of these two paradigms: QPN^C and SPN^C. In the next step, we will focus on the development of analysis tools for SPN^C, and we will include the CPN^C in our work. We are also developing a method to automatically create colored Petri nets from non-coloured Petri nets (automatic folding). This development will provide much stronger support to construct and analyze large-scale biological models. Besides, we are working on a case study with a size of the underlying uncolored model of about 110,000 places and 135,000 transitions.

Acknowledgments

This work has been supported by the Modeling of Pain Switch (MOPS) program of Federal Ministry of Education and Research (Funding Number: 0315449H). We would like to thank Wolfgang Marwan, Christian Rohr, and Martin Schwarick for their assistance in model construction and software development. We also would like to thank the anonymous referees for many constructive comments.

References

[BCP08] R. Blossey, L. Cardelli, A. Phillips: Compositionality, Stochasticity and Cooperativity in Dynamic Models of Gene Regulation. HFSP Journal. 2(1), 17-28 (2008)

[BP03] N. Bahi-Jaber, D. Pontier: Modeling Transmission of Directly Transmitted Infectious Diseases Using Colored Stochastic Petri Nets. Mathematical Biosciences. 185, 1-13 (2003)

[Des01] Design/CPN: http://www.daimi.au.dk/designCPN. (2001)

[EL00] M. B. Elowitz, S. Leibler: A Synthetic Oscillatory Network of Transcriptional Regulators. Nature. 403, 335-338 (2000)

[Fra09] A. Franzke: Charlie 2.0 - a Multi-Threaded Petri Net Analyzer. Diploma Thesis, Brandenburg University of Technology Cottbus. (2009)

[Ger01] R. German: Performance Analysis of Communication Systems With Non-Markovian Stochastic Petri Nets. John Wiley and Sons Ltd. (2001)

[GH06] D. Gilbert, M. Heiner: From Petri Nets to Differential Equations - an Integrative Approach for Biochemical Network Analysis. Proc. ICATPN. LNCS, 4024, 181-200 (2006)

[GHL07] D. Gilbert, M. Heiner, S. Lehrack : A Unifying Framework for Modelling and Analysing Biochemical Pathways Using Petri Nets. Proc. International Conference on Computational Methods in Systems Biology. LNCS/LNBI, 4695, 200-216 (2007)

[GKV01] H. Genrich, R. Kffner, K. Voss: Executable Petri Net Models for the Analysis of Metabolic Pathways. International Journal on Software Tools for Technology Transfer. 3(4), 394-404 (2001)

[Gil77] D. T. Gillespie: Exact Stochastic Simulation of Coupled Chemical Reactions. Journal of Physical Chemistry. 81(25), 2340-2361 (1977)

[HGD08] M. Heiner, D. Gilbert, R. Donaldson: Petri Nets for Systems and Synthetic Biology. In Schools on Formal Methods (SFM), LNCS, 5016, 215-264 (2008)

[HK09] A. P. Heath, L. E. Kavraki: Computational Challenges in Systems Biology. Computer Science Review. 3(1), 1-17 (2009)

[HKV01] M. Heiner, I. Koch, K. Voss: Analysis and Simulation of Steady States in Metabolic Pathways with Petri Nets. Proc. Third Workshop CPN. Univ. of Aarhus, 15-34 (2001)

[HLG+09] M. Heiner, S. Lehrack, D. Gilbert, W. Marwan: Extended Stochastic Petri Nets for Model-based Design of Wetlab Experiments. Transaction on Computational Systems Biology XI. LNCS/LNBI, 5750, 138-163 (2009)

[HRR+08] M. Heiner, R. Richter, C. Rohr, M. Schwarick: Snoopy - A Tool to Design and Execute Graph-Based Formalisms. [Extended Version]. Petri Net Newsletter. 74, 8-22 (2008)

[Jen81] K. Jensen: Coloured Petri Nets and the Invariant-Method. Theor. Comput. Sci. 14, 317-336 (1981)

[JKW07] K. Jensen, L. M. Kristensen, L. M. Wells: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. International Journal on Software Tools for Technology Transfer. 9(3/4) 213-254 (2007)

[JP98] P. J. E. Goss, J. Peccoud: Quantitative Modeling of Stochastic Systems in Molecular Biology by Using Stochastic Petri Nets. Proc. Natl. Acad. Sci. 95, 6750-6755 (1998)

[LB07] A. Loinger, O. Biham: Stochastic Simulations of the Repressilator Circuit. Phisical Review. 76(5), 051917(9) (2007)

[LYL+06] D. Lee, R. Zimmer, S. Lee, S. Park: Colored Petri Net Modeling and Simulation of Signal Transduction Pathways. Metabolic Engineering. 8, 112-122 (2006)

[MBC+95] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis: Modelling with Generalized Stochastic Petri Nets. Wiley Series in Parallel Computing. John Wiley and Sons, 2nd Edition. (1995)

[MHE+06] S. Muller, J. Hofbauer, L. Endler, C. Flamm, S. Widder, P. Schuster: A Generalized Model of the Repressilator. Journal of Mathematical Biology. 53, 905-937 (2006)

[Mur89] T. Murata: Petri Nets: Properties, Analysis and Applications. Proc.of the IEEE. 4, 541-580 (1989)

[Mur07] I. Mura: Simplifying the Stochastic Petri Nets Formalism for Representing Biological Phenomena. Technical Report TR-19-2007, CoSBi (Center for Computational and Systems Biology), University of Trento (2007)

[MWW09] W. Marwan, A. Wagler, R. Weismantel: Petri Nets as a Framework for the Reconstruction and Analysis of Signal Transduction Pathways and Regulatory Networks. Natural Computing. 9 (2009)

[NOG+05] T. Nutsch, D. Oesterhelt, E. D. Gilles, W. Marwan: A Quantitative Model of the Switch Cycle of an Archaeal Flagellar Motor and Its Sensory Control. Biophysical Journal. 89(4), 2307-2323 (2005)

[PC07] A. Phillips, L. Cardelli: Efficient, Correct Simulation of Biological Processes in the Stochastic Pi-calculus. Proc. Computational Methods in Systems Biology. LNCS, 4695, 184-199 (2007)

[PRA05] M. Peleg, D. Rubin, R. B. Altman: Using Petri Net Tools to Study Properties and Dynamics of Biological Systems. Journal of the American Medical Informatics Association. 12(2), 181-199 (2005)

[RML93] V. N. Reddy, M. L. Mavrovouniotis, M. N. Liebman: Petri Net Representation in Metabolic Pathways. Proc. First International Conference on Intelligent System for Molecular Biology. 328-336 (1993)

[Run04] T. Runge: Application of Coloured Petri Nets in Systems Biology. Proc. 5th Workshop CPN. 77-95 (2004)

[SH09] M. Schwarick, M. Heiner: CSL Model Checking of Biochemical Networks with Interval Decision Diagrams. Proc. 7th International Conference on Computational Methods in Systems Biology (CMSB 2009), Springer LNBI 5688, 296-312 (2009)

[Sno10] Snoopy Website: Snoopy - Tool to Design and Animate/Simulate Graphs. http://www-dssz.informatik.tu-cottbus.de/software/snoopy.html (2010)

[TMK+06] C. Tubner, B. Mathiak, A. Kupfer, N. Fleischer, S. Eckstein: Modelling and Simulation of the TLR4 Pathway with Coloured Petri Nets. Proc. IEEE EMBS Annual International Conference. 2009-2012 (2006)

Control of Metabolic Systems Modeled with Timed Continuous Petri Nets

Roberto Ross-León[1], Antonio Ramirez-Treviño[1], José Alejandro Morales[2], and Javier Ruiz-León[1]

[1] Centro de Investigaciones y Estudios Avanzados del I.P.N. Unidad Guadalajara
`{rross,art,jruiz}@gdl.cinvestav.mx`
[2] Centro Universitario de Ciencias Exactas e Ingenierías, Universidad de Guadalajara
`alejandro.morales@cucei.udg.mx`

Abstract. This paper is concerned with the control problem of biological systems modeled with Timed Continuous Petri Nets under infinite server semantics. This work introduces two main contributions. The first one is a bottom-up modeling methodology that uses $TCPN$ to represent cell metabolism.
The second contribution is the control wich solves the Regulation Control Problem (RCP) (to reach a required state and maintain it). The control is based on a Lyapunov criterion that ensures reaching the required state.

Key words: Cell metabolome, Petri nets, Controllability, Stability.

This work was supported by project No. 23-2008-335, COETCYJAL-CEMUE, Mexico. R. Ross-León was supported by CONACYT, grant No. 13527.

1 Introduction

Petri nets PN [1], [2], [3] are a formal paradigm for modelling and analysis of systems that can be seen as discrete dynamical systems. Unfortunately, due to state explosion problem, most of the analysis techniques cannot be applied in heavy marked Petri nets. In order to overcome this problem, the Petri net community developed the Timed Continuous Petri Nets ($TCPN$) [4], [5], a relaxation of the Petri Nets where the marking becomes continuous and the state equation is represented by a positive, bounded set of linear differential equations.

The main $TCPN$ characteristics such as the nice pictorially representation, the mathematical background, the synchronization of several products to start an activity and the representation of causal relationship make $TCPN$ amenable to represent biochemical reactions and cell metabolism. In fact $TCPN$ marking captures the concentration of molecular species while differential equations together with the firing vectors represent the reaction velocity and the graph captures the metabolic pathways. The entire $TCPN$ captures the cell metabolome.

Several works model [6], [7], analyse [8], [9] and control [10], [11] metabolic pathways. Most of them deal with pseudo-steady states of the biochemical reaction dynamic. Nowadays, the scientific community is exploring the use of PN and their extensions [12], [13] to model biological systems since the former are able to capture the compounds flow, the reaction velocity, the enabling/inhibiting reactions and both the transitory and steady states of reaction dynamic into a single formalism.

This work is concerned on how to model the entire metabolome with $TCPN$. It proposes a bottom-up modeling methodology where biochemical reactions are modeled through elementary modules, and shows how these modules are merged to form metabolic pathways, and at the end the cell metabolism. The resulting model captures both, the transitory and steady state metabolome dynamics. It is worth noticing that the derived $TCPN$ model condenses several particular behaviors represented by the set of differential equations generated by the $TCPN$ itself. For instance, a single transition with four input places (a reaction needing four substrates) generates a set of four possible differential equations while two transitions with four input places each will generate a set of sixteen possible differential equations. Therefore highly complex behaviors emerging from few compounds interacting can be captured by $TCPN$.

This work also presents the control problem of reaching a required state (marking) representing a certain metabolite concentration. In order to solve this problem, an error equation is stated and stabilized using a Lyapunov approach. The solution is the reaction rate vector which is greater or equal to zero and lower or equal to the maximum settled by the kinetics of Michaelis-Menten for the current enzyme concentration. Thus, if a solution exists, it could be implemented *in vivo* by directed genetic mutation, knock-in (or knock-out) strategies or pharmacological effects.

Present paper is organized as follows. Section 2 gives $TCPN$ basic definitions, controllability and cell metabolic concepts. Next section introduces the proposed metabolome modeling methodology. Section 4 presents the problem of reaching a required state and synthesizes Lyapunov like transition flow for solving this problem. Following section presents an illustrative example to show the performance of the computed control law. In the last section the conclusions and future work are presented.

2 Basic Definitions

This section presents briefly the basic concepts related with PN, *Continuous* PN and $TCPN$. An interested reader can review [3], [14], [15] and [16] for further information. At the end of this section a useful form of the state equation for $TCPN$ under infinite server semantics is presented.

2.1 Petri Net concepts

Definition 1. *A Continuous Petri Net (ContPN) system is a pair* (N, m_0), *where* $N = (P, T, Pre, Post)$ *is a Petri net structure (PN) and* $m_0 \in \{\mathbb{R}^+ \cup 0\}^{|P|}$ *is the initial marking.* $P = \{p_1, ..., p_n\}$ *and* $T = \{t_1, ..., t_k\}$ *are finite sets of elements named places and transitions, respectively.* $Pre, Post \in \{\mathbb{N} \cup 0\}^{|P| \times |T|}$

are the Pre and Post incidence matrices, respectively, where $Pre[i,j]$, $Post[i,j]$ represent the weights of the arcs from p_i to t_j and from t_j to p_i, respectively. The Incidence matrix denoted by C is defined by $C = Post - Pre$.

Each place p_i has a marking denoted by $m_i \in \{\mathbb{R}^+ \cup 0\}$. The set ${}^\bullet t_i = \{p_j \mid Pre[j,i] > 0\}$, $(t_i^\bullet = \{p_j \mid Post[j,i] > 0\})$ is the preset (postset) of t_i. Similarly the set ${}^\bullet p_i = \{t_j \mid Post[i,j] > 0\}$, $(p_i^\bullet = \{t_j \mid Post[i,j] > 0\})$ is the preset (postset) of p_i.

A transition $t_j \in T$ is enabled at marking m iff $\forall p_i \in {}^\bullet t_j$, $m_i > 0$. Its enabling degree is:

$$enab(t_j, m) = \min_{p_i \in {}^\bullet t_j} \frac{m_i}{Pre\,[i,j]} \tag{1}$$

and it is said that m_i constraints the firing of t_j. Equation (1) denotes the maximum amount that t_j can be fired at marking m; indeed t_j can fire in any real amount α, where $0 < \alpha < enab(t_j, m)$ leading to a new marking $m\prime = m + \alpha C[\bullet, j]$. If m is reachable from m_0 through a finite sequence σ of enabled transitions, then m can be computed with the equation:

$$m = m_0 + C\sigma \tag{2}$$

named the *ContPN* state equation, where $\sigma \in \{\mathbb{R}^+ \cup 0\}^{|T|}$ is the firing count vector, i.e., σ_j is the cumulative amount of firing of t_j in the sequence σ. The set of all reachable markings from m_0 is called the reachability space and it is denoted by $RS\,(N, m_0)$. In the case of a *ContPN* system, $RS\,(N, m_0)$ is a convex set [17].

Definition 2. *A contPN is bounded when every place is bounded ($\forall p \in P, \exists b_p \in \mathbb{R}$ with $m\,[p] \le b_p$ at every reachable marking m). It is live when every transition is live (it can ultimately occur from every reachable marking). Liveness is extended to lim-live when infinitely long sequence can be fired. A transition t is non lim-live iff a sequence of successively reachable markings exists which converge to a marking such that none of its successors enables a transition t.*

2.2 Timed continuous Petri nets

Definition 3. *A timed ContPN is the 3-tuple $TCPN = (N, \lambda, m_0)$, where N is a ContPN, $\lambda : T \to \{\mathbb{R}^+\}^{|T|}$ is a function that associates a maximum firing rate to each transition, and m_0 is the initial marking of the net N.*

The state equation of a $TCPN$ is

$$\overset{\bullet}{m}(\tau) = Cf(\tau) \tag{3}$$
$$\text{where } f(\tau) = \overset{\bullet}{\sigma}(\tau)$$

And under the infinite server semantics, the flow of transition t_j is given by

$$f_j(\tau) = \lambda_j enab(t_j, m(\tau)) \tag{4}$$

where λ_j represents the maximum firing rate of transition t_j. Notice that $TCPN$ under infinite server semantics is a piecewise linear system (a class of hybrid systems) due to the *minimum* operator that appears in the enabling function of the flow definition.

Definition 4. *A configuration of a $TCPN$ at m is a set of (p,t) arcs describing the effective flow of all transitions.*

$$\Pi\left(m\right)\left[i,j\right] = \begin{cases} \frac{1}{Pre[i,j]} & \textit{if } p_i \textit{ is constraining } t_j \\ 0 & \textit{otherwise} \end{cases} \tag{5}$$

Definition 5. *The maximum firing rate matrix is denoted by*

$$\Lambda = diag\left(\lambda_1, \ldots, \lambda_{|T|}\right). \tag{6}$$

According to previous notation, the state equation and the flow vector are described by:

$$\begin{array}{c} \overset{\bullet}{m} = C\Lambda\Pi\left(m\right)\cdot m \\ f = \Lambda\Pi\left(m\right)\cdot m \end{array} \tag{7}$$

The only action that can be applied to a $TCPN$ system is to slow down the firing flow. The forced flow of a controlled transition t_i becomes $f_i - u_i$ where f_i is the flow of the unforced system (i.e. without control) and u is the control action, with $0 \le u_i \le f_i$. The controlled state equation is:

$$\overset{\bullet}{m} = C\left[\Lambda\Pi\left(m\right)\cdot m - u\right] \tag{8}$$

$$0 \le u_i \le \left[\Lambda\Pi\left(m\right)\cdot m\right]_i \tag{9}$$

In order to obtain a simplified version of the state equation, the input vector u is rewritten as $u = I_u\Lambda\Pi\left(m\right)\cdot m$, where $I_u = diag\left(I_{u_1}, \ldots, I_{u_{|T|}}\right)$ and $0 \le I_{u_i} \le 1$. Then the matrix $I_c = I - I_u$ is constructed and the controlled state equation can be rewritten as:

$$\overset{\bullet}{m} = CI_c\Lambda\Pi\left(m\right)\cdot m \tag{10}$$

Notice that $0 \le I_{c_i} \le 1$.

2.3 Controllability

The classical linear systems definition of controllability cannot be applied to $TCPN$ systems because the required hypothesis are not fulfilled, that is, the input should be unbounded and the state space should be $\mathbb{R}^{|P|}$. The next definitions are taken from [18].

Definition 6. *Let N be net of a $TCPN$. The structural admissible states set is defined as $SASS\left(N\right) = \left\{\mathbb{R}^+ \cup \{0\}\right\}^{|P|}$ (all inital markings that can be imposed to a net). Let B be the base of the left annuller of the incidence matrix C. The equivalence relation $\beta : SASS\left(N\right) \rightarrow SASS\left(N\right)$ is defined as $m_1\beta m_2$ iff $B^T m_1 = B^T m_2$, $\forall m_1, m_2 \in SASS(N)$. The system admissible states set is the equivalent class of the initial marking $Class\left(m_0\right)$ under β.*

In the sequel, let us denote by $int\left(Class\left(m_0\right)\right)$ the set of relative interior of $Class\left(m_0\right)$.

Definition 7. *Let (N, λ, m_0) be a TCPN system. It is fully controllable with bounded input ($BIFC$) if there is an input such that for any two markings $m_1, m_2 \in Class\left(m_0\right)$, it is possible to transfer the marking from m_1 to m_2 in finite or infinite time, and the input fulfills (9) along the trajectory, and is controllable with bounded input (BIC) over $S \subseteq Class\left(m_0\right)$ if there is an input such that for any two markings $m_1, m_2 \in S$, it is possible to transfer the marking from m_1 to m_2 in finite or infinite time, and the input fulfills (9) along the trajectory.*

Definition 8. *Let (N, λ, m_0) be a TCPN system. Let $m_r \in RS\left(N, m_0\right)$ and $0 \leq I_{c_r}[i,i] \leq 1$. Then (m_r, I_{c_r}) is an equilibrium point if $\dot{m}_r = CI_{c_r}\Lambda\Pi\left(m_r\right) \cdot m = 0$. Then, the steady state flow for (m_r, I_{c_r}) is $f_{ss}\left(m_r, I_{c_r}\right) = I_{c_r}\Lambda\Pi\left(m_r\right) \cdot m_r$.*

An equilibrium point represents a state in which the system can be maintained using the defined control action. Given an initial marking m_0 and a required marking m_r, one control problem is to reach m_r and then keep it. For a further information about equilibrium points an interested reader can review [19].

2.4 Cell Metabolism

For the wellbeing of an given organism, each cell of that organism must transform the substances available in its surroundings to useful molecules. Such transformations take place as chemical reactions catalyzed by enzymes. In these reactions, a substrate tightly binds non-covalently to its enzyme active site to build an enzyme-substrate complex. At that moment, the enzyme chemically changes the substrate into one or more products and then releases it. The enzyme did not suffer any irreversible alterations in the process, and now is free to accept a new substrate [20].

There is no limit to the number of possible reactions occurring in nature. Nonetheless, after exhaustive analysis certain general patterns had emerged that became useful to describe several characteristics of biochemical reactions. In the case where a sole substrate becomes a single product, the reaction process is represented by the scheme:

$$E + S \rightleftharpoons ES \rightarrow EQ \rightleftharpoons E + Q \tag{11}$$

where E is the enzyme, S is the substrate, ES and EQ are the bound complexes and Q is the product.

Typically, the rate of these reactions is settled by the kinetics of Michaelis-Menten [21]. Under this kinetic model, the enzyme and substrate react rapidly to form an enzyme-substrate complex while $[S]$ and $[ES]$ are considered to be at concentration equilibrium (the same applies to $[EQ]$ and $[Q]$), that is, the rate at which ES dissociates into $E + S$ is much faster than the rate at which ES brakes down to EQ.

Throughout the present work, we will consider a physiological cellular state where $[S] >> [E]$, which means that $[S] \rightleftharpoons [ES]$ equilibrium will always tend to complex formation. Therefore, ES dissociation rate is irrelevant and Scheme 11 can be abbreviated as follows:

$$E + S \rightarrow E + Q \tag{12}$$

where the association-dissociation is implicit.

In reactions with more than one substrate, binding can occur in different sequences; for instance, the following scheme represents an enzyme system with two substrates and all the possible sequences:

$$E + S_1 + S_2 \rightleftharpoons \begin{cases} ES_1 + S_2 \\ \quad \upharpoonleft\downharpoonright \\ ES_1S_2 \rightarrow \ E + Q_1 + ... + Q_n \\ \quad \upharpoonleft\downharpoonright \\ ES_2 + S_1 \end{cases} \tag{13}$$

Frequently the product of an enzyme is the substrate of another reaction and so on, to build a chain of reactions called metabolic pathways represented by $MP^j = \Gamma_1^j \Gamma_2^j \cdots \Gamma_n^j$ where Γ_i^j is a reaction (12) or (13) of a pathway j and Γ_k^j uses one or more products of Γ_i^m. Notice that j and m may represent different pathways.

Then a (Cell) Metabolome is $CM = \{MP^i \,|MP^i \text{ is a metabolic pathway}\}$, and the purpose of CM is to produce a particular set of metabolites in certain concentrations, essential to that cell.

3 Modelling the Metabolome

In order to model the metabolome using $TCPN$ it is necessary to identify how the elements involved in it will be represented. The next table relates the meaning of each element of the $TCPN$ with respect to metabolic reactions.

TCPN term	**Molecular interpretation**
Place	Molecular Species
Marking	Concentration
Transition	Reaction
Firing Rate	Rate of Reaction
Arc Weights	Stoichiometric Coefficients

The bottom-up approach herein proposed to model the metabolome consists of: a) representing reactions, the results of this stage are the elementary modules; b) merging elementary modules, where places of elementary modules representing the same molecular species on the same physical space in the cell will merge

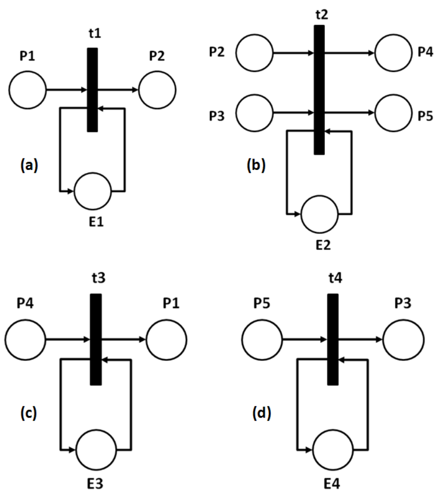


Fig. 1. Four elementary modules representing four diferent reactions.

into a single place. The results of this stage are pathway modules; and c) merging pathways modules, where places of pathways modules representing the same molecular species on the same cellular space will merge into a single place; the result of this stage is the metabolome model. For stages b and c, any specie being protein-mediated transported into a different organelle shall be modeled through the same elementary module, representing instead of substrate and product the same molecule in different spaces.

Next section describes these stages.

3.1 Representing Reactions

In order to represent each reaction $\mathit{\Gamma}_i$ with $TCPN$ elementary modules representing the Scheme (12) or (13) are constructed. There exists one place p_j for each molecular species at the same physical space ms_j and one transition t_i to represent the reaction $\mathit{\Gamma}_i$. There exists one arc (p_s, t_i) if p_s represents a substrate. There exists one arc (t_i, p_q) if p_q represents a product. Finally, there exists a self-loop around p_e and t_i if p_e represents an enzyme. The initial marking $m_0\,[p_j]$ is the concentration of the molecular species ms_j at time $\tau = 0$.

Associated to transition t_i is λ_i representing the rate of reaction.

Example 1. Let $P1 + E1 \rightarrow P2 + E1$ be the $\mathit{\Gamma}_1$ reaction. There is one place for each molecular species ($P1$, $P2$ and $E1$), and one transition t_1 representing $\mathit{\Gamma}_1$. Finally, arcs are fixed in the way depicted in Figure 1a.

Assuming that the substrate concentration will remain higher than the enzyme concentration (this is an expected behavior of the system), the conflict

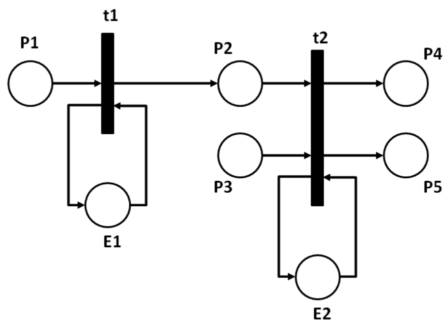


Fig. 2. Example of a Pathway Module.

between substrate and enzyme can be ignored. Hence, if a system has 2^n configurations originated by the n number of enzymes in conflict with substrates, all those configurations are eliminated because $\min([E],[S]) = [E]$ for all $\tau \geq 0$.

3.2 Merging Elementary Modules

Let N^1 and N^2 be two elementary modules, then the merging N is such that $N = (P, T, Pre, Post)$ where $P = P^1 \cup P^2$, $T = T^1 \cup T^2$, $Pre = Pre^1 \cup Pre^2$ and $Post = Post^1 \cup Post^2$. Notice that places representing the same molecular species in the same physical space are merged into a single place.

After a merging of elementary modules is made, pathway modules are obtained.

Example 2. Let $N^1 = \left(P^1, T^1, Pre^1, Post^1\right)$ and $N^2 = \left(P^2, T^2, Pre^2, Post^2\right)$ be two elementary modules showed in Figure 1a and Figure 1b respectively. Then, the merging is $N = (P, T, Pre, Post)$ where $P = P^1 \cup P^2 = \{P1, ..., P5, E1, E2\}$, $T = T^1 \cup T^2 = \{t_1, t_2\}$ and arcs are fixed in the way depicted in Figure 2, where the merging is showed.

3.3 Merging Pathway Modules

Let N^1 be a pathway module and N^2 be a pathway or an elementary module, then the merging N is such that $N = (P, T, Pre, Post)$ where $P = P^1 \cup P^2$, $T = T^1 \cup T^2$, $Pre = Pre^1 \cup Pre^2$ and $Post = Post^1 \cup Post^2$. Notice that places representing the same molecular species are merged into a single place.

After a merging of pathway modules is made, a metabolic model is obtained.

Example 3. Let $N^1 = \left(P^1, T^1, Pre^1, Post^1\right)$ be the pathway module showed in Figure 2. Let $N^2 = \left(P^2, T^2, Pre^2, Post^2\right)$ and $N^3 = \left(P^3, T^3, Pre^3, Post^3\right)$ be two elementary modules showed in Figure 1c and Figure 1d respectively. Then

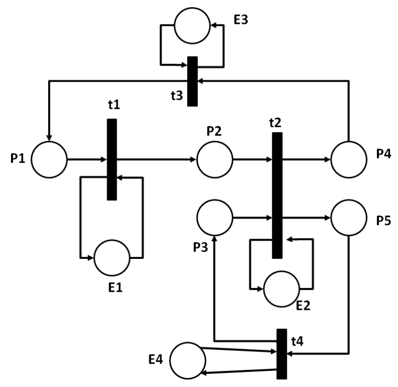


Fig. 3. Metabolic Model.

the merging is $N = (P, T, Pre, Post)$, where $P = \cup P^i = \{P1, ..., P5, E_1, ..., E_4\}$, $T = \cup T^i = \{t_1, t_2, t_3, t_4\}$ for $i = 1, ..., 4$. Arcs are fixed in Figure 3, where the merging is showed.

Although obtained metabolic models could be not live, the addition of a virtual transition and arcs going from the last place representing final products to the virtual transition and from virtual transition to the places representing initial products with an appropriate virtual reaction velocity will make the metabolic model live. For instance, consider the net of Figure 1a, it is a non-live net, but if we add a virtual input transition t_v to the place S and a virtual output transition t_v to the place Q the system will gain liveness, see Figure 4. Notice that t_v must to be the same transition added to the inital and final metabolites, this is because it is necessary to maintain the conservativeness of the matter of the system. This notion is based assuming that each module belongs to a bigger system, therefore, although the real input and output transitions could be not the same, they must have the same firing ratio.

4 Control Law

An important control problem in the metabolic engineering area is to reach a certain metabolome state such that the production of selected metabolites is regulated or particular processes are limited or favored. This problem is captured in $TCPN$ as the reachability problem, i.e. to reach a required state m_r from an initial state m_0 by means of an appropriate control action. This is formalized as follows.

Definition 9. *Let $TCPN$ be a metabolic model. Then the* `Regulation Control Problem` *in (m_r, I_{c_r}) $(RCP(m_r, I_{c_r}))$ deals with the computation of a control law*

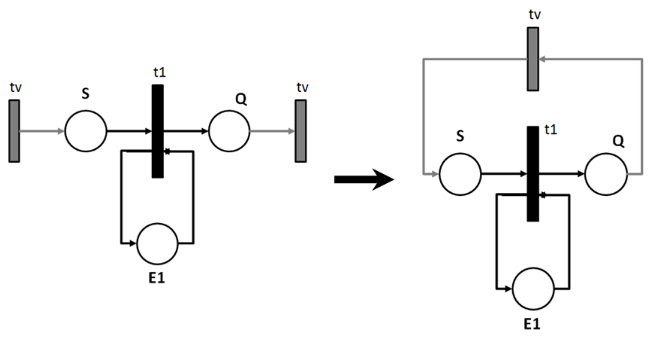


Fig. 4. Module forced to be live with the addition of a virtual transition t_v (in gray).

$I_c(\tau)$, $0 \le \tau < \tau_f$ feasible in the $TCPN$ such that $m(\tau_{ss}) = m_r$ and $I_c(\tau_{ss}) = I_{c_r}$, $\forall \tau_{ss} \ge \tau_f$.

In order to solve this problem, some extra places are added to the $TCPN$ metabolic model to detect the material passing through transitions. The following definition shows how these places are added to the $TCPN$.

Definition 10. *Let (N, m_0, λ) be a metabolic model $TCPN$, where $N = (P, T, F)$. Its extension is defined by $xTCPN = (N_x, m_{0_x}, \lambda)$, where $N_x = (P \cup P_a, T, F \cup F_a)$, $|P_a| = |T|$, $m_{0_x} = \left[m_0 \; 0_{|T|}\right]^T$, $F_a = \{(t_i, p_{a_i}) \,|\forall t_i \in T$ and $\forall p_{a_i} \in P_a\}$. Then the incidence matrix of $xTCPN$ is $C_x = \left[C \; I_{|T|}\right]^T$.*

Since $\Pi_x(m_x) = \left[\Pi(m) \; 0_{|T| \times |T|}\right]$, then the state equation of $xTCPN$ is:

$$\overset{\bullet}{m}_x = \begin{bmatrix} \overset{\bullet}{m} \\ \overset{\bullet}{m}_a \end{bmatrix} = \begin{bmatrix} C I_c \Lambda \Pi(m) \cdot m \\ I_c \Lambda \Pi(m) \cdot m \end{bmatrix} \tag{14}$$

$$m(0) = m_0, \; m_a(0) = 0 \tag{15}$$

Remark 1. Notice that the extension has the same dynamic over the metabolic model places and the extra places can only increase its marking. In fact, due to the $TCPN$ is live, then by construction the $xTCPN$ is also live. Then there exists at least one enabled transition. Hence $\Pi(m) \cdot m > \mathbf{0}$ (or equivalently $\overset{\bullet}{m}_a \ge \mathbf{0}$, the zero could be forced by an appropriate control law I_c).

Example 4. An example of an extended net is presented in Figure 5.

4.1 Solution to the $\boldsymbol{RCP}$ $(\boldsymbol{m_r, I_{c_r}})$

Theorem 1. *Let (N, m_0, λ) be a metabolic model $TCPN$ and let $xTCPN = (N_x, m_{0_x}, \lambda)$ be its extension. If (N, m_0, λ) is BIC over $int(Class(m_0))$ (notice*

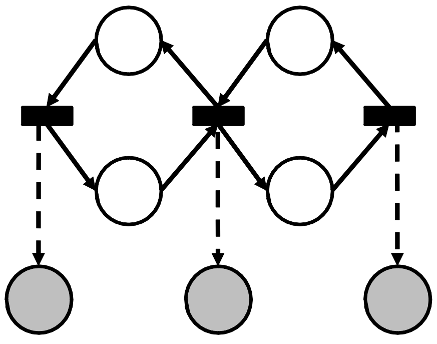


Fig. 5. Example of an extended net. The gray places are the set of added places P_a and the intermittent arrows are the set of added arcs F_a.

that neither the initial marking nor the required marking could be zero components) and (I_{c_r}, m_r) *is an arbitrary equilibrium point, then there exists* $I_c(\tau)$, $0 \leq \tau \leq \tau_f$ *feasible in the TCPN such that* $m(\tau_{ss}) = m_r$, $I_c(\tau_{ss}) = I_{c_r}$, $\forall \tau_{ss} \geq \tau_f$.

Proof. If the system is BIC over $int\,(Class\,(m_0))$, then there exists a positive solution $\sigma_r(\tau)$ feasible such that

$$m_r = m_0 + C\sigma_r \tag{16}$$

This result was taken from [18]. Thus there exists $f(\tau)$ such that:

$$\int_0^{\tau_f} f(\tau)d\tau = \int_0^{\tau_f} I_c \Lambda \Pi\,(m) \cdot m d\tau = \sigma_r \tag{17}$$

From (14):

$$m_a(\tau_f) = \sigma_r \tag{18}$$

Now, let

$$\begin{aligned} e_x(\tau) &= \left[e(\tau)\ e_a(\tau) \right]^T, 0 \leq \tau \leq \tau_f \\ \left[e(\tau)\ e_a(\tau) \right]^T &= \left[m_r - m(\tau)\ \sigma_r - m_a(\tau) \right]^T \end{aligned} \tag{19}$$

and

$$V\,(e_x) = e_x^T P_L e_x \tag{20}$$

where

$$P_L = \begin{bmatrix} 0 & 0 \\ 0 & I_{|T|} \end{bmatrix} \tag{21}$$

and $I_{|T|}$ is an identity matrix of order $|T| \times |T|$. We claim that $V\,(e_x)$ is a Lyapunov function, i.e it is positive definite and its derivative is negative definite.

Since Equation (20) is clearly non-negative definite, then we assume that (22) is positive semidefinite, then there exists $e_x(\tau') \neq 0$ such that:

$$V\left(e_x(\tau')\right) = \begin{bmatrix} e^T \ e_a^T \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & I_{|T|} \end{bmatrix} \begin{bmatrix} e \\ e_a \end{bmatrix} = 0 \tag{22}$$

From (22), it is clear that $e_a(\tau') = 0$, then from (19) $m_a(\tau') = \sigma_r$. Thus, from (14) and (17) and letting $\tau_f = \tau'$:

$$\int_0^{\tau_f} \overset{\bullet}{m}(\tau) d\tau = C\sigma_r \tag{23}$$
$$m(\tau_f) - m(0) = C\sigma_r$$

Thus, from Equation 16 $m(\tau_f) = m_r$, then $e_x = 0$, a contradiction. Hence $V(e_x)$ is positive definite.

Now , we prove that $\overset{\bullet}{V}(e_x)$ is negative definite. The differentiate of $V(e_x)$ is:

$$\overset{\bullet}{V}(e_x) = 2e_a^T \overset{\bullet}{e}_a = -2\left[\sigma_r - m_a\right]^T \overset{\bullet}{m}_a \tag{24}$$

Then, choosing I_c such that:

$$I_{c_i} = \begin{cases} 1 \text{ if } m_a[i] < \sigma_r[i] \\ 0 \quad otherwhise \end{cases} \tag{25}$$

we obtain:

$$\left[\sigma_r - m_a\right]^T I_c > \mathbf{0} \tag{26}$$

and

$$\left[\sigma_r - m_a\right]^T I_c = 0 \text{ iff } \left[\sigma_r - m_a\right]^T = \mathbf{0}$$

thus $\overset{\bullet}{V}(e_x) < 0$ and $\overset{\bullet}{V}(0) = 0$.

Since $m_a(0) = 0$ and it only increase its value, then I_{c_i} is feasible leading from $m_a(0) = 0$ to $m_a(\tau_f) = \sigma_r$, i.e. from m_0 to m_r. Moreover, assuming $m_r \in int\left(Class\left(m_0\right)\right)$ it is reached in finite time because $\overset{\bullet}{m}_a[i] = m\left(\min\left({}^\bullet t_i\right)\right) e^{\lambda\tau}$ and $m\left(\min\left({}^\bullet t\right)\right) \neq 0 \ \forall\tau$. At τ_f the control law must be switched from $I_c(\tau_f)$ to $I_c(\tau_{ss}) = I_{c_r}$ and the regulation control problem is solved.

■

The solution to the $RCP\ (m_r, I_{c_r})$ include both, the transitory and steady state control of metabolic systems. It is an improvement to current control solutions, where the biologist and metabolic engineers use stoichiometric non-dynamical approaches such as FBA (Flux Balance Analysis) [22], [23], [24] for the control of metabolic systems. Those are based on a pseudo-stationary state model, represented by the equation:

$$Sv = 0 \tag{27}$$

where S is the matrix of stoichiometry coefficients and the solution v gives the balance of mass for a single equilibrium point at that state (v is the reaction rates vector in a steady state).

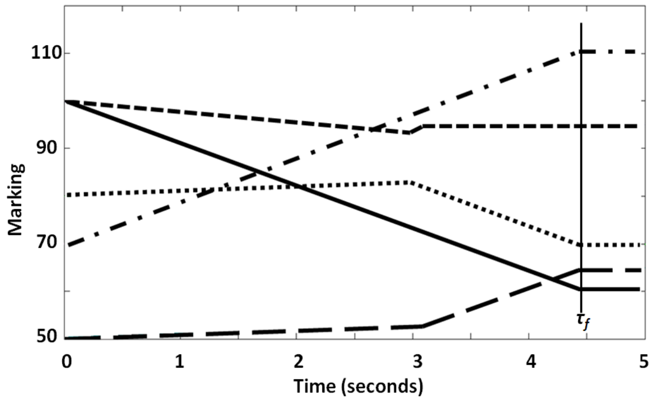


Fig. 6. Marking evolution of the net of Figure 3 applying $RCP\ (m_r, I_{c_r})$.

5 Illustrative Controlling Metabolic System Example

In order to illustrate the $RCP\ (m_r, I_{c_r})$ applied to a metabolic system, suppose the pathway module of Figure 2 together with modules c and d of Figure 1 comprise a cell metabolome. The initial marking used for this example is an arbitrary but physiologically possible initial state for the alleged metabolic model.

Example 5. Let the metabolome model of the Figure 3 be the system $TCPN = (N, \lambda, m_0)$ with $\Lambda = diag\,(2, 3, 4, 1)$ and $m_0 = \begin{bmatrix}100\ 80\ 100\ 50\ 70\ 5\ 3\ 2\ 4\end{bmatrix}^T$. Let $m_r = \begin{bmatrix}95\ 70\ 60\ 65\ 110\ 5\ 3\ 2\ 4\end{bmatrix}^T$ be a required marking. We make the extended system like the procedure showed in the Figure 5. We need the solution of σ_r from $m_r = m_0 + C\sigma_r$. Notice that there are a lot of solutions for σ_r but we only focus on the smallest solution of σ_r. For this example the solution is:

$$\sigma_r = \begin{bmatrix}30\ 40\ 25\ 0\end{bmatrix}^T$$

Solving the $RCP\ (m_r, I_{c_r})$ and applying the control (25) to the $TCPN = (N, \lambda, m_0)$, the metabolite concentrations are depicted in Figure 6. The reaction velocities (transition flux) is depicted in Figure 7. Notice that from $\tau = 0$ to $\tau = \tau_f \approx 4.5$ occurs the transitory dynamics, and for $\tau > \tau_f$ the steady state is reached.

Example 6. In Figure 8 the evolution of marking m_a is depicted. When occurs $m_a[i] = \sigma_r[i]$ the control $I_{c_i} = 0$ makes $f_i = 0$ and $m_a[i]$ is maintained until $\tau = \tau_f$ $(m_a = \sigma_r)$. Then I_c switches to I_{c_r} for the steady state control.

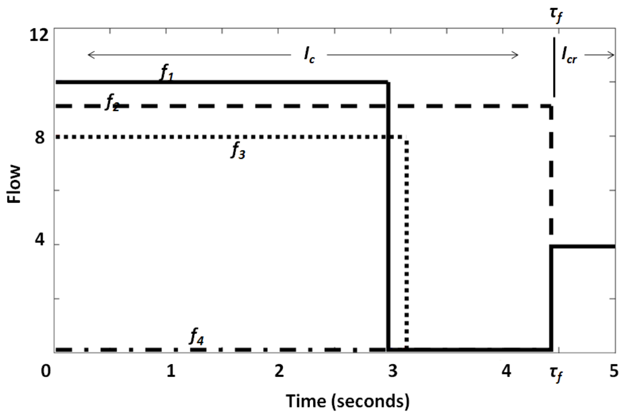


Fig. 7. Reaction velocities (transition flows) of the controlled metabolic model of the Example 5. Notice that $I_c(\tau)$ is applied for $0 \leq \tau < \tau_f$ and $I_{cr}(\tau)$ for $\tau > \tau_f$.

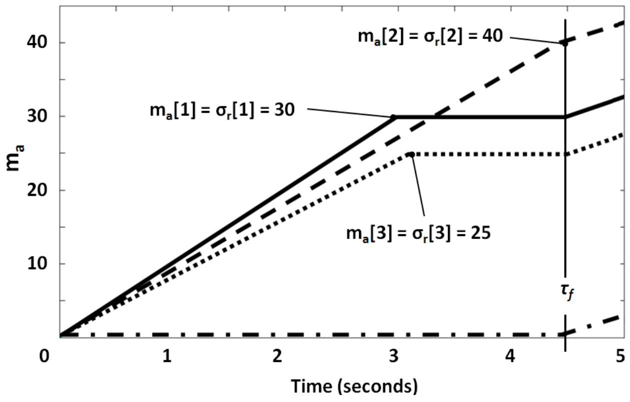


Fig. 8. Marking m_a of the Example 5.

6 Conclusions

This work presented a model methodology to capture the metabolome behavior. It uses a bottom-up approach where each individual biochemical reaction is modeled by elementary $TCPN$ modules and, afterwards, all the modules are merged into a single one to capture the whole metabolome behavior. Such characteristic of the methodology makes it simple and easy to use while the complex cell metabolic behavior is captured. This work also presented the problem of reaching a required metabolome state. The solution to this problem are the instantaneous reaction velocities that are realizable in biological system.

Present results are being applied to optimize metabolome fermentation in the production of tequila and to biofuels generation.

Future perspective involves introduction of stochastic modelling and merging the metabolome with the signaling and genetic networks.

References

1. Murata, T.: Petri nets: Properties, analysis and applications. In: Proceedings of IEEE. Volume 77. (1989) 541–580
2. Javier, Esparza; Manuel, S.: Compositional synthesis of live and bounded free choice petri nets. LNCS **571** (1991) 172–187
3. Jorg Dessel, J.E.: Free Choice Petri Nets. Cambridge University Press (1995)
4. Manuel Silva, L.R.: Continuization of timed petri nets: From performance evaluation to observation and control. 26th International Conference On Application and Theory of Petri Nets and Other Models of Concurrency (2006)
5. R. David, H.A.: Continuous petri nets. In: In Proc. Of the 8th European Workshop on Application and Theory of Petri Nets. (1987) 275–294
6. Peter J. E. Goss, J.P.: Quantitative modeling of stochastic systems in molecular biology by using stochastic petr nets. In: Natl. Acad. Sci. USA. Volume 95. (1998) 6750–6755
7. Masao Nagasaki, Atsushi Doi, H.M.S.M.: Petri net based description and modeling of biological pathways. Algebraic Biology (2005) 19–31
8. Monika Heiner, David Gilbert, R.D.: Petri nets for systems and syntethic biology. (2008) 215–264
9. David Angeli, Patrick De Leenheer, E.S.: A petri net approach to persistance analysis in chemical reaction networks. Biology and Control Theory: Current Challenges (2007) 181–216
10. Riel, N.A.W.V.: Dynamic modelling and analysis of biochemical networks: Mechanism-based models and model-based experiments. Briefings In Bioinformatics **7** (2006) 364–374
11. Nevoigt, E.: Progress in metabolic engineering of saccharomyces cerevisiae. Microbiology And Molecular Biology Reviews **72** (2008) 379–412
12. Mor Peleg, Daniel Rubin, R.B.A.: Using petri net tools to study properties and dynamics of biological systems. Journal of the American Medical Informatics Association **12** (2005) 181–199
13. Hiroshi Matsuno, Atsushi Doi, M.N.S.M.: Hybrid petri net representation of gene regulatory network. Pacific Symposium on Biocomputing **5** (2000) 338–349

14. Cassandras, C.G.: Discrete Event Systems. Modelling and Performance Analysis. Asken Associates (1993)
15. René David, H.A.: Continuous and hybrid petri nets. Journal of Circuites, Systems and Computers **8** (1998) 159–188
16. R. David, H.A.: Continuous petri nets. Proceedings of the 8th European Workshop on Application and Theory of Petri Nets (1987) 275–294
17. Manuel Silva, L.R.: On fluidification of petri net models: From discrete to hybrid and continuous model. Annual Reviews in Control 28 (2004) 253–266
18. C. R. Vázquez, A. Ramírez-Treviño, L.R.M.S.: On controllability of timed continuous petri nets. 11th Int. Workshop Hybrid Systems: Computational and Control **4981** (2008) 528–541
19. Cristian Mahulea, Antonio Ramirez, L.R.M.S.: Steady state control, zero valued poles and token conservation law in continuous net systems. Proceedings of the International Workshop on Control of Hybrid and Discrete Event Systems (2005)
20. Laszlo Kurti, B.C.: Strategic Applications of Named Reactions in Organic Synthesis. Elsevier Academic Press. USA (2005)
21. Segel, I.H.: Enzyme Kinetics. New York: Wiley-Interscience (1975)
22. J. M. Lee, E. P. Gianchandani, J.A.P.: Flux balance analysis in the era of metabolomics. Briefs in Bioinformatics **7** (2006) 140–150
23. J. S. Edwards, M. Covert, B.P.: Metabolic modelling of microbes: The flux-balance approach. Environmental Microbiology **4** (2002) 133–140
24. Francisco Llaneras, J.P.: Stoichiometric modelling of cell metabolism. Journal of Bioscience and Bioengineering **105** (2008) 1–11

Model transformation of metabolic networks using a Petri net based framework

Daniel Machado[1], Rafael S. Costa[1], Miguel Rocha[2], Isabel Rocha[1], Bruce Tidor[3], and Eugénio C. Ferreira[1]

[1] IBB-Institute for Biotechnology and Bioengineering/Centre of Biological Engineering, University of Minho, Campus de Gualtar, 4710-057 Braga, Portugal
{dmachado,rafacosta,irocha,ecferreira}@deb.uminho.pt
[2] Department of Informatics/CCTC, University of Minho, Campus de Gualtar, 4710-057 Braga, Portugal
mrocha@di.uminho.pt
[3] Department of Biological Engineering/Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
tidor@mit.edu

Abstract. The different modeling approaches in Systems Biology create models with different levels of detail. The transformation techniques in Petri net theory can provide a solid framework for zooming between these different levels of abstraction and refinement. This work presents a Petri net based approach to Metabolic Engineering that implements model reduction methods to reduce the complexity of large-scale metabolic networks. These methods can be complemented with kinetics inference to build dynamic models with a smaller number of parameters. The central carbon metabolism model of *E. coli* is used as a test-case to illustrate the application of these concepts. Model transformation is a promising mechanism to facilitate pathway analysis and dynamic modeling at the genome-scale level.

1 Introduction

Systems Biology provides a new perspective in the study of living systems and embraces the complexity emerging of interactions among all biological components. Combining theory and experiments, scientists build models to explain and predict the behavior of the systems under study. Metabolic Engineering is one of the fields where this perspective has proven useful through the optimization of metabolic processes for industrial applications [28, 2].

Modeling in Systems Biology is an iterative process as the life-cycle of a model is comprised of successive refinements using experimental data. Different approaches, such as top-down, bottom-up or middle-out [18] are used depending on the purpose of the model and the type of data available for its construction. In Metabolic Engineering there are macroscopic kinetic models that consider the cell as a black-box converting substrates into biomass and products, which are typically used for bioprocess control. On the other hand, there are reaction-network-level models, either medium-scale dynamic models with detailed kinetic information derived from literature and experimental data [3], or genome-scale stoichiometric reconstructions derived from genome annotation complemented with literature review [5].

Although the ultimate goal of Systems Biology is a complete understanding of the cell as a whole, not only it is extremely difficult to collect all the kinetic information necessary to build a fully detailed whole-cell model due to the lack of experimental data and model identifiability concerns, but also the computational cost of simulating the dynamics of a system with such detail would be tremendous. Therefore, there is a need to fit the level of detail of a model to the specific problem at hand. For instance, Metabolic Pathway Analysis (MPA) has been useful in the analysis of metabolism as a way to determine, classify and optimize the possible pathways throughout a metabolic network. However, due to the combinatorial explosion of pathways with increasing number of reactions, it is still infeasible to apply these methods in genome-scale metabolic reconstructions without decomposing the network into connected modules [23, 24]. This zooming in and out between different levels of abstraction and connecting parts with different levels of detail is a feature where formal methods and particularly Petri nets may play an important role. Concepts such as subnetwork abstraction, transition refinement or node fusion, among others, have been explored in Petri net theory [8] and may provide the theoretical background for method development.

In previous work, we reviewed different modeling formalisms used in Systems Biology from a Metabolic Engineering perspective and concluded that Petri nets are a promising formalism for the creation of a common framework of methods for modeling, analysis and simulation of biological networks [15]. They are a mathematical and graphical formalism, therefore intuitive and amenable to analysis. The different extensions available (*e.g.:* stochastic, continuous, hybrid) provide the flexibility required to model and integrate the diversity of phenomena occurring in the main types of biological networks (metabolic, regulatory and signaling). Moreover, one may find biological meaning in several concepts in Petri net theory; for instance, the incidence matrix of a Petri net is the equivalent of the stoichiometric matrix, and the minimal *t-invariants* correspond to the elementary flux modes (EFMs).

In this work, we explore strategies of model reduction for Petri net representations of metabolic networks, and the integration of this methodology with recent approaches such as genome-scale dynamic modeling. This paper is organized as follows. Section 2 explores the motivation for the work. Section 3 presents the model reduction and kinetics inference methods, Section 4 discusses their application to *E. coli* and Section 5 elaborates on conclusions and future work.

2 Background

There are different examples of model reduction in the literature. One such method was developed in [17], based on timescale analysis for classification of metabolite turnover time using experimental data. The fast metabolites are removed from the differential equations and their surrounding reactions are lumped. In [20] the EFMs of a reaction network are calculated in order to create a macroscopic pathway network, where each EFM is a macro-reaction connecting extracellular substrates and products. A simple Michaelis–Menten rate law is assumed for each macro-reaction and the parameters are inferred from experimental data. The method is applied in a network with 18 reactions and a total of 7 EFMs. However it does not scale well to larger networks because, in the worst case, the number of EFMs grows exponentially with the network size.

The combinatorial pathway explosion problem is well known; there are methods for network decomposition in the literature that address this issue. In [23] the authors perform a genome-scale pathway analysis on a network with 461 reactions. After estimating the number of extreme pathways (EPs) to be over a million, the network is decomposed into 6 subsystems according to biological criteria and the set of EPs is computed separately for each subsystem. A similar idea in [24] consists on automatic decomposition based on topological analysis. The metabolites with higher connectivity are considered as external and connect the formed subnetworks. An automatic decomposition approach based on Petri nets is the so-called *maximal common transition sets* (MCT-sets) [22], and consists on decomposing a network into modules by grouping reactions by participation in the minimal *t-invariants* (equivalent to EFMs). A related approach relies on clustering of *t-invariants* for network modularization [9]. A very recent network coarsening method based on so-called *abstract dependent transition sets* (ADT-sets) is formulated without the requirement of pre-computation of the *t-invariants* and thus may be a promising tool for larger networks [12].

Another problem in genome-scale metabolic modeling is the study of dynamic behavior. Genome-scale metabolic reconstructions are stoichiometric and usually analyzed under steady-state assumption using constraint-based methods [1]. Dynamic flux balance analysis (dFBA) allows variation of external metabolite concentrations, and simulates the network dynamics assuming an internal pseudo steady-state at each time step [16]. It is used in [19] to build a genome-scale dynamic model of *L. lactis* that simulates fermentation profiles. However, this approach gives no insight into intracellular dynamics, neither it integrates reaction kinetics. In [26] the authors build a kinetic genome-scale model of *S. cerevisiae* using linlog kinetics, where the reference steady-state is calculated using FBA. Some of the elasticity parameters and metabolite concentrations are derived from available kinetic models, while the majority use default values. Using the stoichiometric coefficients as elasticity values is a rough estimation of the influence of the metabolites on the reaction rates. Moreover, no time-course simulation is performed. Mass action stoichiometric simulation (MASS) models are introduced in [14] as a way to incorporate kinetics into stoichiometric reconstructions. Parameters are estimated from metabolomic data. Regulation can be included by incorporating the mechanistic metabolite/enzyme interactions. A limitation of these models is that mass-action kinetics do not reflect the usual non-linearity of enzymatic reactions and the incorporation of regulation leads to a significant increase in network size.

3 Methods

The idea of this work is closer to the reduction concepts of [17, 20] than the modularization concepts in [23, 24]. In the latter cases a large model is decomposed into subunits to ease its processing by analyzing the parts individually. Instead, our objective is to facilitate the visualization, analysis and simulation of a large-scale model as a whole by abstracting its components. This reduction is to be attained by reaction lumping in a way that maintains biological meaning and valid application of current analysis and simulation tools. The Michaelis–Menten kinetics is a typical example of abstraction, where the small network of mass-action reactions are lumped into one single reaction.

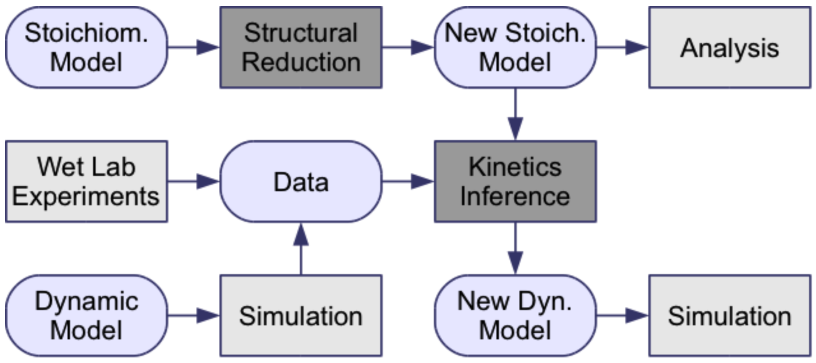


Fig. 1. Overall concept of model reduction and kinetics inference.

The overall idea of the model reduction method is depicted in Fig. 1. A large-scale stoichiometric model can be structurally reduced into a simplified version that can be more easily analyzed by methods such as MPA. Also, one may infer a kinetic structure to build a dynamic version of the reduced model. Due to the smaller size, a lower number of parameters has to be estimated. The data used for estimation may be experimental data found in the literature, or pseudo-experimental data from dynamic simulations if part of the system has been kinetically characterized.

When abstracting a reaction subnetwork into one or more macro-reactions, it is important to consider the assumptions created by such abstraction. As in Michaelis–Menten kinetics, these simplifications result in a pseudo-steady-state assumption for the intermediate species that disappear. While this may not be a problem for flux balance models, it changes the transient behavior of dynamic models because the buffering effect of intermediates in a pathway is neglected. The selection of metabolites to be removed depends on the purpose of the reduction. The network may have different levels of granularity based on the availability of experimental data, topological properties, or simply in order to aggregate pathways according to biological function.

3.1 Basic definitions

The proposed method for model reduction uses several Petri net concepts from the literature. We will use the following definition of an unmarked continuous Petri net (adapted from [4]) for modeling a stoichiometric metabolic network:

$$\begin{aligned} Pn &= < P, T, Pre, Post > \\ Pre &: P \times T \to \mathbb{R}^+ \\ Post &: P \times T \to \mathbb{R}^+ \end{aligned}$$

where the set of places P represents the metabolites, the set of transitions T represents the reactions and $Pre, Post$ are, respectively, the substrate and product stoichiometries of the reactions. Note that for the representation of a stoichiometric network, a discrete Petri net usually suffices; however, because some models may contain non-integer stoichiometric coefficients, the continuous version was adopted. Moreover, we will assume that reversible reactions are split into irreversible reaction pairs. We will also use the following definitions:

$$\begin{aligned} loc(x) &= \{x\} \cup {}^\bullet x \cup x^\bullet \\ In(p) &= \sum_{t \in {}^\bullet p} Post[p,t] \cdot v(t) \\ Out(p) &= \sum_{t \in p^\bullet} Pre[p,t] \cdot v(t) \end{aligned}$$

where ${}^\bullet x, x^\bullet$ are sets representing the input and output nodes of a node x, the set $loc(x) \subseteq P \cup T$ is called the locality of x, function $v : T \to \mathbb{R}_0^+$ is a given flux distribution (or the so-called instantaneous firing rate), and $In, Out : P \to \mathbb{R}_0^+$ are, respectively, the feeding and draining rates of the metabolites.

The method for network reduction consists of eliminating a set of selected metabolites from the network. For each removed metabolite its surrounding reactions are lumped in order to maintain the fluxes through the pathways. This reduction assumes a steady-state condition for the metabolite, *i.e.* $In(p) = Out(p)$.

3.2 Model reduction: Conjunctive fusion

There are two options for lumping the reactions depending on the transformation method applied. The first approach is based on a transformation called *conjunctive transition fusion* [8] and it results in an abstraction that replaces the transition-bordered subnet $loc(p)$ by a single macro-reaction. The drawback of this method is that the flux ratios between the internal reactions are lost. If a known steady-state flux distribution (v) is given, then the stoichiometric coefficients can be adjusted to preserve the ratios for that distribution; however, the space of solutions of the flux balance formulation becomes restricted to a particular solution. In the limiting case, if all the internal metabolites are removed, the cell is represented by one single macro-reaction connecting extracellular substrates and products with the stoichiometric yields inferred from the

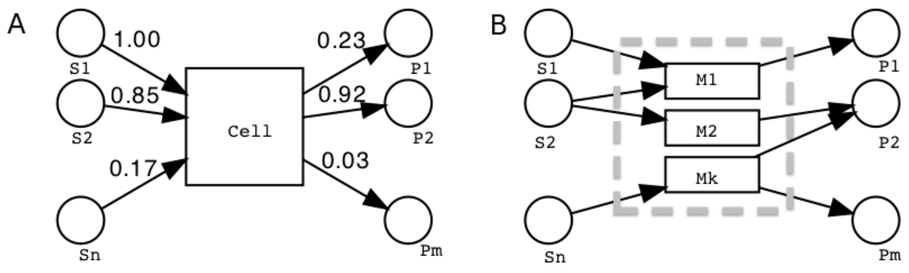


Fig. 2. Exemplification of limit scenarios where all the internal metabolites are removed. (A) In the conjunctive reduction case the result is one single macro-reaction converting substrates into products with the respective yields specified in the stoichiometry. (B) In the disjunctive reduction method, all possible pathways connecting substrates and products are enumerated.

network topology for one particular steady-state (Fig 2A). The transformation method for removing metabolite p in Pn given a flux distribution v is described as follows:

$$\begin{aligned}
Pn' &= < P', T', Pre', Post' > \\
P' &= P \setminus \{p\} \\
T' &= T \setminus ({}^{\bullet}p \cup p^{\bullet}) \cup \{t_p\} \\
Pre' &= \{(p_i, t_j) \mapsto Pre(p_i, t_j) \mid (p_i, t_j) \in dom(Pre) \setminus (P \times ({}^{\bullet}p \cup p^{\bullet}))\} \\
&\cup \{(p_i, t_p) \mapsto f_{in}(p_i) \mid p_i \in {}^{\bullet}({}^{\bullet}p \cup p^{\bullet}), p_i \neq p, v'(t_p) \neq 0, f_{in}(p_i) \neq 0\} \\
Post' &= \{(p_i, t_j) \mapsto Post(p_i, t_j) \mid (p_i, t_j) \in dom(Post) \setminus (P \times ({}^{\bullet}p \cup p^{\bullet}))\} \\
&\cup \{(p_i, t_p) \mapsto f_{out}(p_i) \mid p_i \in ({}^{\bullet}p \cup p^{\bullet})^{\bullet}, p_i \neq p, v'(t_p) \neq 0, f_{out}(p_i) \neq 0\} \\
v' &= \{t \mapsto v(t) \mid t \in T \setminus ({}^{\bullet}p \cup p^{\bullet})\} \cup \{t_p \mapsto In(p)\}.
\end{aligned}$$

where

$$\begin{aligned}
f_{in}(p_i) &= \frac{\sum_{t \in p_i^{\bullet} \cap ({}^{\bullet}p \cup p^{\bullet})} Pre(p_i, t) \cdot v(t)}{v'(t_p)} \\
f_{out}(p_i) &= \frac{\sum_{t \in {}^{\bullet}p_i \cap ({}^{\bullet}p \cup p^{\bullet})} Post(p_i, t) \cdot v(t)}{v'(t_p)}
\end{aligned}$$

The stoichiometric coefficients of the new reaction may be very high or low, depending on $v'(t_p)$ and so, optionally, one may also normalize them with some scalar λ, such that $Pre''(p_i, t_p) = \frac{1}{\lambda} \cdot Pre'(p_i, t_p)$, $Post''(p_i, t_p) = \frac{1}{\lambda} \cdot Post'(p_i, t_p)$ and $v''(t_p) = \lambda \cdot v'(t_p)$. This will also make the final result independent of the order of the metabolites removed. A good choice for λ is:

$$\lambda = \max\left(\{Pre(p_i, t_p) \mid p_i \in {}^{\bullet}t_p\} \cup \{Post(p_i, t_p) \mid p_i \in t_p{}^{\bullet}\}\right)$$

3.3 Model reduction: Disjunctive fusion

The second approach is based on a transformation called *disjunctive transition fusion* [8], where every combination of input and output reaction pairs connected by the removed metabolite is replaced by one macro-reaction. Although this approach does not constrain the steady-state solution space of the flux distribution, it has the drawback of increasing the number of transitions, if the metabolite is highly connected, due to the combinatorial procedure. Note that applying this reduction step to metabolite p_i is equivalent to performing one iteration of the *t-invariant* calculation algorithm to remove column i of the transposed incidence matrix. Therefore, in the limiting case where all internal metabolites are removed, the cell is represented by the set of all possible pathways connecting extracellular substrates and products (Fig. 2B), as was done in [20]. The definition, similar to the previous one, is as follows:

$$
\begin{aligned}
Pn' =& < P', T', Pre', Post' > \\
P' =& P \setminus \{p\} \\
T' =& T \setminus ({}^\bullet p \cup p^\bullet) \cup \{t_{xy} \mid (x, y) \in ({}^\bullet p \times p^\bullet)\} \\
Pre' =& \{(p_i, t) \mapsto Pre(p_i, t) \mid (p_i, t) \in dom(Pre) \setminus (P \times ({}^\bullet p \cup p^\bullet)\} \\
& \cup\{(p_i, t_{xy}) \mapsto Pre_0(p_i, x) \cdot Pre(p, y) + Pre_0(p_i, y) \cdot Post(p, x) \\
& \quad \mid (x, y) \in ({}^\bullet p \times p^\bullet), p_i \in {}^\bullet\{x, y\}\} \\
Post' =& \{(p_i, t) \mapsto Post(p_i, t) \mid (p_i, t) \in dom(Post) \setminus (P \times ({}^\bullet p \cup p^\bullet)\} \\
& \cup\{(p_i, t_{xy}) \mapsto Post_0(p_i, x) \cdot Pre(p, y) + Post_0(p_i, y) \cdot Post(p, x) \\
& \quad \mid (x, y) \in ({}^\bullet p \times p^\bullet), p_i \in \{x, y\}^\bullet\}
\end{aligned}
$$

where

$$
Pre_0(p, t) = \begin{cases} Pre(p, t) & \text{if } (p, t) \in dom(Pre) \\ 0 & \text{if } (p, t) \notin dom(Pre) \end{cases}
$$

$$
Post_0(p, t) = \begin{cases} Post(p, t) & \text{if } (p, t) \in dom(Post) \\ 0 & \text{if } (p, t) \notin dom(Post) \end{cases}
$$

Whenever there are pathways with the same net stoichiometry, these can be removed by checking the columns of the incidence (stoichiometric) matrix and eliminating repeats. It should also be noted that in both methods, if a metabolite acts both as substrate and product in a lumped reaction, it will create a redundant cycle that is not reflected in the incidence matrix. If these cycles are not removed, they propagate through the reduction steps; therefore, they should be replaced by a single arc containing the overall stoichiometry. The procedure works as follows:

$$\begin{aligned} Pre' =& \{(p,t) \mapsto Pre(p,t) \mid (p,t) \in dom(Pre) \setminus dom(Post)\} \\ & \cup \{(p,t) \mapsto Pre(p,t) - Post(p,t) \\ & \quad \mid (p,t) \in dom(Pre) \cap dom(Post), Pre(p,t) > Post(p,t)\} \\ Post' =& \{(p,t) \mapsto Post(p,t) \mid (p,t) \in dom(Post) \setminus dom(Pre)\} \\ & \cup \{(p,t) \mapsto Post(p,t) - Pre(p,t) \\ & \quad \mid (p,t) \in dom(Pre) \cap dom(Post), Post(p,t) > Pre(p,t)\} \end{aligned}$$

The previous arc removing procedure may cause isolation of some nodes when $Pre(p,t) = Post(p,t)$; therefore, the isolated nodes should be removed:

$$\begin{aligned} P' &= \{p \mid p \in P, loc(p) \neq \{p\}\} \\ T' &= \{t \mid t \in T, loc(t) \neq \{t\}\} \end{aligned}$$

3.4 Kinetics inference

Given a stoichiometric model, if metabolomic or fluxomic data are available for parameter estimation, one may try to build a dynamic model by inferring appropriate kinetics for the reactions. In [25] the authors propose that this is performed by assuming linlog kinetics for all reactions using an FBA solution as the reference state and the stoichiometries as elasticity parameters. An integration of Biochemical Systems Theory (BST) with Hybrid Functional Petri Nets (HFPN) is presented in [29], where general mass action (GMA) kinetics is assumed for each transition. The review of kinetic rate formulations is out of the scope of this work and may be found elsewhere [10].

Assuming that all metabolites with unknown concentration were removed, we will extend our definition to a marked continuous Petri net:

$$Pn = < P, T, Pre, Post, m_0 >$$

where $m_0 : P \to \mathbb{R}_0^+$ denotes the initial marking (concentration) of the metabolites. The kinetics inference process consists on defining a firing rate $v : T \to \mathbb{R}_0^+$, which will be dependent on the current marking (m) and the specific kinetic parameters (see [7] for an introduction on marking-dependent firing rates). As we assumed irreversible reactions, each rate will only vary with substrate concentration. The rates can be easily derived from the net topology. In case of GMA kinetics v is given by:

$$v(t) = k_t \prod_{p \in {}^\bullet t} m(p)^{a_{p,t}}$$

where k_t is the kinetic rate of t and $a_{p,t}$ is the kinetic order of metabolite p in reaction t. A usual first approximation for $a_{p,t}$ is $Pre(p,t)$.

Linlog kinetics are formulated based on a reference rate v_0, and defined by:

$$v(t) = v_0(t) \left(1 + \sum_{p \in {}^\bullet t} \varepsilon_{p,t}^0 \ln \left(\frac{m(p)}{m_0(p)} \right) \right)$$

where $\varepsilon^0_{p,t}$ is called the elasticity of metabolite p in reaction t, reflecting the influence of the concentration change of the metabolite in the reference reaction rate. As in the previous case, $Pre(p,t)$ can be chosen as an initial approximation for $\varepsilon^0_{p,t}$. The relative enzyme activity term (e/e_0) commonly present in linlog rate laws to account for regulatory effects at larger time scales will not be considered.

4 Results and Discussion

The proposed methods were tested using the dynamic central carbon metabolism model of *E. coli* [3], where the stoichiometric part was used for the application of the reduction methods, and the dynamic profile was used to generate pseudo-experimental data sets for parameter estimation and validation of the kinetics inference method. A Petri net representation of this model (Fig. 3) was built using the Snoopy tool [21]. All reversible reactions were split into irreversible pairs. The net contains a total of 18 places, 44 transitions and is covered by 95 *semipositive t-invariants*, computed with the Integrated Net Analyzer [27].

In the application of the conjunctive method (Fig 4A), the metabolites were classified as in [17] based on their timescale (Table 1), by calculating their turnover time ($\tau : P \to \mathbb{R}^+_0$) using the reference steady-state of the dynamic model, where:

$$\tau(p) = \frac{m_0(p)}{In(p)}$$

Metabolites with small turnover time are considered fast. In this case, all metabolites except the slowest 5 (*glcex, pep, g6p, pyr, g1p*) were removed.

For the application of the disjunctive method (Fig 4B), the metabolites were classified based on their topology (Table 1). We conveniently opted to remove the metabolites with lower connectivity to avoid the combinatorial explosion problem. All metabolites except 5 (*g6p, pyr, f6p, gap, xyl5p*) were removed. This reduction assumes steady-state for the removed metabolites. However, it makes no assumptions on the ratios between the fluxes, therefore preserving the flux-balance solution space.

Because we are assuming that the reference steady-state is known, the conjunctive reduced model was chosen for the application of the kinetics inference method assuming linlog kinetics at the reference state. The elasticity parameters were estimated using COPASI [13]. The pseudo-experimental data was generated from simulation with the original model after a 1 mM extracellular glucose pulse with the addition of Gaussian noise ($std = 0.05$ mM) (Fig. 5A). The fitted model was then validated using pseudo-experimental data from a 2 mM pulse (Fig. 5B). It is possible to observe an instantaneous increase in *pyr* (from 2.67 to 3.93) and an instantaneous decrease *pep* (from 2.69 to 1.26) which the model is unable to reproduce. The poor fitting in some of the intracellular metabolites is expected given the significant reduction to the model. However, the extracellular glucose consumption profile is remarkably good, both in the fitting and validation cases.

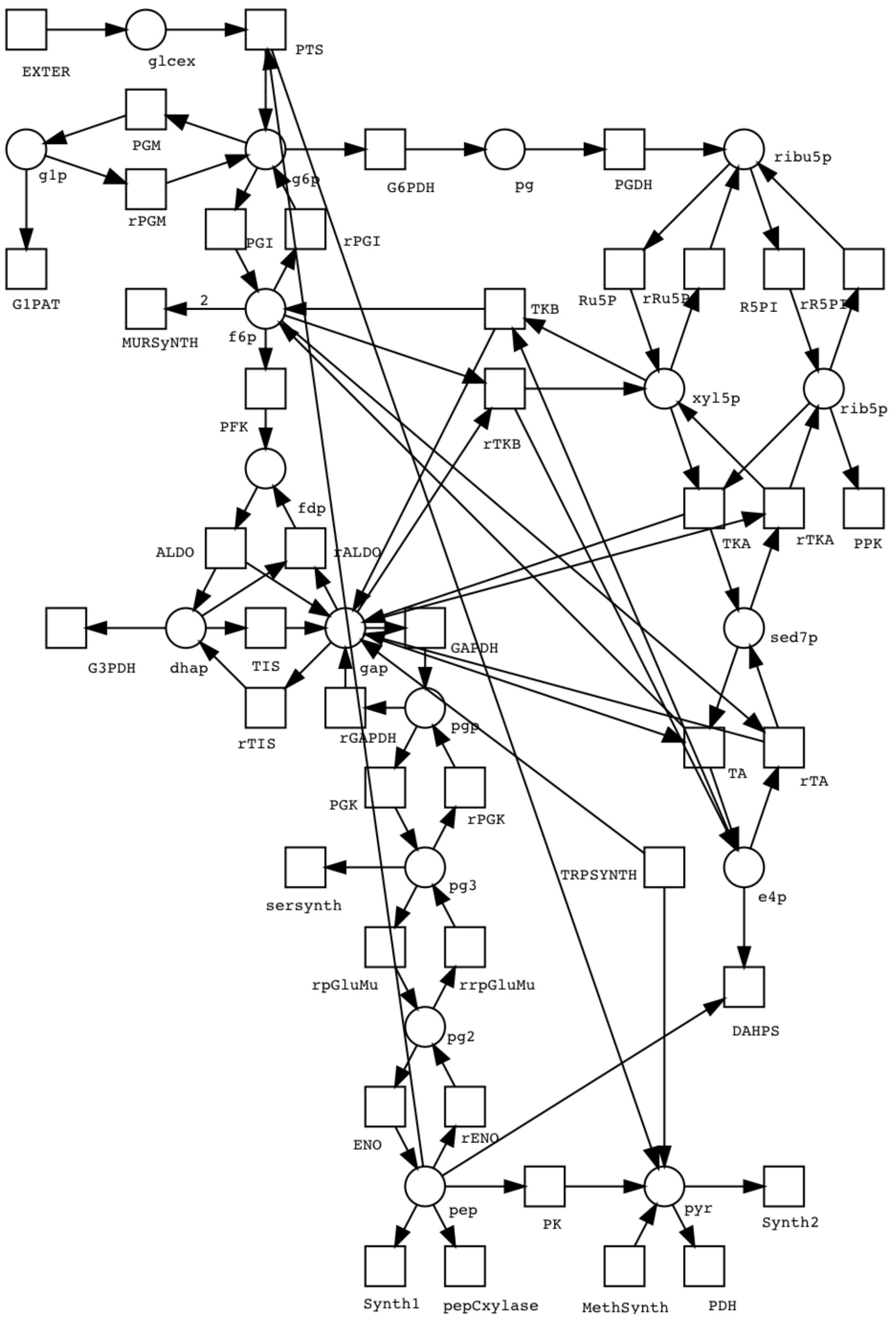


Fig. 3. Petri net model of the dynamic central carbon metabolism model of *E. coli* with reversible reactions split into irreversible pairs.

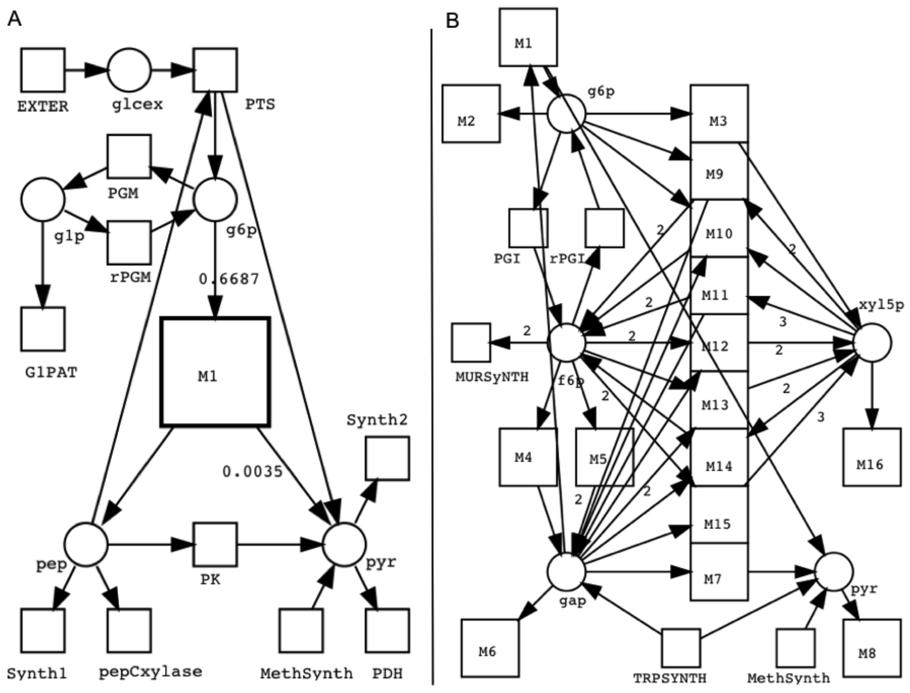


Fig. 4. Reduced versions of the original network. (A) Conjunctive reduction method. (B) Disjunctive reduction method.

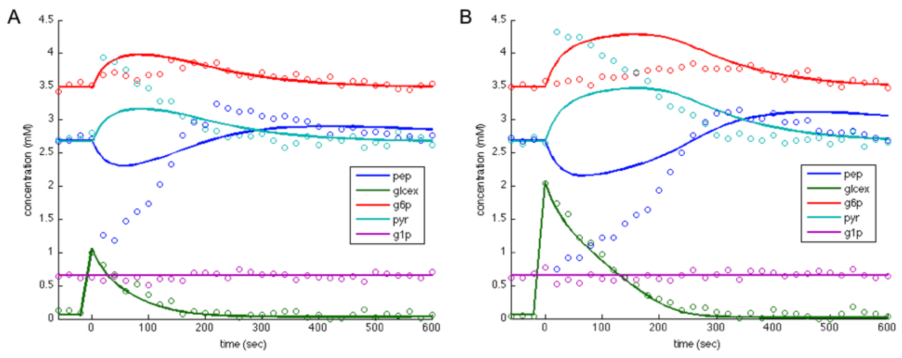


Fig. 5. (A) Results of parameter estimation with pseudo-experimental data with 1 mM extracellular glucose pulse. (B) Validation of the model with a 2 mM extracellular glucose pulse. In both cases, the circles represent the experimental data and the lines represent time-course simulations generated by the reduced model.

Table 1. Metabolite topological properties (input reactions, output reactions, connectivity) and dynamic properties (concentration, flux, turnover time) at the reference steady-state.

Metabolite	$\#({}^\bullet p)$	$\#(p^\bullet)$	$\#({}^\bullet p \times p^\bullet)$	m_0 (mM)	In (mM/s)	τ (s)
glcex	1	1	1	0.0558	0.0031	**18.099**
pep	1	6	6	2.6859	0.3031	**8.8603**
g6p	3	3	**9**	3.4882	0.2004	**17.406**
pyr	4	2	**8**	2.6710	0.2418	**11.044**
f6p	3	5	**15**	0.6014	0.1423	4.2266
g1p	1	2	2	0.6539	0.0023	**278.62**
pg	1	1	1	0.8092	0.1397	5.7929
fdp	2	1	2	0.2757	0.1414	1.9495
sed7p	2	2	4	0.2761	0.0454	6.0757
gap	7	6	**42**	0.2196	0.3661	0.5997
e4p	2	3	6	0.0986	0.0454	2.1684
xyl5p	3	3	**9**	0.1385	0.0839	1.6503
rib5p	2	3	6	0.3994	0.0558	7.1626
dhap	2	3	6	0.1682	0.1414	1.1892
pgp	2	2	4	0.0080	0.3207	0.0251
pg3	2	3	6	2.1437	0.3207	6.6851
pg2	2	2	4	0.4014	0.3031	1.3241
ribu5p	3	2	6	0.1114	0.1397	0.7974

Although both reducing methods can be combined with kinetics inference, the conjunctive version seems more suitable if a steady-state distribution is known, because it generates smaller models, hence less parameters. The disjunctive version is appropriate for analyzing all elementary pathways between a set of metabolites without the burden of calculating the set of EFMs of the whole model. For instance, the macro-reactions *M4* (*ALDO* + *G3PDH*) and *M5* (*ALDO* + *TIS*), with net stoichiometries of, respectively, [*fdp* $\rightarrow$ *gap*] and [*fdp* $\rightarrow$ 2 *gap*], are two unique pathways between these two metabolites.

5 Conclusions

This work presents strategies for model reduction of metabolic networks based on a Petri net framework. Two approaches, conjunctive and disjunctive reduction are presented. The conjunctive approach allows the abstraction of a subnetwork into one lumped macro-reaction, however limited to one particular flux distribution of the subnetwork. The disjunctive approach on the other hand, makes no assumptions on the flux distribution by replacing the removed subnetwork with macro-reactions for all possible pathways through the subnetwork, therefore not constraining the steady-state solution space. In both cases, the reduced model may be transformed into a dynamic model using kinetics inference and parameter estimation if experimental data is available. Using the reduced model, instead of the original, facilitates this process because it significantly decreases the number of parameters to be estimated.

In future work, we intend to build a dynamic genome-scale model of *E. coli* by using the already available central carbon dynamic model [3], complemented with lumped versions of the surrounding pathways in the genome-scale network [5]. Note that some of the reactions on the central carbon model already represent lumped versions of some biosynthetic pathways (*e.g. mursynth, trpsynth, methsynth, sersynth*). However they were not deduced from the genome-scale network and may not be accurate abstractions of these pathways.

Among the extensions available to Petri nets are the addition of different types of arcs, such as read-arcs and inhibitor-arcs, which could be use to represent activation and inhibition in biochemical reactions. They could also be used to integrate metabolic and regulatory networks. Optimization in metabolic processes is usually based on knockout simulations in metabolic networks. However, these simulations do not take into consideration the possible regulatory effects caused by the knockouts. In our transformation methods we removed the arcs with the same stoichiometry in both directions, because these are not reflected in the stoichiometric matrix. In the Michaelis–Menten example this results in removing the enzyme from the network. The proposed methods can be extended to consider read-arcs for these situations, which should be preserved during the reduction steps, therefore establishing connection places to the integration of a regulatory network (Fig 6).

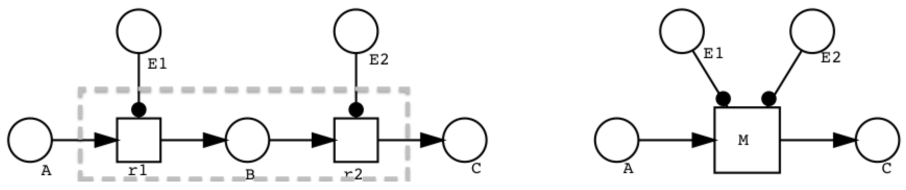


Fig. 6. Reduction step conserving the read-arcs associated with the enzymes of the original reactions.

An alternative to the reduction of the models would be to consider their representation using hierarchical Petri nets. In this case, each macro-reaction would be connected to its detailed subnetwork. Although this would not reduce the number of kinetic parameters in the case of kinetics inference, it would be extremely useful for facilitated modeling and visualization of large-scale networks without compromising detail. It could also be the solution for genome-scale pathway analysis, if it is performed independently at each hierarchical level. The hierarchical model composition proposed for SBML [6] may facilitate the implementation of this alternative. See [11] for an automatic network coarsening algorithm based on hierarchical petri nets applied to different kinds of biological networks.

Acknowledgments. Research supported by PhD grants SFRH/BD/35215/2007 and SFRH/BD/25506/2005 from the Fundação para a Ciência e a Tecnologia (FCT) and the MIT–Portugal Program through the project "Bridging Systems and Synthetic Biology for the development of improved microbial cell factories" (MIT-Pt/BS-BB/0082/2008).

References

1. S.A. Becker, A.M. Feist, M.L. Mo, G. Hannum, B.Ø. Palsson, and M.J. Herrgard. Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. *Nature Protocols*, 2(3):727–738, 2007.
2. A.P. Burgard, P. Pharkya, and C.D. Maranas. Optknock: A bilevel programming framework for identifying gene knockout strategies for microbial strain optimization. *Biotechnology and Bioengineering*, 84(6):647–657, 2003.
3. C. Chassagnole, N. Noisommit-Rizzi, J.W. Schmid, K. Mauch, and M. Reuss. Dynamic modeling of the central carbon metabolism of Escherichia coli. *Biotechnology and Bioengineering*, 79(1):53–73, 2002.
4. R. David and H. Alla. *Discrete, continuous, and hybrid Petri nets*. Springer Verlag, 2005.
5. A.M. Feist, C.S. Henry, J.L. Reed, M. Krummenacker, A.R. Joyce, P.D. Karp, L.J. Broadbelt, V. Hatzimanikatis, and B.Ø. Palsson. A genome-scale metabolic reconstruction for Escherichia coli K-12 MG1655 that accounts for 1260 ORFs and thermodynamic information. *Molecular systems biology*, 3(1), 2007.
6. A. Finney. Developing SBML beyond level 2: proposals for development. In *Computational Methods in Systems Biology*, pages 242–247. Springer, 2005.
7. D. Gilbert and M. Heiner. From Petri nets to differential equations-an integrative approach for biochemical network analysis. *Petri Nets and Other Models of Concurrency-ICATPN 2006*, pages 181–200, 2006.
8. C. Girault and R. Valk. *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.
9. E. Grafahrend-Belau, F. Schreiber, M. Heiner, A. Sackmann, B.H. Junker, S. Grunwald, A. Speer, K. Winder, and I. Koch. Modularization of biochemical networks based on classification of Petri net t-invariants. *BMC bioinformatics*, 9(1):90, 2008.
10. J.J. Heijnen. Approximative kinetic formats used in metabolic network modeling. *Biotechnology and bioengineering*, 91(5):534–545, 2005.
11. M. Heiner. Understanding Network Behavior by Structured Representations of Transition Invariants. *Algorithmic Bioprocesses*, page 367, 2009.
12. M. Heiner and K. Sriram. Structural analysis to determine the core of hypoxia response network. *PloS one*, 5(1):e8600, 2010.
13. S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI—a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074, 2006.
14. N. Jamshidi and B.Ø. Palsson. Mass action stoichiometric simulation models: Incorporating kinetics and regulation into stoichiometric models. *Biophysical Journal*, 98:175–185, 2010.
15. D. Machado, R.S. Costa, M. Rocha, I. Rocha, B. Tidor, and E.C. Ferreira. A critical review on modelling formalisms and simulation tools in computational biosystems. In *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, pages 1063–1070. Springer, 2009.

16. R. Mahadevan, J.S. Edwards, and F.J. Doyle. Dynamic flux balance analysis of diauxic growth in Escherichia coli. *Biophysical journal*, 83(3):1331–1340, 2002.
17. I.E. Nikerel, W.A. van Winden, P.J.T. Verheijen, and J.J. Heijnen. Model reduction and a priori kinetic parameter identifiability analysis using metabolome time series for metabolic reaction networks with linlog kinetics. *Metabolic Engineering*, 11(1):20–30, 2009.
18. D. Noble. The rise of computational biology. *Nature Reviews Molecular Cell Biology*, 3(6):459–463, 2002.
19. G.M. Oddone, D.A. Mills, and D.E. Block. A dynamic, genome-scale flux model of Lactococcus lactis to increase specific recombinant protein expression. *Metabolic Engineering*, 2009.
20. A. Provost and G. Bastin. Dynamic metabolic modelling under the balanced growth condition. *Journal of Process Control*, 14(7):717–728, 2004.
21. C. Rohr, W. Marwan, and M. Heiner. Snoopy-a unifying Petri net framework to investigate biomolecular networks. *Bioinformatics*, 2010.
22. A. Sackmann, M. Heiner, and I. Koch. Application of Petri net based analysis techniques to signal transduction pathways. *BMC bioinformatics*, 7(1):482, 2006.
23. C.H. Schilling and B.Ø. Palsson. Assessment of the metabolic capabilities of Haemophilus influenzae Rd through a genome-scale pathway analysis. *Journal of Theoretical Biology*, 203(3):249–283, 2000.
24. S. Schuster, T. Pfeiffer, F. Moldenhauer, I. Koch, and T. Dandekar. Exploring the pathway structure of metabolism: decomposition into subnetworks and application to Mycoplasma pneumoniae, 2002.
25. K. Smallbone, E. Simeonidis, D.S. Broomhead, and D.B. Kell. Something from nothing-bridging the gap between constraint-based and kinetic modelling. *FEBS Journal*, 274(21):5576–5585, 2007.
26. K. Smallbone, E. Simeonidis, N. Swainston, and P. Mendes. Towards a genome-scale kinetic model of cellular metabolism. *BMC Systems Biology*, 4(1):6, 2010.
27. P.H. Starke. INA: Integrated Net Analyzer. *Reference Manual*, 1992.
28. G. Stephanopoulos. Metabolic engineering. *Biotechnology and Bioengineering*, 58, 1998.
29. J. Wu and E. Voit. Hybrid modeling in biochemical systems theory by means of functional Petri nets. *Journal of bioinformatics and computational biology*, 7(1):107, 2009.

Applying Petri nets for the analysis of the GSH-ASC cycle in chloroplasts [⋆]

Hermenegilda Macià[*], M. Isabel González-Sánchez[**], Valentín Valero[***], and Edelmira Valero[**]

[*] Department of Mathematics, Escuela Superior de Ingeniería Informática. Albacete. University of de Castilla-La Mancha. Spain
[*] Department of Physical Chemistry, Escuela de Ingenieros Industriales. Albacete. University of Castilla-La Mancha. Spain
[***]Department of Computer Science, Escuela Superior de Ingeniería Informática. Albacete. University of Castilla-La Mancha. Spain
{Hermenegilda.Macia, MIsabel.Gonzalez, Valentin.Valero, Edelmira.Valero}@uclm.es

Keywords: Petri nets, metabolic pathways, biological systems.

Abstract. Petri nets are a useful framework for the analysis of biological systems in various complementary ways, integrating both qualitative and quantitative studies. We apply this formalism to the Glutathione Ascorbate Redox cycle (GSH-ASC) in chloroplasts case study, considering structural Petri net techniques from standard Petri nets to validate the model and to infer new properties, as well as continuous Petri nets in order to have a behavior prediction. In this way, from the continuous Petri net representation we can analyze its behavior under oxidative stress conditions, and from the standard Petri net we can identify some state-conserving or mass-conserving properties.

1 Introduction

Petri nets [8, 10] are a well-known mathematical formalism for the modeling and analysis of concurrent systems. They were introduced by Carl A. Petri [11] in the early 60's. Since this time, they have been extended and applied to several areas [4] such as manufacturing systems, workflow management, telecommunications, communication protocols, etc. Some reasons for using Petri nets are the following: it is easy to describe concurrency and they have a rigorous formal semantics, i.e., their behavior is defined in a precise and unambiguous manner. Additionally, one of the main features of Petri nets is that they have a graphical nature, i.e., you can early get a good knowledge of the system by simple inspection of the Petri net model that represents the system. But, most importantly, there are many tools [21] supporting the model, not only to provide the capability to

[⋆] Supported by the Spanish government (cofinanced by FEDER founds) with the project TIN2009-14312-C02-02, and the JCCM regional projects PEII09-0232-7745, PAI-08-0175-8618 and POII10-0235-8597

create or edit Petri net models, but also to simulate the system evolution and even to analyze some properties of interest. Then, there has been an intensive research in the area of Petri nets in the last 40 years to extend the basic model by including some additional features that are of special interest in some specific application domains. Thus, timed and probabilistic extensions of the basic model have been defined [20, 7], as well as continuous and hybrid Petri nets [1]. The application of Petri nets to the description of chemical processes was already proposed by Carl A. Petri in the 70's [12]. In the 90's Reddy et al. [14] were the first who applied Petri nets to the modeling and analysis of metabolic pathways. Nowadays, there are several different extensions of Petri nets for modeling and simulating biological systems, depending on the specifics of the particular chemical processes described (see [9]). A rich framework for modeling and analyzing biochemical pathways which unifies the qualitative, stochastic and continuous paradigms using Petri nets can be found in [3].

In this paper we consider the GSH-ASC cycle in chloroplasts, which is described and analyzed by using continuous and standard Petri nets. Thus, the main goals of this paper can be summarized as follows:

(i) The application of continuous Petri nets to this specific biological process, which provides us with a graphical representation of this chemical process, which becomes easier to modify and analyze than the corresponding (equivalent) ODE, which can be found in [17].
(ii) The application of the classical theory and tools of Petri nets (in the discrete Petri net), and specifically in this paper the structural theory in order to get a better understanding of the biological model and conclude the relationship between the structural elements (invariants) of the underlying discrete Petri net with the chemical properties of this biological process.

There are two models of the GSH-ASC cycle in the literature: Polle's model [13] and the ours one [17]. Polle's model has some shortcomings that were discussed and improved in our paper. The PN model here described for the GSH-ASC cycle in the continuous case was validated by checking that the same results were obtained with the ODE case [17], as indicated in the (i) goal of the present paper.

The outline of the paper is as follows. Section 2 contains a brief description of the GSH-ASC cycle in chloroplasts. In Section 3 we study the dynamic behavior of this biological model by using continuous Petri nets. Then, the structural qualitative study is presented in Section 4, and finally, the conclusions and hints for future research are presented in section 5.

2 The Biological Model

The glutathione-ascorbate redox (GSH-ASC) pathway in chloroplasts is a complex network of spontaneous, photochemical, and enzymatic reactions for detoxifying hydrogen peroxide. In brief, superoxide dismutase (*SOD*) acts as the first line of defense, dismutating superoxide radical (O_2^-) to H_2O_2 and O_2. In chloroplasts, H_2O_2 thus generated is reduced to water by ascorbate (ASC) catalyzed with L-ascorbate peroxidase (APX). This is the first step of the GSH-ASC cycle, producing monodehydroascorbate radicals (MDA), which spontaneously disproportionate to ASC and dehydroascorbate (DHA). The next step in the cycle is the regeneration of ASC by glutathione (GSH) either enzymatically catalyzed by glutathione dehydrogenase ($DHAR$) or chemically but a too slow rate to account for the observed photoreduction of DHA in chloroplasts. Lastly, the redox cycle is closed by the regeneration of GSH catalyzed by glutathione reductase (GR) at the expense of photoproduced $NADPH$. These steps are captured by the continuous Petri net model depicted in Figure 1, which provides us with a graphical representation of this biological process.

Tables 1-5 provide mathematical expressions for rate equations as well as the conditions (rate constants and initial concentrations) used for the mathematical modeling of the pathway. Due to the lack of space we omit a detailed description of this metabolic pathway, which can be found in [17], which contains a supplementary material section devoted to this description.

It is very difficult to validate numerical data here shown against real biological data. The metabolic pathway under study includes four enzymatic steps and a complex set of photochemical and spontaneous chemical reactions, which is not possible to implement under "in vitro" conditions so that data from Figures 2 and 3 can be tested in an experimental way in the laboratory. However, the values of the kinetic constants and initial conditions used to run the model (Tables 4 and 5) have been taken, when possible, from data reported in the scientific literature, obtained with real systems. APX does not appear on Table 2 because of it has not been considered under steady-state conditions, since it is the most hydrogen peroxide sensitive enzyme in the pathway. Instead, we have introduced its catalytic mechanism including a stage of inactivation by excess of hydrogen peroxide and a stage of de novo synthesis of the protein, which gives the cell the opportunity to recover the amount of APX inactivated, which represents one of the main defense mechanisms of plants to mitigate oxidative stress.

Table 1. Chemical reactions involved in the cycle which have been introduced in the model and notation used for their respective apparent bimolecular rate constants

Reaction	Notation Reaction
$MDA + MDA \rightarrow ASC + DHA$	k_1
$DHA + 2\ GSH \rightarrow ASC + GSSG$	k_4
$2\ O_2^- + 2\ H^+ \rightarrow O_2 + H_2O_2$	k_5
$O_2^- + ASC \rightarrow H_2O_2 + MDA$	k_6
$O_2^- + 2\ GSH \rightarrow H_2O_2 + GSSG$	k_7
$H_2O_2 + 2\ ASC \rightarrow 2\ H_2O + 2\ MDA$	k_8

Table 2. Chemical reactions involved in the APX mechanism

Reaction	Notation Reaction
$APX + H_2O_2 \rightarrow CoI + H_2O$	k_1^{APX}
$CoI + ASC \rightarrow CoII + MDA$	k_2^{APX}
$CoII + ASC \rightarrow APX + MDA$	k_3^{APX}
$CoI + H_2O_2 \rightarrow APX_i$	k_4^{APX}
synthesis de novo of APX	k_5^{APX}

Table 3. Steady-state rate equations used for the enzymes involved in the model

Enzyme	Rate equation
SOD	$k_{SOD}[SOD]_0[O_2^-]$
DHAR	$\frac{k_{cat}^{DHAR}[DHAR]_0[DHA][GSH]}{K_i^{DHA}K_M^{GSH1} + K_M^{DHA}[GSH] + (K_M^{GSH1} + K_M^{GSH2})[DHA] + [DHA][GSH]}$
GR	$\frac{k_{cat}^{GR}[GR]_0[NADPH][GSSG]}{K_M^{NADPH}[GSSG] + K_M^{GSSG}[NADPH] + [NADPH][GSSG]}$

Table 4. List of kinetic constants values used to simulate the model under "standard" conditions.

F	640	k_{cat}^{GR}	595	k_{cat}^{DHAR}	142	k_{SOD}	200	k_1^{APX}	12	k_2^{APX}	50
k_3^{APX}	2.1	k_4^{APX}	0.7	k_5^{APX}	0.01	k_1	0.5	k_4	0.1	k_5	0.2
k_6	0.2	k_7	0.7	k_8	$2E-6$	k_{12}	1.3	k_{13}	42.5	k_N	0.5
K_M^{NADPH}	3	K_M^{GSSG}	200	K_M^{GSH}	2500	K	$5E5$				

Table 5. List of (non-zero) initial concentrations used to simulate the model under "standard" conditions.

Enzymes	Initial concentration (μM)	Species	Initial concentration (μM)
GR	1.4	APX	70
DHAR	1.7	NADPH	150
SOD	50	GSH	4000
		ASC	10000

3 Continuous Petri Nets

We use continuous Petri nets [1], for which places no longer contain integer values, but positive real numbers, and transitions fire in a continuous way. Semantics

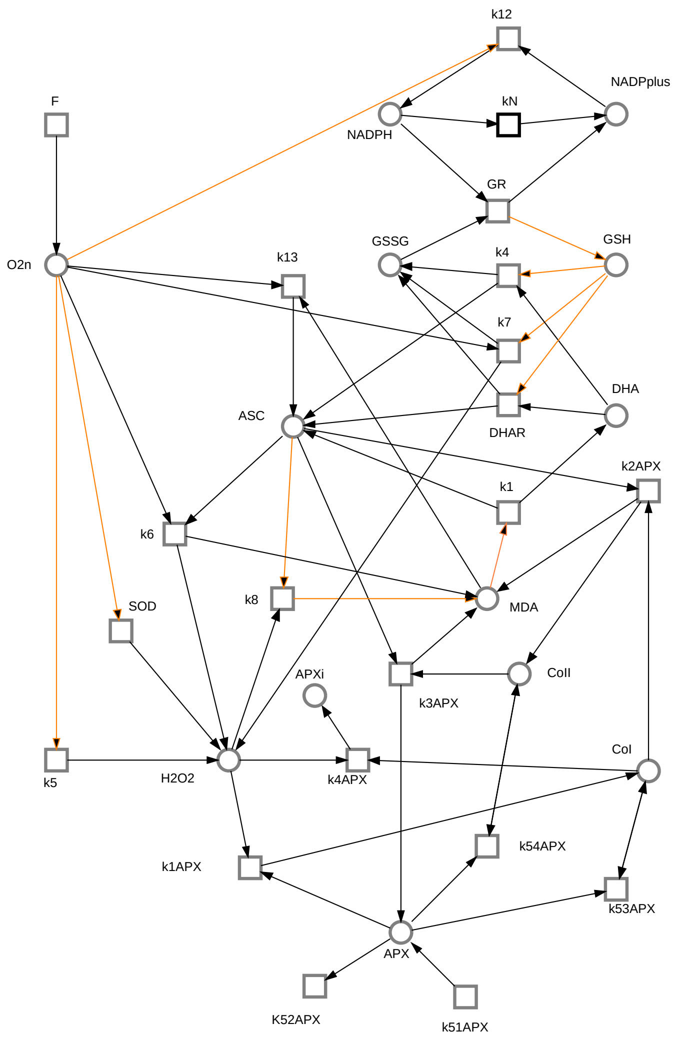


Fig. 1. Continuous Petri net model for the GSH-ASC cycle (orange arcs have weight 2)

of continuous Petri Nets is then defined by means of a system of Ordinary Differential Equations [2]. The continuous Petri net model for the GSH-ASC cycle is that shown in Figure 1 (obtained using the Snoopy tool [16]). This model has been obtained by adapting the biological process described in [17], according to the following considerations:

1. $[CO_2]$ is considered constant, so that the flux of NADPH consumption by the Calvin cycle (and other electron-consuming reactions) is $k_N = k'_N\ [CO_2]$.

2. The synthesis de novo of APX in [17] is considered as $k_5^{APX}([APX]_0 - [APX] - [CoI] - [CoII])$. Then, read arcs are used for CoI and $CoII$ places, since even if k_5^{APX} does not appear in their corresponding equations, it is included in the APX equation. We have then considered four new transitions in the continuous Petri net model, with the following associated kinetic constants: $k51APX$ with $k_5^{APX}[APX_0]$, and $k5APX2$, $k5APX3$ and $k5APX4$ with k_5^{APX}.
3. There is a constant electron source in the model, F, whose flux is divided among three competitive routes: the photoproduction of O_2^- (transition $k5$), the photoreduction of $NADP^+$ (transition $k12$) and the photoreduction of MDA (transition $k13$).

The corresponding ODEs for this continuous Petri net model are those shown in Table 6, which consist of 13 molecular species and 21 reactions defining the equations. These are the same ODEs that we obtained in [17] (supplementary material).

Table 6. Differential equations system

$$\frac{d[NADPH]}{dt} = -v_{GR} - k'_N[CO_2][NADPH] + k_{12}[NADP^+] \quad (1)$$

$$\frac{d[NADP^+]}{dt} = v_{GR} + k'_N[CO_2][NADPH] - k_{12}[NADP^+] \quad (2)$$

$$\frac{d[GSH]}{dt} = 2\,(v_{GR} - v_{DHAR} - k_7[O2^-][GSH] - k_4[DHA][GSH]) \quad (3)$$

$$\frac{d[GSSG]}{dt} = -v_{GR} + v_{DHAR} + k_7[O2^-][GSH] + k_4[DHA][GSH] \quad (4)$$

$$\begin{aligned}\frac{d[ASC]}{dt} = {} & v_{DHAR} + k_1[MDA]^2 + k_4[DHA][GSH] + k_{13}[MDA] \\ & - k_2^{APX}[ASC][CoI] - k_3^{APX}[ASC][CoII] - k_6[O2^-][ASC] \\ & - 2\,k_8[H_2O_2][ASC] \end{aligned} \quad (5)$$

$$\frac{d[DHA]}{dt} = -v_{DHAR} + k_1[MDA]^2 - k_4[DHA][GSH] \quad (6)$$

$$\begin{aligned}\frac{d[MDA]}{dt} = {} & k_2^{APX}[ASC][CoI] + k_3^{APX}[ASC][CoII] - 2\,k_1[MDA]^2 \\ & + k_6[O_2^-][ASC] + 2\,k_8[H_2O_2][ASC] - k_{13}[MDA] \end{aligned} \quad (7)$$

$$\begin{aligned}\frac{d[H_2O_2]}{dt} = {} & v_{SOD} - k_1^{APX}[H_2O_2][APX] - k_4^{APX}[H_2O_2][CoI] + k_5[O_2^-]^2 \\ & + k_6[O_2^-][ASC] + k_7[O2^-][GSH] - k_8[H_2O_2][ASC] \end{aligned} \quad (8)$$

$$\begin{aligned}\frac{d[APX]}{dt} = {} & -k_1^{APX}[H_2O_2][APX] + k_3^{APX}[ASC][CoII] \\ & + k_5^{APX}([APX]_0 - [APX] - [CoI] - [CoII]) \end{aligned} \quad (9)$$

$$\frac{d[CoI]}{dt} = k_1^{APX}[H_2O_2][APX] - k_2^{APX}[ASC][CoI] - k_4^{APX}[H_2O_2][CoI] \quad (10)$$

$$\frac{d[CoII]}{dt} = k_2^{APX}[ASC][CoI] - k_3^{APX}[ASC][CoII] \quad (11)$$

$$\frac{d[APX_i]}{dt} = k_4^{APX}[H_2O_2][CoI] \quad (12)$$

$$\begin{aligned}\frac{d[O_2^-]}{dt} = {} & -2\,v_{SOD} + F - 2\,k_{12}[NADP^+] - 2\,k_5[O_2^-]^2 \\ & - k_6[O_2^-][ASC] - k_7[O2^-][GSH] - k_{13}[MDA] \end{aligned} \quad (13)$$

As a consequence of the photoreduction explained above, the recovery of the reducing power is variable and dependent on the $NADP^+$ and MDA concentrations present in chloroplasts. This provides a great flexibility to the model and a greater ability to study stress conditions. The next values have been used for F:

(i) Unstressed chloroplasts, $F = 640$, giving a production rate of of 222.2 Ms^{-1}, which is within the range previously mentioned in chloroplasts (Figure 2); under these conditions, a steady state was rapidly achieved by the system, in which metabolite concentrations and fluxes remained constant.
(ii) Stressed chloroplasts, $F = 2400$ (intense light exposure), which gives rise to APX photoinactivation (Figure 3); under these conditions, the antioxidant concentration in the chloroplast gradually decreased, in the order NADPH, GSH and ASC, so that their respective oxidized species concentrations increased. The disappearance of ASC was followed by the rapid inactivation of APX, reflecting what occurs in reality, accompanied by a sharp increase in APXi and H_2O_2.

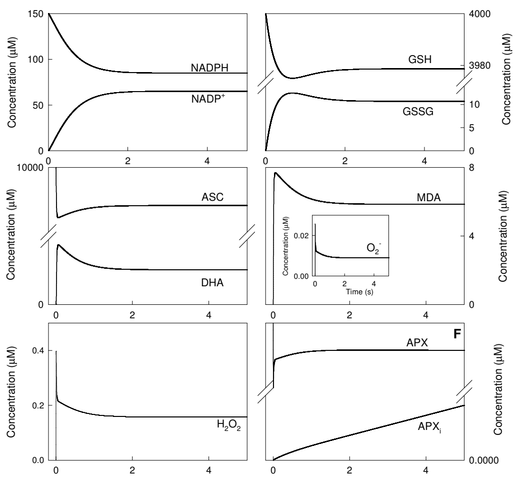


Fig. 2. Simulated progress curves corresponding to the species involved in the mechanism with $F = 640$

4 Structural Analysis

In this section we apply the classical structural techniques on Petri nets [10] in order to verify and analyze the metabolic pathway. For that purpose we build a discrete Petri net model (see Figure 4) from the description of the cycle following the steps described in [2]. We can remove the read arcs (also called test arcs)

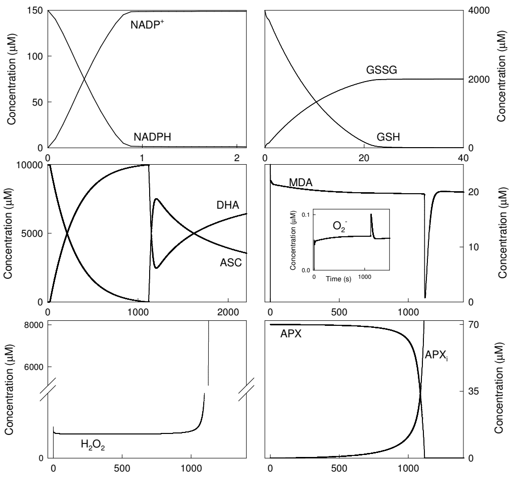


Fig. 3. Simulated progress curves corresponding to the species involved in the mechanism with $F = 2400$

from the continuous Petri net of Figure 1, since they are irrelevant in the corresponding incidence matrix, and therefore in the structural analysis. We then join the transitions *k52APX*, *k53APX* and *k54APX* into a single output transition from *APX*, named *k5APXo*, and we also rename the input transition *k51APX* by *k5APXi*. On the other hand, in order to identify the I/O behavior we add a new place that represents the water generated by the reactions (transitions) *k4APX* and *k1APX*, and a new transition (*outwater*) that models the water self control of chloroplasts.

The obtained Petri net has been analyzed by using a well known Petri net tool, Charlie [15], which allows us to obtain the corresponding invariants for this Petri net.

4.1 P-invariants

A P-invariant defines a mass conservation law and has associated its corresponding biological interpretation. In this case we have obtained three P-invariants (Table 7).

Table 7. P-invariants

$P - inv_1 =$	{ NADPH, $NADP^+$ }
$P - inv_2 =$	{ 2 GSSG, GSH }
$P - inv_3 =$	{ ASC, DHA, MDA }

This means that the pathway under study consists of three moiety-conserved cycles coupled in series to attain a very high amplification capacity [18] against an increase in hydrogen peroxide concentration. In its evolution, since the appearance of oxygen in the atmosphere, the cell has developed a very efficient defense

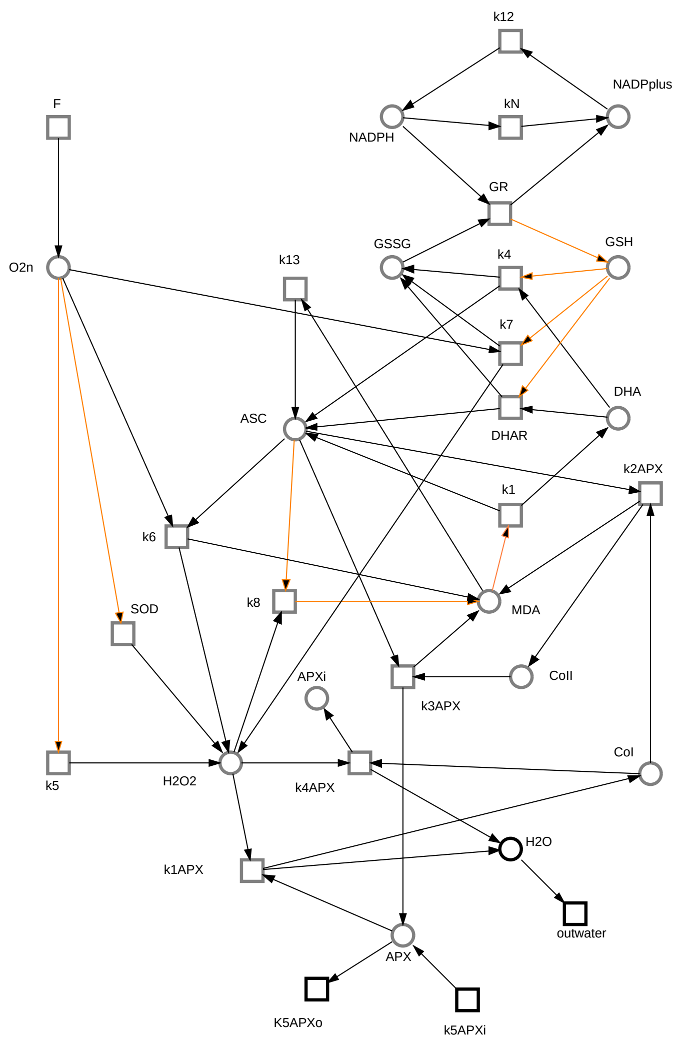


Fig. 4. Petri net model for the GSH-ASC cycle (orange arcs are of weight 2)

tool against oxygen toxicity, although it needs a continuous supply of NADPH. Observe that for each P-invariant there must be a non-zero initial concentration spread along its places, otherwise these places would remain unmarked forever.

(i) $P - inv_1$ captures the consumption of $NADPH$ by the Calvin cycle and GR, and its corresponding recovery in daylight.

(ii) $P - inv_2$ corresponds to the glutahione pool in chloroplasts involving the enzymes GR and $DHAR$. Spontaneous oxidation of GSH in the presence of DHA and O_2^- is also included.

(iii) $P - inv_3$ is related to the interconversion of ASC both spontaneously and catalyzed by $DHAR$ and APX.

4.2 T-invariants

A T-invariant defines a state-conserving subnetwork and has associated its corresponding biological interpretation. In our case (Table 8), using again the Charlie tool, there are 26 minimal semipositive transition invariants.

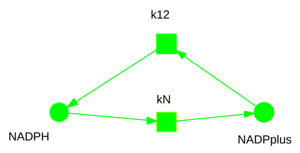


Fig. 5. $T - inv_2$

Let us see a brief description of some of them:

(i) $T - inv_2$ (Fig. 5) is a trivial T-invariant. These transitions capture a reversible reaction, each one modeling a direction in this reaction. Biologically speaking, it corresponds to the consumption and regeneration of $NADPH$ in the two stages of the photosynthesis.
(ii) $T - inv_6$ (Fig. 6) represents in a very clear way the removal of O_2^- and H_2O_2 (reactive oxygen species) by reaction with the reducing agents GSH and ASC, at the expense of the reducing power of $NADPH$.
(iii) $T - inv_7$ (Fig. 6) refers to the catalytic cycle of APX.
(iv) $T - inv_{15}$ (Fig. 7) represents the removal of O_2^- by its spontaneous reduction to H_2O_2 in the presence of ASC, the subsequent removal of H_2O_2 by the catalytic cycle of APX, and the recovery of ASC through the substrate cycling of GSH and $NADPH$.
(v) $T - inv_{24}$ (Fig. 7) is a reflection of the enzymatic steps involved in the pathway: SOD, GR, $DHAR$ and APX.

Another advantage of the Petri net representation is that it can be easily modified for modeling different situations. For instance, in order to consider the same cycle in dark conditions, we only have to remove in Figure 4 the transition F. Then, if we now apply structural analysis we obtain the same three P-invariants, but we only obtain the two first T-invariants, $T - inv_1$ and $T - inv_2$, which are the input (synthesis *de novo*) and output (inactive enzyme) of APX, and the two stages of photosynthesis.

Table 8. T-invariants

T-invariant	Transitions/Reactions (number of fires)
$T - inv_1$	k5APXo (1), k5APXi (1)
$T - inv_2$	kN (1), k12 (1)
$T - inv_3$	k13 (3),k6 (1), k8 (1), F (1)
$T - inv_4$	k13 (2), k8 (1), SOD (1), F (2)
$T - inv_5$	k13 (2), k8 (1), k5 (1), F (2)
$T - inv_6$	GR (1), k12 (1), k7 (1), k13 (2), k8 (1), F (1)
$T - inv_7$	k13 (3), k2APX (1), k3APX (1), k6(1), k1APX (1), F (1), outwater (1)
$T - inv_8$	k13 (2), k2APX (1), k3APX (1), SOD (1), k1APX (1), F (2), outwater (1)
$T - inv_9$	k13 (2), k2APX (1), k3APX (1),k5 (1), k1APX (1),F (2), outwater (1)
$T - inv_{10}$	GR (1), k12 (1), k7 (1), k13 (2), k2APX (1), k3APX(1), k1APX (1), F (1), outwater (1)
$T - inv_{11}$	GR (3), k12 (3), k4 (3), k1 (3), k6 (2), k8 (2), F (2)
$T - inv_{12}$	GR (1), k12 (1), k4 (1), k1 (1), k8 (1), SOD (1), F (2)
$T - inv_{13}$	GR (1), k12 (1), k4 (1), k1 (1), k8 (1), k5 (1), F (2)
$T - inv_{14}$	GR (2), k12 (2), k4 (1), k7 (1), k1 (1), k8 (1), F (1)
$T - inv_{15}$	GR (3), k12 (3), k4 (3), k1 (3), k2APX (2), k3APX (2), k6 (2), k1APX (2), F (2), outwater (2)
$T - inv_{16}$	GR (1), k12 (1), k4 (1), k1 (1), k2APX (1), k3APX (1), SOD (1), k1APX (1), F (2), outwater (1)
$T - inv_{17}$	GR (1), k12 (1), k4 (1), k1 (1), k2APX (1), k3APX (1), k5 (1), k1APX (1), F (2), outwater (1)
$T - inv_{18}$	GR (2), k12 (2), k4 (1), k7 (1), k1 (1), k2APX (1), k3APX (1), k1APX (1), F (1), outwater (1)
$T - inv_{19}$	GR (3), k12 (3), DHAR (3), k1 (3), k6 (2), k8 (2), F (2)
$T - inv_{20}$	GR (1), k12 (1), DHAR (1), k1 (1), k8 (1), SOD (1), F (2)
$T - inv_{21}$	GR (1), k12 (1), DHAR (1), k1 (1), k8 (1), k5 (1), F (2)
$T - inv_{22}$	GR (2), k12 (2), k7 (1), DHAR (1), k1 (1), k8 (1), F (1)
$T - inv_{23}$	GR (3), k12 (3), DHAR (3), k1 (3), k2APX (2), k3APX (2), k6 (2), k1APX (2), F (2), outwater (2)
$T - inv_{24}$	GR (1), k12 (1), DHAR (1), k1 (1), k2APX (1), k3APX (1), SOD (1), k1APX (1), F (2), outwater (1)
$T - inv_{25}$	GR (1), k12 (1), DHAR (1), k1 (1), k2APX (1), k3APX (1), k5 (1), k1APX (1), F (2), outwater (1)
$T - inv_{26}$	GR (2), k12 (2), k7 (1), DHAR (1), k1 (1), k2APX (1), k3APX (1), k1APX (1), F (1), outwater (1)

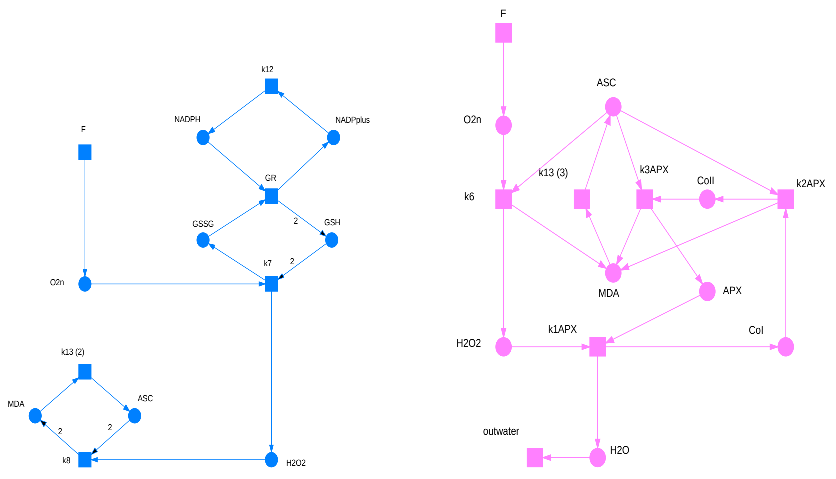


Fig. 6. $T - inv_6$ and $T - inv_7$

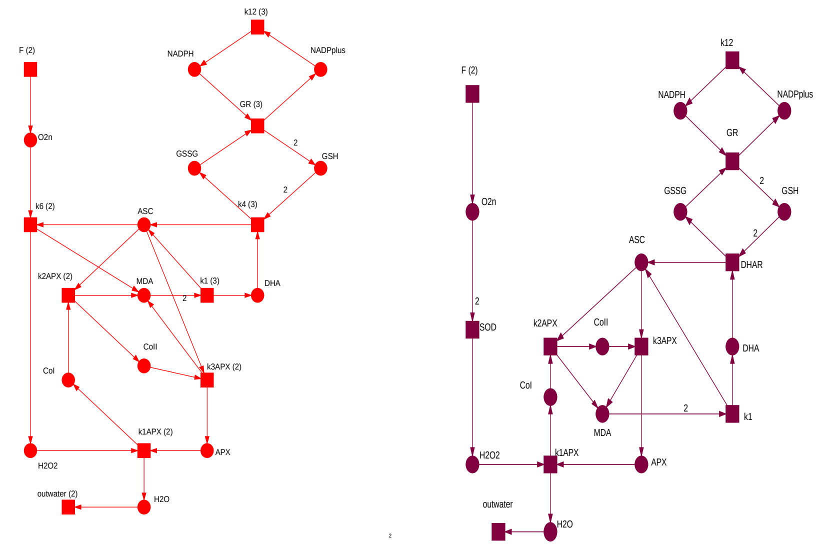


Fig. 7. $T - inv_{15}$ and $T - inv_{24}$

4.3 Core network

We now apply the procedure proposed in [5] in order to identify the core net that represents the network's dynamics. Transition $k4APX$ does not belong to any T-invariant, therefore, it can be removed at the steady-state, together with the place $APXi$, which becomes isolated upon the removal of $k4APX$. Furthermore, $T - inv_2$ is a trivial T-invariant, so that we can use a macro transition for these transitions. Next, we compute the maximal Abstract Dependent Transition (ADT) sets, considering that two transitions depend on each other if they occur always together in the set of T-invariants. In this case we obtain an only connected ADT set $\{k1APX, k2APX, k3APX\}$, which can also be collapsed in a single macro-transition (together with $T - inv_1$). This coarse network (Figure 8) gives us a reduced vision of the chemical process behavior, so it contributes to attain a better understanding of this process, also allowing us to test the robustness and the identification of the fragile nodes.

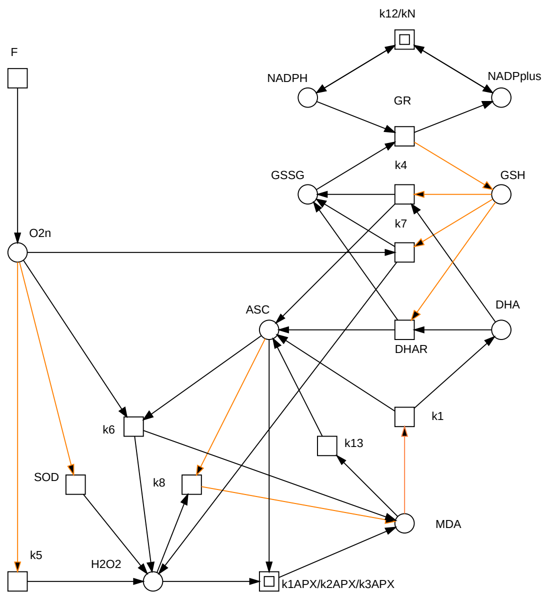


Fig. 8. Coarse Petri net structure of the GSH-ASC cycle

Robustness is defined as the ability of the system to maintain its function against internal and external perturbations [6]. In the pathway under study, robustness is directly related to APX activity [17]. To maintain APX activity, the cell has developed a very efficient defense tool against oxygen toxicity, based on two coupled substrate cycles: *GSH-GSSG* ($P - inv_2$) and *ASC-MDA-DHA* ($P - inv_3$), although it needs a continuous supply of *NADPH* ($P - inv_1$). Substrate cycles are powerful metabolic tools involving two enzymes acting in opposite directions, whereby a target metabolite is reversibly interconverted into another chemical species without being consumed [19]. The physiological explanation proposed for this wasteful cycling is that is mainly a way of amplifying a metabolic response to a change in a metabolic concentration, thus greatly improving the sensitivity of metabolic regulation. The waste of $NADPH$ can the be understood as the cost that chloroplasts must pay to swiftly detoxify H_2O_2 and O_2^-.

It is also very important to analyze the redundancy of a pathway. It is the hallmark of biological networks where the very same function is carried out by different pathways, which provides robustness against perturbations like mutation. In the *GSH-ASC* cycle, if a mutation block *SOD*, there is a parallel spontaneous step for ${O_2}^-$ dismutation (k_5), as can been seen in Figure 8. The same holds for *DHAR* (k_4), in such a way that chloroplasts can recover the reducing power necessary to detoxify reactive oxygen species in the absence of these enzymes. Redundancy of the pathway under study is clearly revealed by comparison of $T - inv_{15}$ and $T - inv_{24}$, which represent the chemical and the enzymatic pathways, respectively, to eliminate H_2O_2.

Another information that is teased out from the coarse network is that *NADPH* is the shared node for two pathways: the *Calvin* cycle and the *GSH-ASC* cycle. If the recovery of *NADPH* is silenced, it results in a complete loss of function of both pathways in the core network indicating that it is indeed the fragile node in the network. The pathway under study is very interesting, since the same day-light that gives rise to O_2^- radicals also generates $NADPH$ and *ASC* to detoxify H_2O_2 arising from O_2^- dismutation. Therefore it is very important to know the relative weight of each route in the growth conditions of plants.

5 Conclusions and Future Work

The GSH-ASC cycle in chloroplasts has been modelled using continuous and discrete Petri nets. For that purpose, we have defined the specific continuous Petri net model that corresponds to the network of chemical and enzymatic steps involved in the cycle, and we have studied it in two ways: the quantitative one, which helps us to make a prediction behavior; and the qualitative one, applying structural techniques, considering the core structure. We have obtained their corresponding biological interpretation that help us to understand this biological system.

As future work we intend to add some additional steps into the pathway, which would be helpful to have a better understanding of the biological behavior considering some new features, such as dark-light interactions. We also intend to apply other known formal techniques to the study of the GSH-ASC cycle in chloroplasts, for instance, we may apply model checking techniques in order to conclude whether a certain property is fulfilled or not by the chemical system. Finally, it can also be of interest to derive probabilistic informations from a chemical system, i.e., we can use a probabilistic framework, like stochastic Petri nets (SPNs), for the modeling of the GSH-ASC cycle in chloroplasts, and derive the relevant stochastic information of the system.

References

1. R. David and H. Alla. *Discrete, Continuous and Hybrid Petri Nets.* Springer, 2005.
2. D. Gilbert and M. Heiner. *From Petri Nets to Differential Equations - an Integrative Approach for Biochemical Network Analysis.* Proc. ICATPN 2006, Turku, June, Springer LNCS 4024, pp. 181-200, 2006.
3. D. Gilbert, M. Heiner, and S. Lehrack. *A Unifying Framework for Modelling and Analysing Biochemical Pathways Using Petri Nets.* Proc. of Computational Methods in Systems Biology 2007, CMSB 2007, M. Calder and S. Gilmore (eds), Springer, LNCS/LNBI 4695, pp.200-216, 2007.
4. C. Girault and R. Valk. *Petri Nets for Systems Engineering. A Guide to Modelling, Verification, and Applications.* Springer, 2003.
5. M. Heiner and K. Sriram. *Structural Analysis to Determie the Core of Hypoxia Response Network.* PLoS ONE 5(1): e8600, doi:10.1371/journal.pone.0008600, 2010.
6. H. Kitano. *Biological robustness.* Nature Reviews Genetics, Vol. 5, pp. 826-837, 2004.
7. M. Kudlek. *Probability in Petri Nets.* Fundamenta Informaticae, vol 67,1-3, pp.121-130, 2005.
8. T. Murata. *Petri Nets: Properties, Analysis and Applications.* Poc. of IEEE, vol. 77, 4, pp.541-580, 1989.
9. M. Peleg, D. Rubin, and R.B. Altman. *Using Petri Net Tools to Study Properties and Dynamics of Biological Systems.* Journal of the American Medical, vol 12, 2, pp.181-199, 2005.
10. J.L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, 1981.
11. C.A. Petri. *Kommunication mit Automaten.* PhD thesis, Schriften des IMN, Institut für Instrumentelle Mathematic, Bonn, 1962.
12. C.A. Petri. *Interpretations of Net Theory.* GMD, Interner Bericht, 2nd improved edition, 1976.
13. A. Polle. *Dissesting the superoxide dismutase-ascorbate-glutathione-pathway in chloroplasts by metaboloc modeling: computer simulations as a step towards flux analysis.* Plant Physiology, Vol. 126, pp. 445-461, 2001.
14. V.N. Reddy, M.L. Mavrovouniotis, and M.N. Liebman. *Petri Net Representation in Metabolic Pathways.* In Proc. First International Conference on Intelligent Systems for Molecular Biology, pp. 328û336, Menlo Park. AAI, 1993.
15. Data Structures University of Technology in Cottbus, Dep. of Computer Science and Software Dependability. *Charlie.* 2010. http://www-dssz.informatik.tu-cottbus.de/.
16. Data Structures University of Technology in Cottbus, Dep. of Computer Science and Software Dependability. *Snoopy.* 2009. http://www-dssz.informatik.tu-cottbus.de/.
17. E. Valero, M. I. González-Sánchez, H. Macià, and F. García-Carmona. *Computer Simulation of the Dynamic Behavior of the Glutathione-Ascorbate Redox Cycle in Chloroplasts.* Plant Physiology, Vol. 149, pp. 1958-1969, 2009.
18. E. Valero, R. Varón, and F. García-Carmona. *Kinetic analysis of a model for double substrate cycling: highly amplified ADP (and/or ATP) quantification.* Biophysical Journal, Vol. 86, pp. 3598-3606, 2004.
19. Valero, E. and Varón, R. and García-Carmona, F. *Kinetics of a self-amplifying substrate cycle: ADP-ATP cycling assay.* Biochem J 350, pp. 237-243, 2000.
20. J. Wang. *Timed Petri nets: theory and application.* Kluwer international series on discrete event dynamic systems, vol.9, 1998.
21. Petri Nets World. *http://www.informatik.uni-hamburg.de/TGI/PetriNets/.*

Petri Net Modeling via a Modular and Hierarchical Approach Applied to Nociception

Mary Ann Blätke[1], Sonja Meyer[1], Christoph Stein[2] and Wolfgang Marwan[1]

[1]Otto-von-Guericke Universität Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany
[2]Dep. Anesthesiology, Charité - Freie Universität Berlin, Hindenburgdamm 30, 12200 Berlin, Germany
wolfgang.marwan@ovgu.de

Abstract. We describe signal transduction of nociceptive mechanisms involved in chronic pain by a qualitative Petri net model. More precisely, we investigate signaling in the peripheral terminals of dorsal root ganglion (DRG) neurons. It is a first approach to integrate the current neurobiological and clinical knowledge about nociception on the molecular level from literature in a model describing all the interactions between the involved molecules.
Due to the large expected total size of the model under development, we employed a hierarchical and modular approach. In our entire nociceptive network, each biological entity like a receptor, enzyme, macromolecular complex etc. is represented by a self-contained and functional autonomous Petri net, a module.
Analysis of the Petri net modules and simulation studies ensure the fulfillment of criteria important for biological Petri nets and the ability to represent the modeled biological function.

Key words: Petri net, qualititative approach, module, pain, nociception, G-protein-coupled receptor, large biological systems

1 Introduction

Clinical pain is a very complex phenomenon with behavioural, peripheral and central nervous system components. Often, pain can not be successfully treated due to the lack of knowledge about the molecular basis on which pain killers take effect. A mechanism-based pain therapy is largely missing, rendering undertreated pain a serious public health issue (see [7] and references therein).
At the molecular level, many extracellular stimuli and substances in the peripheral tissue are known that provoke nociceptive signaling in DRG neurons and subsequent pain (a complex sensation resulting from integration of peripheral and central messages). A variety of membrane components and intracellular signaling molecules have been identified that play key roles in pain sensation. Examples are G-protein-coupled receptors (GPCR), ion channels, receptor tyrosine kinases, cytokine and hormone receptors, which in turn activate a plethora

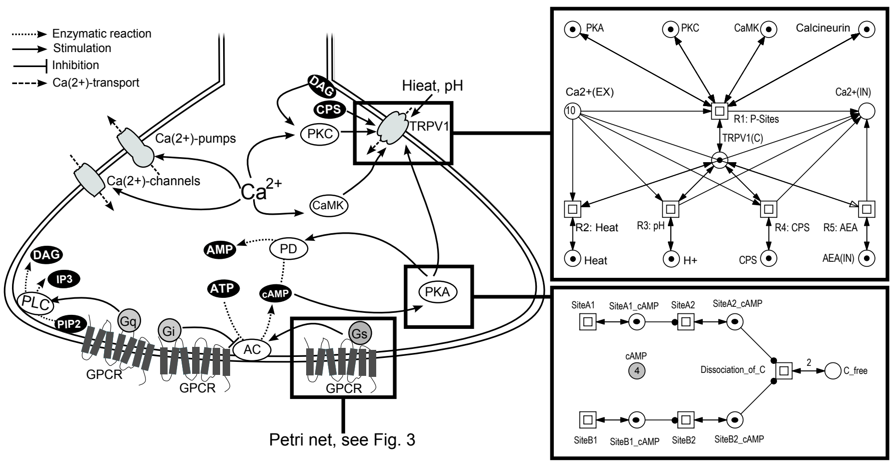


Fig. 1: *Left:* Signaling components in nociceptors. Nociception is triggered by a large number of extracellular signals acting through several receptor classes and initiating a plethora of intracellular signaling cascades. *Right:* Molecular entities like receptors, enzymes (and other biomolecules or macro-molecular complexes) are represented as functional units in the form of self-contained and functionally autonomous Petri nets. The subnets can be coupled by shared places representing identical, common components.

of signaling cascades like the cAMP pathway and calcium signaling [6, 7] (see Fig. 1). However, the quantitative and qualitative relationships between the different intracellular signaling mechanisms acting downstream of the receptor to which those substances bind are still poorly understood [7].
It seems straightforward to apply the Petri net framework to study pain signaling 'in silico', because Petri nets are designed for concurrent systems and also were shown to be ideally suited to model biological systems [9].
For the description of the nociceptive network we choose qualitative modeling as the preceding step for simulation studies which can be performed either stochastically or continuously. It has been shown that a continuous Petri net is equivalent to a structured description of ODEs [9]. However, it is known that many of the involved processes are inherently stochastic. Due to this reason, we prefer stochastic simulations studies to validate our model. The extension of the entire qualitative Petri net to a stochastic one with parameters from experimental data is not possible at the moment because kinetic information of nociceptive mechanisms is hitherto largely missing.
In our modular approach, a module represents a biological functional entity like a receptor, a channel, an enzyme or a macro-molecular complex in form of a self-contained and functional autonomous Petri net graph. The places of a module correspond to functional domains (binding domains, phosphorylation sites, autoinhibitoy domains etc.). These functional domains are regulated by other biological entities and second messengers or are responsible for the effector func-

tion. Thus, transitions stand for actions (dissociation, binding, phosphorylation etc.) occurring within a biological entity. There exist no input or output transitions (sources or sinks of a certain molecule). Due to mass conservation and the fact that a molecular entity is not used up by signaling, the corresponding Petri net graph must be covered with P-invariants [9]. Likewise, the Petri net graph of a module should be bounded to ensure that biological entities, second messengers, precursors, degradation products and energy equivalents do not accumulate. The coverage of T-Invariants of the whole module is not necessary due to the limitation of components which take part in the regulation of the module or which are substrates for the effector function. Therefore, the fulfillment of properties like liveness, reversibility and no dead states it is not mandatory. In contrast, substructures of the modules where reversible changes occur should be covered with T-invariants to assure that the initial state of the involved domains can be restored. Ideally, the computed T-invariants have to be covered by P-invariants [9]. Both, T- and P-invariants, correspond to important biological functions. The up and down regulation of molecular entities by others and second messengers should be reflected in the token flow of the module especially in the increase or decrease of its effector function.

2 Goal

Our goal is to represent nociceptive mechanisms in DRG neurons in a single, coherent Petri net and to establish relationships between signaling components. With the help of simulations we aim at reproducing effects of known nociceptive stimuli correctly and attempt to predict effects of specific perturbations (drugs for therapeutic interventions).
We also aim to establish a module repository. A major advantage is that the modules can be variably combined and reused in other systems according to the requirements of specific 'wet lab' or 'in silico' experiments.

3 Method

We collected literature about nociceptive signaling in DRG neurons, the most investigated cell type in pain-related studies at the molecular level. We extracted those nociceptive signaling components from the literature, whose molecular interaction with other pain-related components is well described and proven by experiments. Further, we searched in detail for the regulatory and effector functions of each of those molecules.
Subsequently, we translated each biological functional unit into a Petri net using the qualitative approach, see e.g. [1, 2, 9, 8]. We used time-free transitions and obtained a time-free Petri net accordingly [9]. Our nets were constructed with Snoopy, a tool to design and animate hierarchical graphs [13].
Each qualitative Petri net is finally subjected to a comprehensive analysis. Here, we apply all validation criteria for biochemical pathway models given in [9]. Therefore, we determine behavioural properties like liveness, reversibility and

boundness, as well as P- and T-invariants. The analyses have been performed using the software Charlie, a software tool to analyse place/transition nets [11]. Having successfully validated the qualitative model, we perform stochastic simulations by assigning stochastic rate functions to all reactions in the network to study the dynamic behaviour of the systems in terms of the flow of token in our model. In particular, we used the stochastic biomass action function, which is available in Snoopy, together with a simple test parameter sets. In these sets, the firing rates of transitions inactivating the effector function of a molecule are assumed to be lower compared to those of transitions activating the effector function (also see section 5).

4 Nociceptive Network

The entire nociceptive network is build by connection of the constructed modules. Here, places sharing the same molecules/molecular complexes (logical places) constitute the natural connections between the modules.
Currently, we have constructed approximately 40 modules on the basis of 251 scientific articles [8]. We expect that at least twice as many modules are required for a comprehensive description of the entire nociceptive network on the basis of the current knowledge.
This expected total size of the model under development precludes a flat representation. Thus, a modeling approach is applied, which yields immediately a hierarchically structured model. So far, the latest version of the entire network consists of 22 connected modules, the representation is distributed over 67 pages with a nesting depth up to 4, compare Fig. A.1 in the appendix. The model consists of about 300 places and 350 transitions.

5 Example for a Module : G-Protein-coupled Receptor

In this section, we representatively describe the construction and structural analysis of one functional unit of our entire net, the G-protein-coupled receptor (GPCR), a typical seven-helix-transmembrane receptor.
GPCRs relay external signals by activating heterotrimeric guanine-nucleotide-binding proteins (G-protein). Seven-helix receptors form the largest family of transmembrane receptors and are therefore crucial components in many signal cascades including nociceptive pathways. There are several GPCRs in nociception interacting specifically with endogenous and exogenous opioids, cannabinoids or substances released as a result of inflammation (e.g. bradykinin), thus having substantial modulating effects on pain sensation. A heterotrimeric G-protein consists of α[1], β and γ subunits (see also [3–5]). Fig. 2 shows the interaction of GPCR with coupled G-protein.

[1] The Gα subunit occurs in three main isoforms with distinct functions: Gα_s (stimulation of adenylyl cyclases), Gα_i (inhibition of adenylyl cyclases)and Gα_q (stimulation

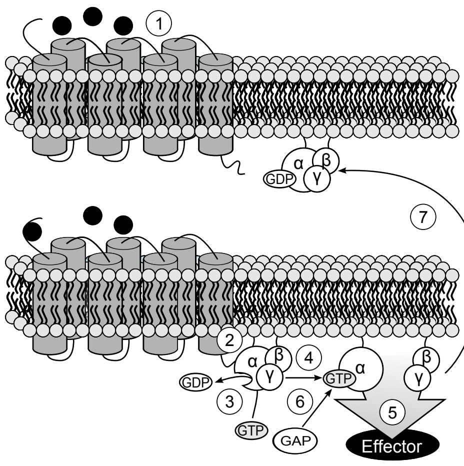


Fig. 2: Regulation of GPCR and its coupled G-protein (see also [3–5]): The activation of a GPCR occurs by binding of a specific ligand at the extracellular side *(step 1)* causing a conformational change *(step 2)*, which activates the recruited resting G-protein in its GDP-bound form. This causes the exchange of GDP by GTP in the specific binding pocket of the Gα subunit. *(step 3)*. Gα subunits with GTP bound dissociate from the G-protein complex *(step 4)* and act on further downstream signal molecules like adenylyl cyclase or phospholipase C β *(step 5)*. The remaining Gβ/γ subunit in addition causes multiple regulatory effects mostly on ion channels and on isoforms of adenylyl cyclases *(also step 5)*. The effector function of the Gα subunit is terminated by the binding of a GTPase activating protein (GAP) stimulating the intrinsic GTPase function of the Gα subunit. GTP is hydrolysed to GDP *(step 6)*. The GDP bound form of the Gα subunit then reassociates with the Gβ/γ subunit to assume its initial pre-stimulus state *(step 7)*.

The regulatory mechanisms and effector functions of GPCRs and the associated G-proteins are translated into a place/transition Petri net (see Fig. 3).
Places may either represent individual molecules or functional states of more complex molecules. Places that are connected by two opposite edges (in this example replaceable by read arcs) with a transition represent molecules or states, which are necessary for a signaling event to occur without being consumed by the reaction. Transitions describe biochemical reactions and molecular interactions.
To provide a neat arrangement of the Petri net, we used coarse transitions (double squares), integrated at the top level. The entire (flattened) place/transition Petri net of this submodel consists of 27 places and 17 transitions connected by 72 edges.
Computation of the invariants shows the coverage of the net by P- and partly by T-invariants (see Fig. 4). Furthermore, there are no invariants without biological

of phospholipase d β). GPCR are mostly associated with one particular G-protein isoform.

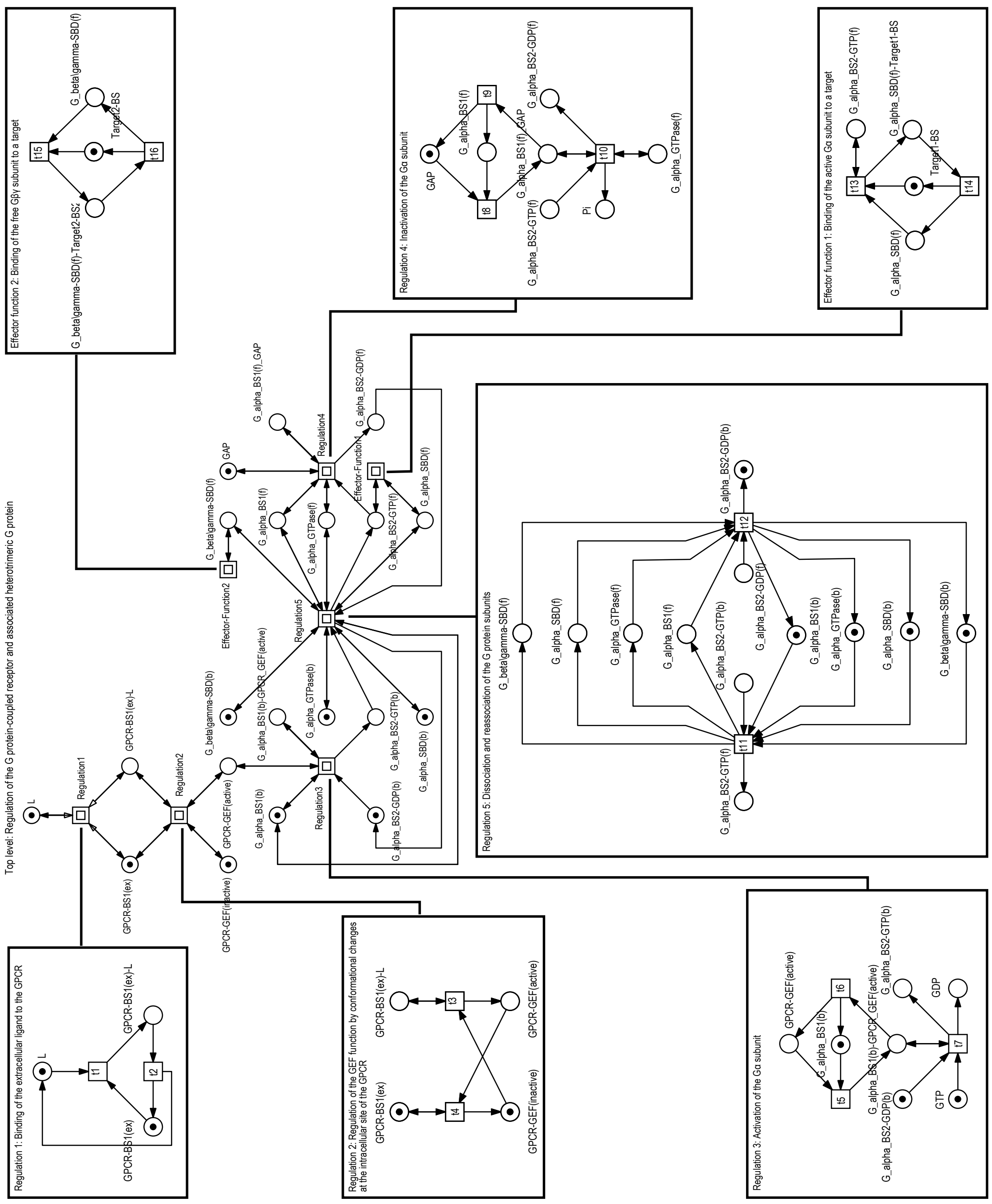

Top level: Regulation of the G protein-coupled receptor and associated heterotrimeric G protein
Regulation 1: Binding of the extracellular ligand to the GPCR
Regulation 2: Regulation of the GEF function by conformational changes at the intracellular site of the GPCR
Regulation 3: Activation of the Gα subunit
Regulation 4: Inactivation of the Gα subunit
Regulation 5: Dissociation and reassociation of the G protein subunits
Effector function 1: Binding of the active Gα subunit to a target
Effector function 2: Binding of the free Gβγ subunit to a target

Fig. 3: Petri net module representing GPCR and G-protein regulation: The top level in the center represents all functional sites of GPCR and G-protein which take part in the regulation and effector function. The surroundig Petri nets show the respective coarse transitions in detail.

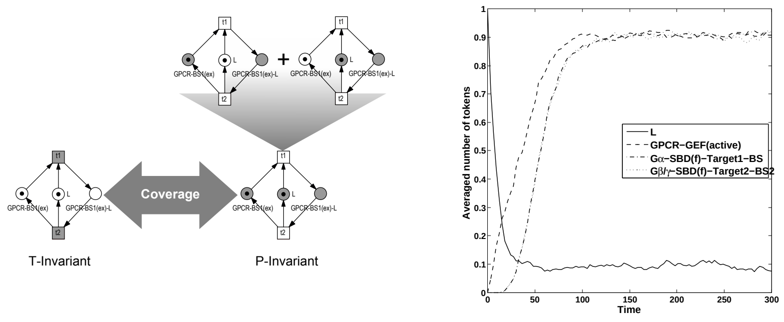


Fig. 4: *Left:* Result of the structural analysis. Shown are T-Invariant 1 (with transitions t1, t2) and P-Invariant 1 (with places GPCR-BS1(ex), GPCR-BS1(ex)-L), 2 (with places L, GPCR-BS1(ex)-L). For the biological meaning of these invariants and all others of the GPCR module see Tab. A.1 and Tab. A.2 in the appendix. *Right:* Result of a stochastic simulation run with test firing rates.

meaning (see Tab. A.1 and Tab. A.2 in the appendix). Thus, essential validation criteria for a Petri net model of a signal transduction network are fulfilled. Stochastic simulations with test rates show the expected effector function of the module. The dissociation of the ligand from GPCR (t2) and the dissociation of the targets from the substrate binding sites of both G-protein subunits (t16, t18) are assumed to proceed slower *(BioMassAction(0.01))* than all other reactions *(BioMassAction(0.1))*. Upon ligand binding to the receptor (decrease of free ligand), we first observe an increase in the activated GPCR GEF function, followed by an increase of the dissociated G-protein subunits, which can subsequently trigger downstream signaling events.

6 Conclusion

Models allow to perform experiments 'in silico', to study the systems properties and behaviour, to make predictions and thus to contribute to a further understanding of the involved processes. As the body of biological data is steadily increasing, it becomes more and more important to find a way to integrate huge amounts of available information in the form of a model. We are currently working on a method consisting of a modular design principle that allows to check and validate each functional subunit thorougly due to its managable size. Step by step connection and combination of subunits (in the form of submodels) and validation of the connected parts ensures that the resulting composed net is coherent as well. Depending on specific 'wet lab' experiments, which are performed to validate the model in turn, different modules can be combined in order to study the behaviour of subsystems or of the entire system that has been modeled. As many biological functional units (like enzymes, receptors) play a role in different signaling pathways, the respective modules can be reused and recombined in different ways. The modules can be applied to other Petri net

classes; they can be easily converted into a colored Petri net [12], or a stochastic Petri net, see intoduction. In a next step we intend to color our low-level Petri net [12] in cooperation with the group of Prof. Heiner. This more compact description will enable us to depict and study the behavior of populations of nociceptive DRG neurons as well as multiple copies of biological entities.
As far as pain and the contribution of nociceptors is concerned, we hope to contribute with our net to a mechanism-based pain therapy by identifying possible targets for the development of new therapeutic intervention strategies.
The modular design together with the Petri net framework seems to be a promising tool to handle large biological systems even when exact quantitative parameter values are missing.

7 Acknowledgements

This work is supported by the Modeling Pain Switches (MOPS) program of Federal Ministry of Education and Research (Funding Number: 0315449F). We thank Prof. Heiner for the outstanding support and cooperation during this work.

References

1. Reisig, W.: Petri Nets; An Introduction. Springer (1982)
2. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proc.of the IEEE. 4, 541-580 (1989)
3. Strader, C. et al.: Structure and Function of G-Protein-coupled Receptors. Annual Review of Biochemistry 63, 101-132 (1994)
4. Lambright, D. et al.: The 2.0 Crystal Structure of a Heterotrimeric G-Protein. Nature 379, 311 - 319 (1996)
5. Ross, E. and Wilkie T.: GTPase-activating G-Proteins for Heterotrimeric G-Proteins: Regulators of G-Protein Signaling (RGS) and RGS-like Proteins. Annual Review of Biochemistry 69, 795-827 (2000)
6. McMahon, S. and Koltzenburg, M.: Textbook of Pain. Churchill Livingstong (2005)
7. Hucho, T. and Levine, J.: Signaling Pathways in Sensitization: Toward a Nociceptor Cell Biology. Neuron 55, 365-376 (2007)
8. Blätke, MA.: Petri-Netz-Module eines Integrativen Nozizeptiven Neurons. Otto von Guericke University Magdeburg (2009)
9. Heiner, M., Gilbert, D., and Donaldson, R.: Petri Nets in Systems and Synthetic Biology. In Schools on Formal Methods (SFM), LNCS 5016 Springer. , 215-264 (2009)
10. Stein, C. and Lang, L.J.: Peripheral Mechanisms of Opioid Analgesia. Current Opinion in Pharmacology 9, 3-8(2009).
11. Franzke, A.: Charlie 2.0 - A Multithreaded Petri Net Analyzer. Brandenburg University of Technology Cottbus (2009)
12. Liu, F. and Heiner, M.: Using Colored Petri Nets to Model and Simulate Biological Systems. Same Issue (2010)
13. Rohr, C., Marwan, W. and Heiner, M.: Snoopy - A Unifying Petri Net Framework to Investigate Biomolecular Networks (2010)

Tab. 1: List of P-invariants and their interpretation

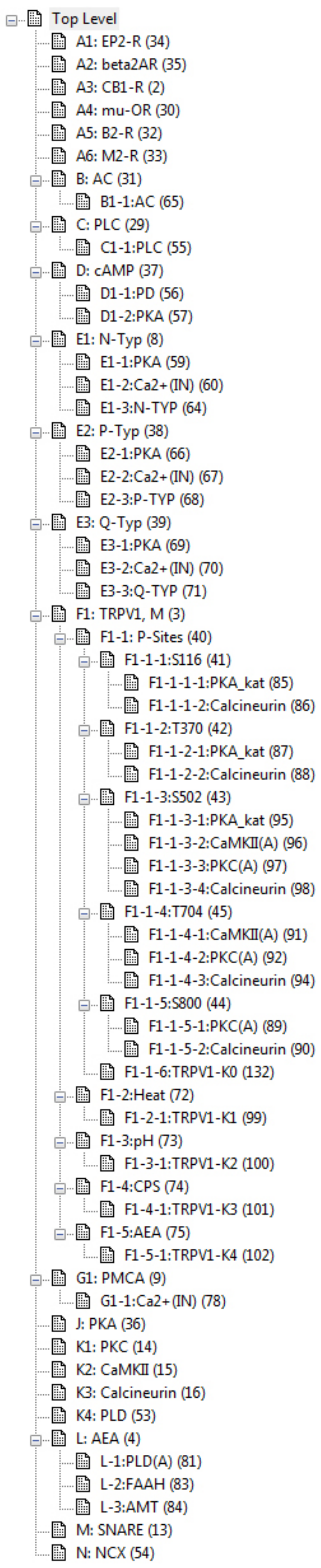
Number	Place	Interpretation
1	GPCR-BS1(ex) GPCR-BS1(ex)-L	Extracellular binding site of the GPCR is unbound or bound to the ligand.
2	GPCR-BS1(ex)-L L	The ligand is free in the extracellular space or bound to the GPCR.

3	GPCR-GEF(active)
	GPCR-GEF(inactive)
	Gα-BS1(b)-GPCR-GEF(active)
		GEF part of the GPCR can be inactive or active or active and bound to the G-protein.

4	GAP *Gα*-BS1(f)-GAP	GAP is free in cytoplasma or bound to the G-Protein.
5	Gα-SBD(f)-Target1-BS Target1-BS	The target for the Gα subunit is free or bound to Gα substrate binding domain.
6	Gβ/γ-SBD-Target2-BS2 Target2-BS	The target for the Gβ/γ subunit is free or bound to Gβ/γ substrate binding domain.
7	Gα-GTPase(b) Gα-GTPase(f)	The confromation of the GTPase domain corresponds to that of the whole Gα subunit .
8	Gα-BS2-GDP(b) Gα-BS2-GDP(f) Gα-BS2-GTP(b) Gα-BS2-GTP(f)	The same as above goes for binding site 2 of the Gα subunit. In both cases GTP or GDP is bound.
9	Gα-BS1(b) Gα-BS1(b)-GPCR-GEF(active) Gα-BS1(f) Gα-BS1(f)-GAP	The same as above goes for binding site 1 of the Gα subunit. In both cases it can be unbound or bound to GAP respectively the GEF part of the GPCR.
10	Gβ/γ-SBD(b) Gβ/γ-SBD(f) Gβ/γ-SBD(f)-Target2-BS2	The Gβ/γ subunit can be associated to the G-protein complex (no substrate binding) or free (substrate binding possible).
11	Gα-BS1(b) Gα-BS1(b)-GPCR-GEF(active) Gα-SBD(f) Gα-SBD(f)-Target1-BS	If one domain is in the conformation where the Gα subunit is associated to the G-protein complex another domain can not be in the comformation where the Gα subunit is free (vice versa).
12	Gα-BS1(f) Gα-BS1(f)-GAP Gα-BS2-GDP(b) Gα-BS2-GTP(b)	see no. 11
13	Gα-BS2-GDP(b) Gα-BS2-GTP(b) Gα-SBD(f) Gα-SBD(f)-Target1-BS	see no. 11
14	Gα-SBD(b) Gα-GTPase(f)	see no. 11
15	Gα-BS1(f) Gα-BS1(f)-GAP	see no. 11

	Gα-SBD(b)	
16	Gα-BS2-GDP(f) Gα-BS2-GTP(f) Gα-SBD(b)	see no. 11
17	Gα-SBD(b) Gα-SBD(f) Gα-SBD(f)-Target1-BS	see no. 11
18	Gα-BS1(f) Gα-BS1(f)-GAP Gα-GTPase(b)	see no. 11
19	Gα-BS2-GDP(f) Gα-BS2-GTP(f) Gα-GTPase(b)	see no. 11
20	Gα-SBD(f) Gα-SBD(f)-Target1-BS Gα-GTPase(b)	see no. 11
21	Gα-BS1(b) Gα-BS1(b)-GPCR-GEF(active) Gα-GTPase(f)	see no. 11
22	Gα-BS2-GDP(b) Gα-BS2-GTP(b) Gα-GTPase(f)	see no. 11
23	Gα-BS1(b) Gα-BS1(b)-GPCR-GEF(active) Gα-BS2-GDP(f) Gα-BS2-GTP(f)	see no. 11
24	Gα-BS1(b) Gα-BS1(b)-GPCR-GEF(active) Gβ/γ-SBD(f) Gβ/γ-SBD-Target2-BS2	If the substrate binding domain is in the conformation where the Gβ/γ subunit is associated to the G-protein complex, another domain can not be in the comformation where Gα subunit is free (vice versa).
25	Gα-SBD(f) Gα-SBD(f)-Target1-BS Gβ/γ-SBD(b)	see no. 24
26	Gα-BS1(f) Gα-BS1(f)-GAP Gβ/γ-SBD(b)	see no. 24
27	Gα-BS2-GDP(f) Gα-BS2-GTP(f) Gβ/γ-SBD(b)	see no. 24
28	Gα-BS2-GDP(b) Gα-BS2-GTP(b) Gβ/γ-SBD(f) Gβ/γ-SBD-Target2-BS2	see no. 24
29	Gα-GTPase(f) Gβ/γ-SBD(b)	see no. 24
30	Gα-GTPase(b) Gβ/γ-SBD(f)	see no. 24

	Gβ/γ-SBD(f)-Target2-BS2	
31	Gα-SBD(b) Gβ/γ-SBD(f) Gβ/γ-SBD-Target2-BS2	see no. 24
32	GDP GTP	Free GTP can just be in a high or low engergy state
33	GTP Gα-BS2-GTP(b) Gα-BS2-GTP(f) Pi	The high energy state of GTP can just be free, bound at the free Gα subunit or at Gα subunit in the G Protein complex. If GTP is in one of those states there cannot be free Pi (vice versa)



```
Top Level
    A1: EP2-R (34)
    A2: beta2AR (35)
    A3: CB1-R (2)
    A4: mu-OR (30)
    A5: B2-R (32)
    A6: M2-R (33)
    B: AC (31)
        B1-1:AC (65)
    C: PLC (29)
        C1-1:PLC (55)
    D: cAMP (37)
        D1-1:PD (56)
        D1-2:PKA (57)
    E1: N-Typ (8)
        E1-1:PKA (59)
        E1-2:Ca2+(IN) (60)
        E1-3:N-TYP (64)
    E2: P-Typ (38)
        E2-1:PKA (66)
        E2-2:Ca2+(IN) (67)
        E2-3:P-TYP (68)
    E3: Q-Typ (39)
        E3-1:PKA (69)
        E3-2:Ca2+(IN) (70)
        E3-3:Q-TYP (71)
    F1: TRPV1, M (3)
        F1-1: P-Sites (40)
            F1-1-1:S116 (41)
                F1-1-1-1:PKA_kat (85)
                F1-1-1-2:Calcineurin (86)
            F1-1-2:T370 (42)
                F1-1-2-1:PKA_kat (87)
                F1-1-2-2:Calcineurin (88)
            F1-1-3:S502 (43)
                F1-1-3-1:PKA_kat (95)
                F1-1-3-2:CaMKII(A) (96)
                F1-1-3-3:PKC(A) (97)
                F1-1-3-4:Calcineurin (98)
            F1-1-4:T704 (45)
                F1-1-4-1:CaMKII(A) (91)
                F1-1-4-2:PKC(A) (92)
                F1-1-4-3:Calcineurin (94)
            F1-1-5:S800 (44)
                F1-1-5-1:PKC(A) (89)
                F1-1-5-2:Calcineurin (90)
            F1-1-6:TRPV1-K0 (132)
        F1-2:Heat (72)
            F1-2-1:TRPV1-K1 (99)
        F1-3:pH (73)
            F1-3-1:TRPV1-K2 (100)
        F1-4:CPS (74)
            F1-4-1:TRPV1-K3 (101)
        F1-5:AEA (75)
            F1-5-1:TRPV1-K4 (102)
    G1: PMCA (9)
        G1-1:Ca2+(IN) (78)
    J: PKA (36)
    K1: PKC (14)
    K2: CaMKII (15)
    K3: Calcineurin (16)
    K4: PLD (53)
    L: AEA (4)
        L-1:PLD(A) (81)
        L-2:FAAH (83)
        L-3:AMT (84)
    M: SNARE (13)
    N: NCX (54)
```

Fig. 5: Hierarchy graph of the entire nociceptive network.

Applications of Region Theory (ART)

Introduction

Regions have been defined about 20 years ago by Andrzej Ehrenfeucht and Grzegorz Rozenberg as sets of nodes of a finite transition system that correspond to potential conditions that enable or disable transition occurrences in a corresponding elementary net system. Thus, regions have been the essential concept for synthesis of elementary net systems from its "anonymous" state graph (states are unknown but transitions between states are known). Since that time, many generalizations and variants of the synthesis problem of Petri nets from behavioural descriptions have been studied, including synthesis of more general Petri net classes, synthesis from languages, synthesis from partially ordered runs and synthesis from incomplete behavioural descriptions. All this work has in common that the transition names are given more or less directly by the behavioural description. The places of the net to be synthesized always correspond to regions which are defined in many different ways, depending on the form of the behavioural description. A major issue in this research is the study of regions, whence we call the entire research direction *Region Theory.*

Region Theory was applied in many different areas such as:

- hardware synthesis from precise specifications (synthesis from transition systems)
- visualization of concurrent hardware behaviour (synthesis from logic circuit models, transition systems and partial orders)
- GALS synthesis and desynchronisation based on synthesis (synthesis from step transition systems and re-synthesis from Petri nets)
- synthesis of control and policies for discrete event systems (synthesis from both languages and transition systems)
- modelling biological (membrane) systems with localities (synthesis from step transition systems)
- generation of specifications from incomplete specifications (mining from transition systems)
- model generation from examples (specification from (partial) languages)
- mining of process descriptions (mining from languages)

The aim of the ART workshop series was to bring together people working in these or other application areas of region theory, to exchange ideas and concepts and to work on common workshop results.

This chapter contains reviewed contributions submitted to and presented at the 1st ART workshop in Braga, Portugal.

Jörg Desel (Hagen, Germany)
Alex Yakovlev (Newcastle University, UK)

Modeling and Mining of Collaborative Learnflows

Robin Bergenthum, Andreas Harrer, Sebastian Mauser

Katholische Universität Eichstätt-Ingolstadt, Fachgebiet Informatik
forename.name@ku-eichstaett.de

Abstract. This short paper first presents a modeling language for collaborative learnflows. This language is based on ideas from the area of business process modeling. Second, it is shown how to automatically generate respective learnflow models from log files of learning systems.

1 Introduction

Business processes have been established in the research and application field of *business process resp. workflow engineering* with matured methods, representations and computer support with tools. In contrast, the closely related learning and teaching processes only recently gained attention in research and practice and have not yet created shared terms, methods, and representations. This is especially true for the field of Computer-supported Collaborative Learning (CSCL), a discipline that investigates in the affordances and effects of computer applications supporting groups of students in knowledge construction and skill development. Thus, while we will call the formal representation of learning / teaching processes *learnflow engineering* in this paper, this term is neither firmly established nor fully deserves the engineering character, but is approaching this currently with the work of our colleagues and our own contributions. To reach this goal we compared in earlier work [1] commonalities, specifics, and potential methodological transfer between workflow and learnflow engineering. We also introduced first approaches for formal modeling of collaborative learning processes by means of Petri nets and the synthesis of nets from protocols and log files via process mining algorithms.

Of particular interest are the explicit representation of collaborative learning and of roles and learning groups. In comparison to business processes [2] the following specifics of learning processes have to be taken into account:

– For business processes the ultimate goal of performing / enacting a process with its associated activities is the achievement of a product with guaranteed quality criteria, while the participation of individual actors is only a minor concern. However, for learning processes the priority is that the *learners* involved in the activities get the opportunity to gain knowledge and experience: an objectively measurable result of the process is - besides the use of formative evaluations / exams - much less important than the completion of the learning experience for each participant and its implicit result of constructed knowledge in the student's mind. This requires that in the modeling of learning processes the individual actors have to be modeled thoroughly and explicitly.

- The concept of *role* in workflow engineering is mainly based on responsibility and ability for a set of activities, which is static after an initial assignment of roles to actors. Dynamic constraints on the activities (e.g. the same actor that created a proposal should also fix the contract) and special rules for allocation of actors to activities (e.g. the actor of a given role that has a minimum number of other roles should be chosen to distribute the workload) have been formulated in respective work by means of additional model constructs and notations. In contrary to this rigid role concept, the usage of roles in learning processes is frequently guided by exercising specific skills where roles are changed and acquired during a learning process. Therefore, an extension of static role concepts to dynamic ones that are capable of taking into account the learning history are needed for learnflow processes.
- Each activity, potentially even the whole learning process, may require *group work* and discussion and especially these group activities can have a high importance for the learning experience. Thus, the flexible and explicit representation of groups with required roles and - if needed - the dynamic re-arrangement of groups is one more requirement for learnflow engineering methods.

In this paper we will present a modeling language for learning processes that takes specifically into account the requirements identified above. To make use of expertise and experience from workflow engineering we will build the proposal upon sound existing approaches from this field [2]. Besides the intended applicability for learning and teaching processes we also consider our approach useful for business processes with dynamic roles, cooperative and collaborative activities such as in adaptive workflows where a static process structure is not satisfactory.

A first modeling approach in this direction has been proposed in [1]. There, we represented learning processes as Petri nets as is usual in the field of workflow management [2]. The allocation of actors was made by assignment of the required roles to activities and the usage of a global pool of actors that have the possible roles they can take associated to them. The established workflow concepts were extended by a mechanism to allow the change of actor roles while performing an activity.

Because of the significance of role changes in the modeling of learning processes in this paper we propose a refinement of that approach. We explicitly model dynamic role assignments and changes in a state diagram: actors can change their assigned roles when performing activities, which means that the role changes in the state diagram are synchronized with the activities in the process model (Petri net). Learning groups are taking into account by collaborative activities performed by several roles jointly.

We will discuss the potential and methods for the automated synthesis of these models from real protocol instances as an approach for *collaboration flow mining*. For this we extend our first proposal of a learnflow mining approach presented in [1]. This previous work focused on the discovery of the structures for the control flow using methods from the area of workflow mining [3, 4], but in this paper we will address additionally the challenge how to gain information about dynamic roles and collaboration situations from the protocol instances. Related work to this is organizational mining [5] that is limited to static roles and organizational units.

In Section 2 we will present our new modeling language by an example learning process. By means of this example we also illustrate the core ideas of collaboration flow mining in Section 3.

2 Modeling Language

As an example we consider the following computer supported learning scenario. Groups of three students use the tool Freestyler (www.collide.info) to learn the effect of different factors such as lightning conditions and CO2-concentration on the growth of plants. For this purpose, Freestyler provides several tabs with different functionalities. There are for instance tabs to formulate questions, to create simple models or to import data from a simulation tool. In this example, the set of tabs corresponds to the set of supported learning activities. Namely we consider the following activities: In = **In**troduction, Qu = Elaboration of the Research **Qu**estion, Pl = **Pl**anning, Mo = **Mo**deling of the Relations of the Different Factors, Hy = **Hy**pothesize, E1 & E2 = **E**xperiment One resp. Two for Hypothesis Testing, Da = Study of Existing Experimental **Da**ta for Hypothesis Testing, An = **An**alysis of Experimental Results Including Comprehensive Hypothesis Testing, Pr = **Pr**esentation of the Research Results. Some of these learning activities require collaboration (In, Hy, An require all three learners and Pl, Mo, Pr each require two learners).

In this context a learnflow model first describes the order in which the tabs have to be processed by the learners. Second, it determines who is allowed to work on a tab. This depends on the dynamic roles of the learners within the learning group. The learners may have the role Student, Modeler, ExModeler, Recorder and ExRecorder. Student is the default role. The other roles encode the learning history of the learners, e.g. Recorder and ExRecorder store the information that the learner has performed the activity Question.

Figure 1 together with Figure 2 show a possible learnflow model for the example scenario. Figure 1 illustrates the process aspect in the form of a Petri net with transition annotations. The annotations refer to the numbers of roles required for an activity. The state chart of Figure 2 complements the Petri net model. It represents a consistent role diagram, which models the dynamic roles of the three learners in the pool of actors. Each learner corresponds to one instance of the state chart. Instead of state charts, it would also be possible to consider state machine Petri nets to model the roles of learners, but state charts are in our opinion the more natural modelling approach.

The dynamic of the learnflow model is as follows. An activity of the net can only be accomplished, if the pool of actors contains learners having the roles annotated at the transition. For instance, the collaborative activity Pl requires two actors with the role Student. The occurrence of a transition can change the roles of the involved actors. Such change is modeled in the role diagram by a state transition with the activity name as the input symbol. Therefore, in our example the activity Pl causes the two students to switch over to the role Modeler. This role is later on required to perform the activity Mo. For simplicity of the role diagrams, we use the following convention. If an actor performs an activity, which does not explicitly cause a change of his role in the state chart, he preserves his role. For instance, the activity Mo does not change the role of an actor with the role Modeler and therefore can be neglected in the state chart.

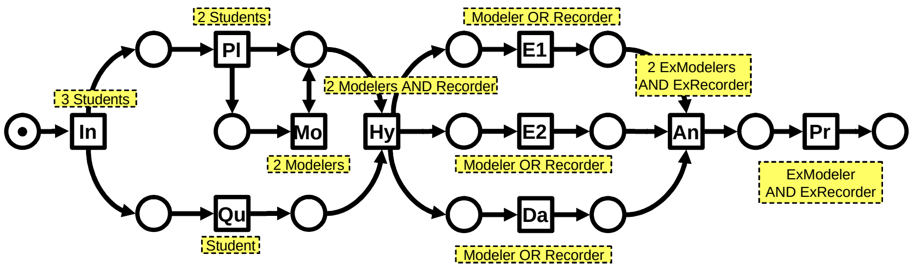


Fig. 1. Petri net model with role annotations

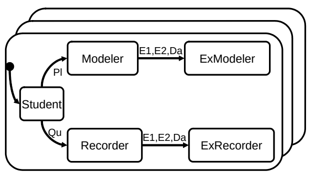


Fig. 2. State chart representing a role diagram

The example shows, that the presented modeling language allows to comprehensibly represent collaborative activities as well as dynamic roles and the learning progress of learners within a learning process. Still the language is simple and intuitive. It naturally extends well established modeling approaches from the domain of business process management. The approach supports a clear separation of the process perspective and the role perspective of a learn flow.

For a formal definition of the occurrence rule of the new modeling language, we consider a translation of respective learn flow models into a special class of colored Petri nets [6]. All the standard Petri net components are kept in the high-level Petri net model. The annotations of the transitions and the state diagrams (we require that a role occurs only once in a state diagram) are translated into one high-level place modelling the pool of ressources resp. actors. The color set of this place is given by the possible roles of learners. The initial marking contains for each state diagram one token of the kind given by the initial state of the state diagram. The place has an outgoing and an ingoing arc connected with each transition of the net. A transition consumes tokens from the place as given by its annotation. For each consumed token, there are two cases. Either the state corresponding to the token type in the state diagram allows a state transfer labeled with the name of the considered transition or it does not allow such transfer. In the first case, the transition produces a token of the kind given by the follower state of the state diagram in the ressource pool. In the second case, it produces a token of the same kind as the consumed token. Note that for classical business process models with static roles, an analogous translation is possible. But in this situation the first case neven occurs, i.e. each transition produces the same tokens in the ressource pool, that the transition consumes from the pool. This shows, that our new concept is more general than the classical approach.

Figure 3 illustrates the described translation for our example model (for the sake of clearness, the single ressource pool place is splitted in the illustration). The resulting high-level model on the one hand does not any more show explicitely the dynamic roles and the behavior of the learners. On the other hand it is quite difficult to read and understand. Therefore, for modeling purposes, the original representation should be prefered. The high-level view should only be used for formal considerations.

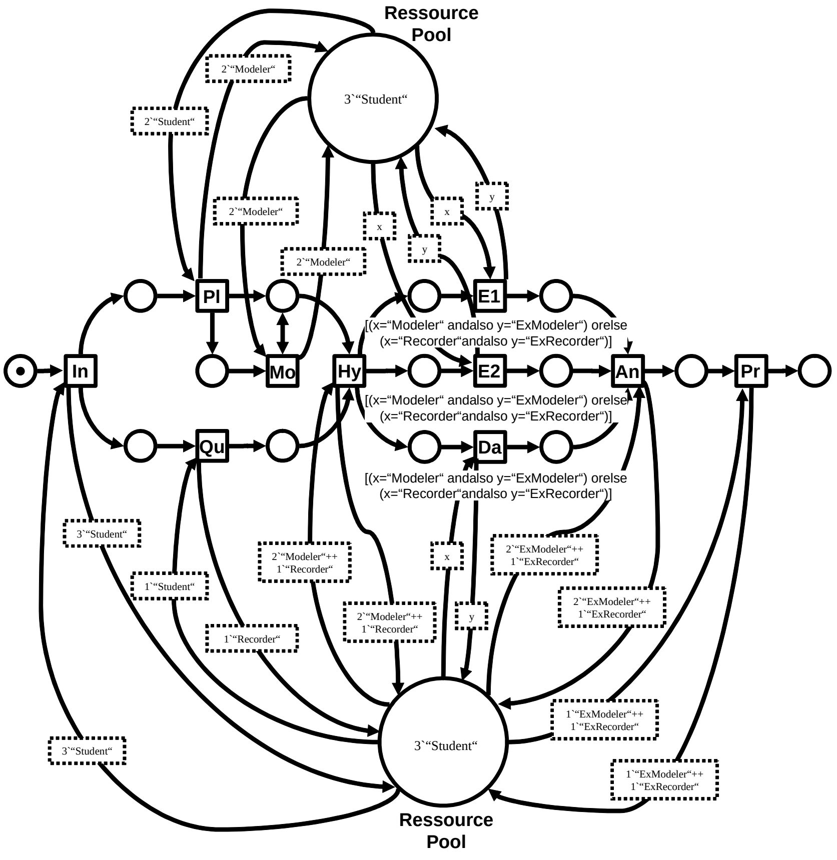

Ressource Pool
3`"Student"
2`"Modeler"
2`"Student"
2`"Modeler"
2`"Modeler"
x
y
x
y
In
Pl
Qu
Mo
Hy
E1
E2
Da
An
Pr
[(x="Modeler" andalso y="ExModeler") orelse (x="Recorder"andalso y="ExRecorder")]
[(x="Modeler" andalso y="ExModeler") orelse (x="Recorder"andalso y="ExRecorder")]
[(x="Modeler" andalso y="ExModeler") orelse (x="Recorder"andalso y="ExRecorder")]
3`"Student"
1`"Student"
1`"Recorder"
2`"Modeler"++ 1`"Recorder"
2`"Modeler"++ 1`"Recorder"
x
y
2`"ExModeler"++ 1`"ExRecorder"
2`"ExModeler"++ 1`"ExRecorder"
1`"ExModeler"++ 1`"ExRecorder"
1`"ExModeler"++ 1`"ExRecorder"
3`"Student"
3`"Student"
Ressource Pool

Fig. 3. High-level Petri net

There is the following important observation for our new modeling approach. The learning progress and the learning history of the actors are encoded in the role diagram. For instance, the activity E1 initiates a learning progress of the involved learner. This is represented by a change of the role of the learner. The new role then does not allow an execution of E2 and Da by this learner anymore. Thus, the three activities E1, E2 and Da have to be divided among the three learners. As an outlook, we also work on a further similar modeling language which avoids such progress-dependend role changes by applying a nets-in-nets concept.

Lastly, our modeling approach regards learning groups implicitly by means of collaborative activities. In view of an explicit modeling of groups, there are two natural and useful possibilities. First of all, different groups can simply be represented by different process instances. However, in this case an extension of the modeling language which allows to model dependencies of process instances would be helpful to regard dynamic group structures. Second, groups can also be modeled analogously as roles, i.e. the role diagrams of the actors can also capture information about group membership of actors. However, in contrast to roles it is important to represent the overall dynamic of the groups. This dynamic can only implicitly be regarded by group-memberships of single actors. Therefore, an extension of the modeling approach which at any time explicitly represents the learning groups would be interesting. This can for instance be achieved by a respective grouping of the state charts in the pool of actors.

3 Collaboration Flow Mining

In this section we introduce an approach to mine a learnflow model given a log file containing recorded learning activities. We construct a learnflow model which reflects a learning process performed by the students (maybe unknown to the teacher) or if the log file is filtered by the teacher even a desired learning process.

If an information system supports an actor while performing a learning activity this event can be recorded and gathered in protocol instances. Each recorded event contains information about the respective process instance, the name of the activity, the time of its occurrence and the involved actors. First, the events are ordered by their respective process and process instance. Second, they are ordered by the time of their occurrence within each process instance. Thus, each process instance yields a sequence of activities together with the respective actors. In the following we use these sequences as an input to a mining algorithm creating a process model. This process model can be used for verification and analysis issues or even as an input for information systems controlling the learnflow.

The tool Freestyler records the activities of the students. Figure 4 shows a part of a Freestyler log file of the considered learning process. It also shows a sequence of activities together with the corresponding actors resulting from that log file. The teacher may filter the log file by adding additional learning sequences or by removing unwanted learning sequences according to certain criteria such as subsequently measured learning achievements. In this case the mining yields a model of the desired learning process, while without filtering a model of the actual behaviour of the students is generated.

In the following we assume the log file to be complete for the given learning process, i.e. each possible learning sequence of the learning process is recorded in the log, where we distinguish learning sequences in the form shown in the last table of Figure 4. After abstracting from the set of actors in the learning sequences well know process mining algorithms [1, 3, 4] can be used to automatically construct a model of the control flow of the learning process. Since we consider a complete log file, we highly recommend to use a precise mining algorithm. Such algorithms exactly reproduce the sequences

Log file

Process	Process instance	Action	Student	Time
Photosynthesis	Group A	Introduction	Andi, Basti, Robin	10:03:12
Photosynthesis	Group A	Question	Robin	10:06:43
Photosynthesis	Group B	Introduction	Bert, Caro, Hans	10:07:33
...	...	...	...	...

Learning sequences

Group A	(Introduction; Andi,Basti,Robin), (Question; Robin), (Planning; Andi,Basti), (Modeling; Andi,Basti), (Hypothesis; Andi,Basti,Robin), (Experiment1; Andi), (Experiment2; Robin), (Data; Basti), (Analysis; Andi,Basti,Robin), (Presentation; Andi,Robin)
...	

Projection of learning sequences onto single students

Introduction, Planning, Modeling, Hypothesis, Experiment1, Analysis, Presentation
Introduction, Planning, Modeling, Hypothesis, Data, Analysis
Introduction, Question, Hypothesis, Experiment2, Analysis, Presentation
...

Learning sequences for role annotations

Group A	(Introduction; -,-,-), (Question; In), (Planning; In,In), (Modeling; InPl,InPl), (Hypothesis; InPlMo,InPlMo,InQu), (Experiment1; InPlMoHy), (Experiment2; InQuHy), (Data; InPlMoHy), (Analysis; InPlMoHyE1,InPlMoHyDa,InQuHyE2), (Presentation; InPlMoHyE1An,InQuHyE2An)
...	

Fig. 4. Example log file

of a log file if this is possible. Precise mining algorithms for Petri nets are base on the so called theory of regions. In [4] we propose to apply regions of languages for mining, since the sequences of a log file in a natural way determine a language. We have also implemented a respective algorithm. From the log file in Figure 3 this algorithm generates the Petri net model shown in Figure 1 yet without role annotations. In order to generate these role annotations and the state chart describing the dynamic roles of the students we introduce an additional mining method.

For every learning sequence and each occurring student within the learning sequence we consider the sequence of activities the student performs. Figure 4 shows this projection of the learning sequences onto the students for the considered example. All these sequences are integrated into a deterministic state chart in the form of a tree. Its states are determined by the history of previously performed activities (also regarding the order of the activities) and are named accordingly. In our example the resulting state chart is given in Figure 5 yielding a first model of roles for our learning process. In order to consistently use these roles as annotations in the Petri net model, in every learning sequence each actor must be renamed by the role describing the activities performed by the actor in the history of this learning sequence (see Figure 4 lower part). The role annotation of an activity in the Petri net is determined by all roles or combinations of roles in the case of a collaborative activity that occur together with this activity in any such learning sequence.

Although we have not formally proven this yet, it can be shown that using this approach and given a complete log file, i.e. a complete set of learning sequences, a model is calculated which has the same behaviour as show in the log file (if such a model exists). That means, the mined model and the log define the same learning sequences in the form depicted in the last table of Figure 4. In our example the generated model has equivalent behaviour to the learning process model shown in Figure 1. Yet the model has a much lager set of roles. The reason for this is that the role model encodes the complete history of each actor.

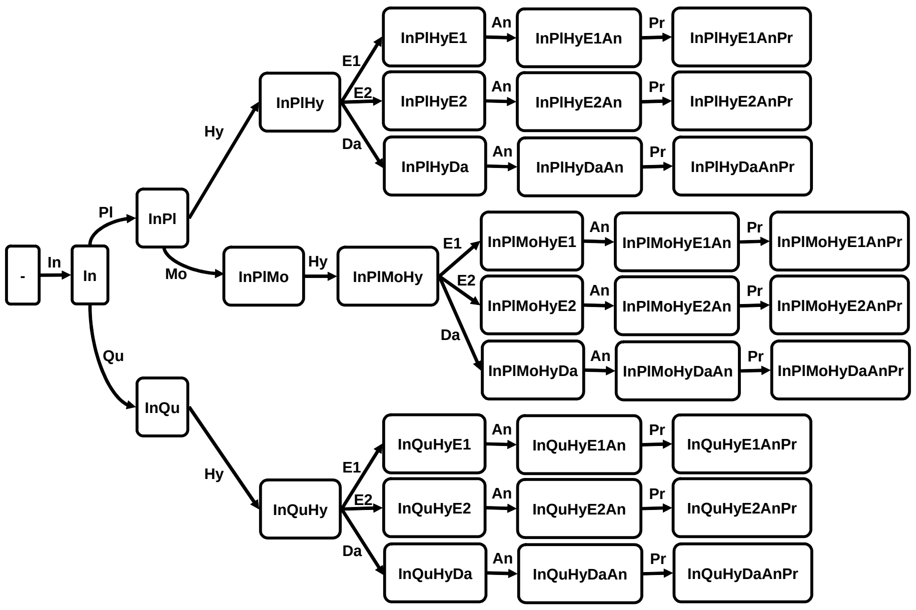


Fig. 5. Role diagram with finest granularity

In the following our goal is to simplify the model of roles by combining roles. Therefore, we developed the following rules. Remark that the annotations in the Petri net model have to be renamed consistently.

– Role transitions triggered by actions performed collaboratively by all actors belonging to one process instance can be neglected.
– Roles having the same postset of roles (respectively an empty postset) and the same outgoing activities can be fused, if for each outgoing activity the roles occur together with the same roles over all learning sequences. Thereby, outgoing activities between the roles to be fused can be neglected.
– Only as a last step roles having an empty postset can be neglected.

It can be proven, that given a complete log file, applying these rules to the previously mined role diagram again yields a behaviour equivalent learnflow model. In our example with these three rules we get a model of roles which is isomorphic to the model shown in Figure 2. Apart from the role names that are not present in the log files anyway we were in this case able to recreate the original learning process model from a complete log file. The role names have to be assigned by the teacher afterwards.

Finally, in practical settings typically we have to assume that not all possible learning sequences have been recorded in a log file. For such log files the presented approach has to be adapted. Heuristics can help to infer the missing sequences and to integrate them into the process model. For the control flow perspective existing methods from the area of process mining can be used in this context [3]. For the role diagrams we plan to use methods from the theories of structural equivalence and generalized block modelling [7] where missing information is penalized and a solution with minimal penalties can be used for the generalized role model.

References

1. Bergenthum, R., Desel, J., Harrer, A., Mauser, S.: Learnflow mining. In: DeLFI, LNI 132, GI (2008) 269–280
2. Aalst, W., Hee, K.: Workflow Management: Models, Methods, and Systems. MIT Press (2002)
3. Aalst, W.: Finding Structure in Unstructured Processes: The Case for Process Mining. In: ACSD 2007, IEEE (2007) 3–12
4. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process Mining Based on Regions of Languages. In: BPM 2007, Springer (2007) 375 – 383
5. Song, M., Aalst, W.: Towards comprehensive support for organizational mining. Decision Support Systems **46**(1) (2008) 300–317
6. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1-3 of Monographs in Theoretical Computer Science. Springer (1992, 1994, 1997)
7. Doreian, P., Batagelj, V., Ferligoj, A.: Generalized Blockmodeling. Volume 25 of Structural Analysis in the Social Sciences. Cambridge University Press, Cambridge (2005)

Synthesis of General Petri Nets with Localities

Maciej Koutny and Marta Pietkiewicz-Koutny

School of Computing Science
Newcastle University
Newcastle upon Tyne, NE1 7RU
United Kingdom
{maciej.koutny,marta.koutny}@newcastle.ac.uk

Abstract. There is a growing need to introduce and develop computational models capable of faithfully modelling systems whose behaviour combines synchrony with asynchrony in a variety of complicated ways. Examples of such real-life systems can be found from VLSI hardware GALS systems to systems of cells within which biochemical reactions happen in synchronised pulses. One way of capturing the resulting intricate behaviours is to use Petri nets with localities where transitions are partitioned into disjoint groups within which execution is synchronous and maximally concurrent. In this paper, we generalise this type of nets by allowing each transition to belong to several localities. Moreover, we define this extension in a generic way for all classes of nets defined by net-types. We show that Petri nets with overlapping localities are an instance of the general model of nets with policies. Thanks to this fact, it is possible to automatically construct nets with localities from behavioural specifications given in terms of finite step transition systems. After that we outline our initial ideas concerning net synthesis when the association of transition to localities is not given and has to be determined by the synthesis algorithm.
Keywords: theory of concurrency, Petri nets, localities, analysis and synthesis, step sequence semantics, conflict, theory of regions, transition systems.

1 Introduction

In the formal modelling of computational systems there is a growing need to faithfully capture real-life systems exhibiting behaviour which can be described as ‘globally asynchronous locally (maximally) synchronous’ (GALS). Examples can be found in hardware design, where a VLSI chip may contain multiple clocks responsible for synchronising different subsets of gates [6], and in biologically inspired membrane systems representing cells within which biochemical reactions happen in synchronised pulses [15]. To capture such systems in a formal manner, [9] introduced *Place/Transition-nets with localities* (PTL-nets), where each locality identifies a distinct set of transitions which must be executed synchronously, i.e., in a maximally concurrent manner (akin to *local maximal concurrency*). The expressiveness of PTL-nets (even after enhancing them with inhibitor and activator arcs in [8]) was constrained by the fact that each transition belonged to a unique locality, and so the localities were all *non-overlapping*. In this paper, we drop this restriction aiming at a net model which we believe should provide a greater scope for faithful (or direct) modelling features implied by the complex nature of, for example, modern VLSI systems or biological systems.

> To explain the basic idea behind nets with *overlapping localities*, let us consider an array of n transitions t_i $(0 \leq i \leq n-1)$ which are arranged in a circular manner, i.e., t_i is adjacent to $t_{(i+n-1) \,\mathsf{mod}\, n}$ and $t_{(i+1) \,\mathsf{mod}\, n}$ which form its 'neighbourhood'. Each of the transitions belongs to some subsystem which is left unspecified. What is important from our point of view is that to be executed, t_i needs, in addition to being enabled by its subsystem, to receive an external stimulus (e.g., an electric charge when transitions represent biological cells) which then spreads to its neighbourhood forcing the execution of transition $t_{(i+n-1) \,\mathsf{mod}\, n}$ and $t_{(i+1) \,\mathsf{mod}\, n}$ provided they are enabled by their subsystems. Thus, stimulating a transition amounts to stimulating its neighbourhood, and neighbourhoods can overlap which means that a given transition can be triggered in possibly many ways. To model such a scenario in a direct way we can use a Petri net augmented with a locality mapping ℓ such that $\ell(t_i) = \{(i+n-1) \,\mathsf{mod}\, n, i, (i+1) \,\mathsf{mod}\, n\}$, where each integer represents a distinct locality, and assuming that a transition may be executed only if it belongs to some stimulated neighbourhood. For example, if all the transitions t_i are enabled by their subsystems, then the following are examples of legal steps of the Petri net:
>
> | | |
> |---|---|
> | $\{t_2, t_3, t_4\}$ | t_3 stimulated |
> | $\{t_2, t_3, t_4, t_5\}$ | t_3 and t_4 stimulated |
> | $\{t_2, t_3, t_4, t_5, t_8, t_9, t_{10}\}$ | t_3, t_4 and t_9 stimulated |
>
> and two examples of illegal steps are $\{t_2, t_3\}$ and $\{t_2, t_3, t_4, t_6\}$.
> In the abstract capture of the underlying mechanisms like that above, we will demand that an executed transition belongs to at least one *saturated* locality, i.e., it is not possible to additionally execute any more transitions associated with that locality.

Rather than introducing nets with overlapping localities for PT-nets or their extensions, we will move straight to the general case of τ-nets [2] which encapsulate a majority of Petri net classes for which the synthesis problem has been investigated. In fact, the task of defining τ-nets with (potentially) overlapping localities is straightforward, as the resulting model of *τ-nets with localities* turns out to be an instance of the general framework of τ-nets with *policies* introduced in [4].

After introducing the new model of nets, we turn our attention to their automatic synthesis from behavioural specifications given in terms of step transition systems. Since τ-nets with localities are an instance of a more general scheme treated in [4], we directly import synthesis results presented there which are based on the regions of a transition system studied in other contexts, in particular, in [1–3, 7, 13, 14, 16, 10, 11].

The results in [4] assume that policies are given which, in our case, means that we know exactly the localities associated with all the net transitions. This may be difficult to guarantee in practice, and so in the second part of the paper we outline our initial ideas concerning net synthesis when this is not the case, extending our previous work on non-overlapping localities reported in [12].

2 Preliminaries

In this section, we recall some basic notions concerning τ-nets, policies and the synthesis problem as presented in [4].

An *abelian monoid* is a set S with a commutative and associative binary (composition) operation $+$ on S, and a neutral element $\mathbf{0}$. The monoid element resulting from composing n copies of $s \in S$ will be denoted by $n \cdot s$, and so $\mathbf{0} = 0 \cdot s$ and $s = 1 \cdot s$.

A specific abelian monoid, $\langle T \rangle$, is the free abelian monoid generated by a set (of transitions) T. It can be seen as the set of all the multisets over T. We will use $\alpha, \beta, \gamma, \ldots$ to range over the elements of $\langle T \rangle$. Moreover, for all $t \in T$ and $\alpha \in \langle T \rangle$, we will use $\alpha(t)$ to denote the multiplicity of t in α. We will write $t \in \alpha$ whenever $\alpha(t) > 0$, and denote by $supp(\alpha)$ the set of all $t \in \alpha$. The size of α is given by $|\alpha| = \sum_{t \in T} \alpha(t)$.

We denote $\alpha \leq \beta$ whenever $\alpha(t) \leq \beta(t)$ for all $t \in T$ (and $\alpha < \beta$ if $\alpha \leq \beta$ and $\alpha \neq \beta$). For $X \subseteq \langle T \rangle$, we denote by $\max_{\leq}(X)$ the set of all $\leq$-maximal elements of X, and by $\min_{\leq}(X)$ the set of all non-empty $\leq$-minimal elements of X.

If $T' \subseteq T$ then $\alpha|_{T'}$ is a multiset α' such that $\alpha'(t) = \alpha(t)$ if $t \in T'$ and otherwise $\alpha'(t) = 0$. The sum of two multisets, α and β, will be denoted by $\alpha + \beta$, and a singleton multiset $\{t\}$ simply by t.

A *transition system* over an abelian monoid S is a triple (Q, S, δ) such that Q is a set of *states*, and $\delta : Q \times S \to Q$ a partial *transition function*[1] satisfying $\delta(q, \mathbf{0}) = q$ for all $q \in Q$. An *initialised* transition system $\mathcal{T} \overset{\mathrm{df}}{=} (Q, S, \delta, q_0)$ has in addition an *initial* state $q_0 \in Q$ from which every other state is *reachable*. For every state q of a (non-initialised or initialised) transition system TS, $enbld_{TS}(q) \overset{\mathrm{df}}{=} \{s \in S \mid \delta(q, s) \text{ is defined}\}$.

Initialised transition systems $\mathcal{T}$ over free abelian monoids — called *step transition systems* — will represent concurrent behaviours of Petri nets. Non-initialised transition systems τ over arbitrary abelian monoids — called *net-types* — will provide ways to define various classes of nets. Throughout the paper, we will assume that:

– T is a <u>fixed</u> finite set (of net transitions);
– *Loc* is a <u>fixed</u> finite set (of net transitions' localities);

[1] Transition functions and net transitions are unrelated notions.

– $\mathcal{T} = (Q, S, \delta, q_0)$ is a <u>fixed</u> step transition system over $S = \langle T \rangle$.
– $\tau = (\mathbb{Q}, \mathbb{S}, \Delta)$ is a <u>fixed</u> net-type over an abelian monoid $\mathbb{S}$. In this paper, we will assume that τ is *substep closed* which means that, for every state $q \in Q$, if $\alpha + \beta \in enbld_\tau(q)$ then also $\alpha \in enbld_\tau(q)$. This will imply that substeps of resource enabled steps are also resource enabled which is a condition usually satisfied in practice.

The net-type defines a class of nets, by specifying the values (markings) that can be stored in net places ($\mathbb{Q}$), the operations and tests (inscriptions on the arcs) that a net transition may perform on these values ($\mathbb{S}$), and the enabling condition and the newly generated values for steps of transitions (Δ).

Definition 1 (τ-net). *A* τ-net system *is a tuple* $\mathcal{N} \overset{\text{df}}{=} (P, T, F, M_0)$*, where* P *and* T *are disjoint sets of places and transitions, respectively;* $F : (P \times T) \to \mathbb{S}$ *is a flow mapping; and* $M_0 : P \to \mathbb{Q}$ *is an* initial marking.

In general, any mapping $M : P \to \mathbb{Q}$ is a marking. For each place $p \in P$ and step $\alpha \in \langle T \rangle$, $F(p, \alpha) \overset{\text{df}}{=} \sum_{t \in T} \alpha(t) \cdot F(p, t)$.

Definition 2 (step semantics). *Given a* τ*-net system* $\mathcal{N} = (P, T, F, M_0)$*, a step* $\alpha \in \langle T \rangle$ *is* (resource) enabled *at a marking* M *if, for every place* $p \in P$*:*

$$F(p, \alpha) \in enbld_\tau(M(p)) \; .$$

We denote this by $\alpha \in enbld_\mathcal{N}(M)$*. The* firing *of such a step produces the marking* M' *such that, for every* $p \in P$*:*

$$M'(p) \overset{\text{df}}{=} \Delta(M(p), F(p, \alpha)) \; .$$

Step firing policies are means of controlling and constraining the huge number of execution paths resulting from the concurrent nature of a majority of computing systems.

Let $\mathcal{X}_\tau$ be the family of all sets of steps enabled at some reachable marking M of some τ-net $\mathcal{N}$ with the set of transitions T.

Definition 3 (bounded step firing policy). *A* bounded step firing policy *for* τ*-nets over* $\langle T \rangle$ *is given by a control disabled steps mapping* $cds : 2^{\langle T \rangle} \to 2^{\langle T \rangle \setminus \{\mathbf{0}\}}$ *such that, for all* $X \subseteq \langle T \rangle$*, the following hold:*

1. *If* X *is infinite then* $cds(X) = \varnothing$*.*
2. *If* X *is finite then, for every* $Y \subseteq X$*:*
   (a) $cds(X) \subseteq X$*;*
   (b) $cds(Y) \subseteq cds(X)$*; and*
   (c) $X \in \mathcal{X}_\tau$ *and* $X \setminus cds(X) \subseteq Y$ *imply* $cds(X) \cap Y \subseteq cds(Y)$*.*

We will now discuss further step firing policies and their effect on net behaviour.

Definition 4 (τ-net with policy). *Let cds be a bounded step firing policy for τ-nets over $\langle T\rangle$. A tuple $\mathcal{NP} \overset{\text{df}}{=} (P, T, F, M_0, cds)$ is a τ*-net system with policy *if $\mathcal{N} = (P, T, F, M_0)$ is a τ-net and the (control) enabled steps of $\mathcal{NP}$ at a marking M are:*

$$Enbld_{\mathcal{NP}}(M) \overset{\text{df}}{=} enbld_{\mathcal{N}}(M) \setminus cds(enbld_{\mathcal{N}}(M)).$$

Moreover, let $enbld_{\mathcal{NP}}(M) \overset{\text{df}}{=} enbld_{\mathcal{N}}(M)$ be the set of resource enabled steps of $\mathcal{NP}$ at marking M. The effect of executions of enabled steps in $\mathcal{NP}$ is the same as in $\mathcal{N}$.

We will denote by $CRG(\mathcal{NP})$ the step transition system with the initial state M_0 formed by firing inductively from M_0 all possible control enabled steps of $\mathcal{NP}$, and call it concurrent reachability graph *of $\mathcal{NP}$.*

In this paper our concern will be to find a general solution to the synthesis problem for τ-nets with localities. Since they are special kinds of τ-nets with policies we will be able to use the theory developed for those nets in [4]. By solving a synthesis problem we mean finding a procedure for building a net of a certain class with the desired behaviour (in our case, concurrent reachability graph). In our case the problem can be defined as follows.

SYNTHESIS PROBLEM

Let $\mathcal{T}$ be a given finite step transition system. Provide necessary and sufficient conditions for $\mathcal{T}$ to be *realised* by some τ-net system with policy $\mathcal{NP}$ (i.e., $\mathcal{T} \cong CRG(\mathcal{NP})$ where $\cong$ is transition system isomorphism preserving the initial states and transition labels).

The solution of the synthesis problem is based on the idea of a region of a transition system.

Definition 5 (τ-region). *A τ-*region *of $\mathcal{T}$ is a pair of mappings*

$$(\sigma : Q \to \mathbb{Q} \ , \ \eta : \langle T\rangle \to \mathbb{S})$$

such that η is a morphism of monoids and, for all $q \in Q$ and $\alpha \in enbld_{\mathcal{T}}(q)$:

$$\eta(\alpha) \in enbld_{\tau}(\sigma(q)) \quad and \quad \Delta(\sigma(q), \eta(\alpha)) = \sigma(\delta(q, \alpha)) \ .$$

For every state q of Q, we denote by $enbld_{\mathcal{T},\tau}(q)$ the set of all steps α such that $\eta(\alpha) \in enbld_{\tau}(\sigma(q))$, for all τ-regions (σ, η) of $\mathcal{T}$.

We then have the following general result from [4].

Theorem 1. *$\mathcal{T}$ can be realised by a τ-net system with a (bounded step firing) policy cds* iff *the following two regional axioms are satisfied:*

AXIOM I: STATE SEPARATION

For any pair of states $q \neq r$ of $\mathcal{T}$, there is a τ-region (σ, η) of $\mathcal{T}$ such that $\sigma(q) \neq \sigma(r)$.

AXIOM II: FORWARD CLOSURE WITH POLICIES

For every state q of $\mathcal{T}$, $enbld_{\mathcal{T}}(q) = enbld_{\mathcal{T},\tau}(q) \setminus cds(enbld_{\mathcal{T},\tau}(q))$. □

A solution to the synthesis problem is obtained if one can compute a finite set $\mathcal{WR}$ of τ-regions of $\mathcal{T}$ *witnessing* the satisfaction of all instances of AXIOMS I and II [5]. A suitable τ-net system with policy *cds*, $\mathcal{NP}_{\mathcal{WR}} = (P, T, F, M_0, cds)$, can be then constructed with $P = \mathcal{WR}$ and, for any place $p = (\sigma, \eta)$ in P and every $t \in T$, $F(p,t) = \eta(t)$ and $M_0(p) = \sigma(q_0)$ (recall that q_0 is the initial state of $\mathcal{T}$, and $T \subseteq \langle T \rangle$).

3 τ-nets with localities

We will now introduce a general class of Petri nets with localities, based on a specific class of control disabled steps mappings.

A locality mapping for the transition set T is any $\ell : T \to 2^{Loc}$ such that $\ell(t) \neq \varnothing$ for all $t \in T$. (Below we will denote $l \in \ell(\alpha)$, for every step α and a locality $l \in Loc$, whenever there is a transition $t \in \alpha$ such that $l \in \ell(t)$.) Then the induced control disabled steps mapping is

$$cds_\ell : 2^{\langle T \rangle} \to 2^{\langle T \rangle \setminus \{\mathbf{0}\}}$$

such that, for all $X \subseteq \langle T \rangle$:

$$cds_\ell(X) \overset{\text{df}}{=} \begin{cases} \{\alpha \in X \mid \exists t \in \alpha\ \forall l \in \ell(t)\ \exists \alpha + \beta \in X :\ l \in \ell(\beta)\} & \text{if } X \text{ is finite} \\ \varnothing & \text{otherwise .} \end{cases}$$

Proposition 1. *cds_ℓ is a bounded step firing policy.*

Proof. All we need to prove is that if $X \in \mathcal{X}_\tau$ is finite and $Y \subseteq X$ and $X \setminus cds_\ell(X) \subseteq Y$ and $\alpha \in cds_\ell(X) \cap Y$, then $\alpha \in cds_\ell(Y)$.

We first observe that $\max_\leq(X) \cap cds_\ell(X) = \varnothing$ and so $\max_\leq(X) \subseteq X \setminus cds_\ell(X) \subseteq Y$. Then we observe that since X is finite and $\alpha \in cds_\ell(X)$, there is $t \in \alpha$ such that for all $l \in \ell(t)$ there exists $\alpha + \beta \in \max_\leq(X) \subseteq Y$ satisfying $l \in \ell(\beta)$. This and the fact that Y is finite (as $Y \subseteq X$) means that $\alpha \in cds_\ell(Y)$. □

We will call a τ-net system with a policy cds_ℓ a *τ/ℓ-net system* (or τ-net with localities). Moreover, we will call $\mathcal{T}$ a *τ/ℓ-transition system* if AXIOM I and AXIOM II are satisfied for $\mathcal{T}$ with policy $cds = cds_\ell$.

Proposition 2. *Let M be a marking of a τ/ℓ-net system $\mathcal{NP}$ such that the set $enbld_{\mathcal{NP}}(M)$ is finite. A step $\alpha \in enbld_{\mathcal{NP}}(M)$ belongs to $Enbld_{\mathcal{NP}}(M)$* iff *for every $t \in \alpha$ there is $l \in \ell(t)$ such that:*

$$l \in \ell(t') \implies \alpha + t' \notin enbld_{\mathcal{NP}}(M) ,$$

for every transition t'.

Proof. Follows from $\max_\leq(enbld_{\mathcal{NP}}(M)) \subseteq Enbld_{\mathcal{NP}}(M)$, and the fact that $\gamma \leq \delta$ and $\delta \in enbld_{\mathcal{NP}}(M)$ together imply $\gamma \in enbld_{\mathcal{NP}}(M)$. □

We obtain an immediate solution to the synthesis problem for τ/ℓ-nets.

Theorem 2. *A finite step transition system $\mathcal{T}$ can be realised by a τ/ℓ-net system* iff *$\mathcal{T}$ is a τ/ℓ-transition system.*

Proof. Follows from Theorem 1 and Proposition 1. □

The synthesis problem for PT-nets and EN-systems with localities (and with or without inhibitor and read arcs) have been investigated in [10–12]. For such nets, the locality mapping ℓ has the property that $|\ell(t)| = 1$, for all $t \in T$. Such an ℓ defines localities which are mutually disjoint or *non-overlapping.* In this paper, we allow fully general, i.e., possibly overlapping localities.

As to the effective construction of synthesised net, it has been demonstrated in [10–12] that this can be easily done for net classes with non-overlapping localities mentioned above. Similar argument can be applied also in the general setting of overlapping localities and τ-nets corresponding to PT-nets and EN-systems with localities. We omit details.

Finally, it is interesting to observe that in the (previously considered) case of non-overlapping localities, cds_ℓ can be defined through a pre-order on steps. This is no longer the case for the general locality mappings.

4 Towards synthesis with unknown localities

The synthesis result presented in the previous section was obtained assuming that the locality mapping was given. However, in practice such a mapping might be unknown (or partially known), and part of the outcome of a successful synthesis procedure would be a suitable (or *good*) locality mapping. Clearly, what really matters in a locality mapping is the identification of (possibly overlapping) clusters of transitions, each cluster containing all transitions sharing a locality. Since there are only finitely many clusters, there are also finitely many non-equivalent locality mappings, and the synthesis procedure could simply enumerate them and then check one-by-one using Theorem 2. This, however, would be highly impractical as the number of clusters is exponential in the number of transitions. We will now present some initial ideas and results aimed at reducing the number of checks.

From now on we will assume that $\mathcal{T}$ is *finite*. We will also assume that we have checked that, for every state q of $\mathcal{T}$, the set of steps $\mathit{enbld}_{\mathcal{T},\tau}(q)$ is finite; otherwise $\mathcal{T}$ could not be isomorphic to the concurrent reachability graph of any τ-net with localities (see AXIOM II and Theorem 2).

In the rest of this section, for every state q of $\mathcal{T}$ and locality mappings ℓ, ℓ':

- $\mathit{allSteps}_q$ is the set of all steps labelling arcs outgoing from q.
- $\mathit{minSteps}_q$ is the set of all non-empty steps $\alpha \in \mathit{allSteps}_q$ for which there is no non-empty $\beta \in \mathit{allSteps}_q$ such that $\beta < \alpha$.
- T_q is the set of all net transitions occurring in the steps of $\mathit{allSteps}_q$.
- $\mathit{clusters}_q^\ell$ is the set of all sets $\{t \in T_q \mid l \in \ell(t)\}$, for every $l \in \ell(T_q)$.

– ℓ and ℓ' are node-consistent if $clusters_r^{\ell} = clusters_r^{\ell'}$, for every state r of $\mathcal{T}$.

A general result concerning locality mappings is that they are equally suitable for being good locality mapping whenever they induce the same clusters of co-located transitions in each individual node of the step transition system.

Proposition 3. *Let ℓ and ℓ' be two node-consistent locality mappings. Then $\mathcal{T}$ is τ/ℓ-transition system* iff *$\mathcal{T}$ is τ/ℓ'-transition system.*

Proof. Suppose that $\mathcal{T}$ is τ/ℓ-transition system. First we notice that AXIOM I does not depend on the locality mapping. For AXIOM II and ℓ' it suffices to show that, for each state q of $\mathcal{T}$:

$$cds_{\ell}(enbld_{\mathcal{T},\tau}(q)) = cds_{\ell'}(enbld_{\mathcal{T},\tau}(q)) \ . \tag{1}$$

We observe that the steps from $enbld_{\mathcal{T},\tau}(q)$ have transitions belonging to T_q (as the maximal steps in $enbld_{\mathcal{T},\tau}(q)$ never belong to $cds_{\ell}(enbld_{\mathcal{T},\tau}(q))$ and AXIOM II holds for ℓ), and thus according to the definition of $cds_{\ell}(X)$ the influence of each locality $l \in \ell(T_q)$ and $l' \in \ell'(T_q)$ can be accurately represented by the clusters $\{t \in T_q \mid l \in \ell(t)\}$ and $\{t' \in T_q \mid l' \in \ell'(t')\}$, respectively. Hence, since ℓ and ℓ' are node-consistent, (1) holds. □

As a consequence, a good locality mapping can be arbitrarily modified to yield another good locality mapping as long as the two mappings are node-consistent (there is no need to re-check the two axioms involved in Theorem 2). This should allow one to search for an optimal good locality mapping starting from some initial choice (for example, one might prefer to have as few localities per transition as possible).

The construction of a good locality mapping could be seen as modular process, in the following way. First, separately for each state q, we produce a list of possible cluster-sets of transitions in T_q induced by hypothetical good locality mappings. Each such *cluster-set clSet* $\overset{\text{df}}{=} \{C_1, \ldots, C_k\}$ is composed of non-empty subsets of T_q so that $C_1 \cup \ldots \cup C_k = T_q$ and:

$$enbld_{\mathcal{T}}(q) = enbld_{\mathcal{T},\tau}(q) \setminus cds_{clSet}(enbld_{\mathcal{T},\tau}(q))$$

where

$$cds_{clSet}(X) \overset{\text{df}}{=} \{\alpha \in X \mid \exists t \in \alpha \ \forall i \leq k : \ (t \in C_i \ \Rightarrow \ \exists t' \in C_i : \ \alpha + t' \in X)\} \ .$$

Similarly, one may produce, for each state q, a characterisation of inadmissible clustering of transitions. We can then select different cluster-sets (one per each state of the step transition system) and check whether combining them together yields a good locality mapping. Such a procedure was used in [12] to construct 'canonical' locality mappings for the case of non-overlapping localities (and the combining of cluster-sets was based on the operation of transitive closure).

The search for a good locality mapping outlined above can be improved if one looks for solutions in a specific class of nets, or if the locality mapping is partially known or constrained (for example, that two specific transitions cannot share a locality).

4.1 Localised conflicts

Intuitively, localities and conflicts may have opposite effects on step enabledness. Whereas joining two localities may reduce the number of control enabled steps, adding a conflict between transitions with shared localities may turn a non-enabled step into a control enabled one. It is therefore interesting what simplifications, if any, one might obtain if conflicts were constrained to exist between transitions sharing localities.

In the paper [12] we looked at this issue in the context of PTL-nets and ENL-nets, coming up with the notion of nets with *localised conflicts*. For the synthesis problem for such nets, it turned out that for each state q there was at most one cluster-set to be considered, providing particularly pleasant simplification of the original problem. In the rest of this section, we provide some initial results towards extending this to the case of nets with overlapping localities. Below, for a step α and locality l we denote $\alpha|_l \overset{\text{df}}{=} \alpha|_{\{t\in T \mid l\in\ell(t)\}}$ assuming that ℓ is given. Moreover, $\mathit{Enbld}^{min}_{\mathcal{NP}}(M) \overset{\text{df}}{=} \min_{\leq}\{\alpha \in \mathit{Enbld}_{\mathcal{NP}}(M) \mid \alpha \neq \varnothing\}$.

To start with, the set of *saturated localities* of a step α which is resource enabled at some marking M of a τ/ℓ-net $\mathcal{NP}$ is defined as:

$$\mathit{satlocalities}_M(\alpha) \overset{\text{df}}{=} \{l \in \ell(\alpha) \mid \neg\, \exists\, \alpha+\beta \in \mathit{enbld}_{\mathcal{NP}}(M) :\ l \in \ell(\beta)\}\ .$$

Intuitively, saturated localities are those which have been 'active' during the execution of a step α. It is immediate to see that if $\alpha \in \mathit{Enbld}_{\mathcal{NP}}(M)$ then, for all $t \in \alpha$:

$$\mathit{satlocalities}_M(\alpha) \cap \ell(t) \neq \varnothing\ .$$

Moreover, the above intersection may contain more than one active localities which are 'responsible' for the execution of transition t. At the level of potential clusters of a step transition system (i.e., groups of transitions which share a locality), we can define *saturated clusters* in a state q as:

$$\mathit{satclusters}_q(\alpha) \overset{\text{df}}{=} \{C \subseteq T_q \mid \alpha|_C \neq \varnothing\ \wedge\ \forall\, \alpha+\beta \in \mathit{allSteps}_q :\ \beta|_C = \varnothing\}\ .$$

Note that if $\mathcal{T}$ is the concurrent reachability graph of a τ/ℓ-net $\mathcal{NP}$, and M is a reachable marking of $\mathcal{NP}$, then:

$$\{\ \{t \in T_M \mid l \in \ell(t)\}\ \mid\ l \in \mathit{satlocalities}_M(\alpha)\ \} \subseteq \mathit{satclusters}_M(\alpha)\ .$$

The following definition is our first attempt to generalise the notion of nets with localised conflicts investigated in [12].

Definition 6 (localised conflicts). *A τ/ℓ-net system $\mathcal{NP}$ has* partially localised conflicts *if for all reachable markings M and non-empty steps α belonging to $\mathit{enbld}_{\mathcal{NP}}(M)$,*

$$t \in \mathit{enbld}_{\mathcal{NP}}(M) \quad \text{and} \quad \alpha + t \notin \mathit{enbld}_{\mathcal{NP}}(M)$$

implies $\mathit{satlocalities}_M(\alpha) \neq \varnothing$ and

$$\forall\, l \in \ell(t) \cap \mathit{satlocalities}_M(\alpha)\ :\ \alpha|_l + t \notin \mathit{enbld}_{\mathcal{NP}}(M)\ .$$

Intuitively, if there is a (global) conflict between a transition and a step, then this conflict can also be witnessed locally. We will now be concerned with the synthesis problem aimed at constructing τ/ℓ-net systems with partially localised conflicts.

Proposition 4. *Let $\mathcal{NP}$ be a τ/ℓ-net system with partially localised conflicts and M be its reachable marking.*
If $\alpha \in \mathit{Enbld}_{\mathcal{NP}}(M)$ then $\alpha|_l \in \mathit{Enbld}_{\mathcal{NP}}(M)$, for all $l \in \mathit{satlocalities}_M(\alpha)$.

Proof. Suppose that $\widetilde{l} \in \mathit{satlocalities}_M(\alpha)$ and $\alpha|_{\widetilde{l}} \notin \mathit{Enbld}_{\mathcal{NP}}(M)$.

Since $\alpha|_{\widetilde{l}} \leq \alpha \in \mathit{Enbld}_{\mathcal{NP}}(M)$, we have $\alpha|_{\widetilde{l}} \in \mathit{enbld}_{\mathcal{NP}}(M)$. Hence there is $\widetilde{t} \in \alpha|_{\widetilde{l}}$ such that, for all $l \in \ell(\widetilde{t})$, there is $\alpha|_{\widetilde{l}} + t' \in \mathit{enbld}_{\mathcal{NP}}(M)$ with $l \in \ell(t')$. In particular, since $\widetilde{l} \in \ell(\widetilde{t})$, there is $\alpha|_{\widetilde{l}} + \widehat{t} \in \mathit{enbld}_{\mathcal{NP}}(M)$ such that $\widetilde{l} \in \ell(\widehat{t})$. Hence $\widehat{t} \in \mathit{enbld}_{\mathcal{NP}}(M)$, and so we can use Definition 6 to infer that $\alpha + \widehat{t} \in \mathit{enbld}_{\mathcal{NP}}(M)$, producing a contradiction with $\widetilde{l} \in \mathit{satlocalities}_M(\alpha)$. □

Thus in terms of selecting clusters in the construction outlined in the previous section, if C has been selected at a state q then, for every α such that $C \in \mathit{satclusters}_q(\alpha)$, it must be the case that $\alpha|_C \in \mathit{allSteps}_q$.

Proposition 5. *Let $\mathcal{NP}$ be a τ/ℓ-net system with partially localised conflicts and M be its reachable marking.*
If $\alpha \in \mathit{Enbld}^{\mathit{min}}_{\mathcal{NP}}(M)$ then $\alpha = \alpha|_l$, for all $l \in \mathit{satlocalities}_M(\alpha)$.

Proof. By Proposition 4, we have $\alpha|_l \in \mathit{Enbld}_{\mathcal{NP}}(M)$, and by definition of $\alpha|_l$, we have that $\alpha|_l \leq \alpha$. Moreover, $\alpha|_l \leq \alpha$ and $\alpha|_l \neq \varnothing$ (as $l \in \ell(\alpha)$). Hence, as α is a minimal non-empty step in $\mathit{Enbld}_{\mathcal{NP}}(M)$, we have $\alpha = \alpha|_l$. □

Thus in terms of selecting clusters in the construction outlined in the previous section, if C has been selected at a state q then, for every $\alpha \in \mathit{minSteps}_q$ such that $C \in \mathit{satclusters}_q(\alpha)$, it must be the case that $\alpha|_C = \alpha$.

We will now present a series of results which can all be useful in the selection of clusters in the construction outlined in the previous section.

Proposition 6. *Let $\mathcal{NP}$ be a τ/ℓ-net system with partially localised conflicts and M be its reachable marking. Then, for all $\alpha \in \mathit{Enbld}^{\mathit{min}}_{\mathcal{NP}}(M)$:*

$$\mathit{satlocalities}_M(\alpha) \subseteq \bigcap\{\ell(t) \mid t \in \alpha\} \ .$$

Proof. Let $l \in \mathit{satlocalities}_M(\alpha)$. By Proposition 5, we have $\alpha = \alpha|_l$. Hence $l \in \ell(t)$, for all $t \in \alpha$. □

Corollary 1. *Let $\mathcal{NP}$ be a τ/ℓ-net system with partially localised conflicts and M be its reachable marking. Then, for all $\alpha \in \mathit{Enbld}^{\mathit{min}}_{\mathcal{NP}}(M)$:*

$$\bigcap\{\ell(t) \mid t \in \alpha\} \neq \varnothing \ .$$

Proof. By $\alpha \in \mathit{Enbld}_{\mathcal{NP}}(M)$, $\mathit{satlocalities}_M(\alpha) \cap \ell(t) \neq \varnothing$, for every $t \in \alpha$. Hence $\mathit{satlocalities}_M(\alpha) \neq \varnothing$ and the result follows from Proposition 6. □

We will now need the following auxiliary fact.

Proposition 7. *Let $\mathcal{NP}$ be a τ/ℓ-net system with partially localised conflicts and M be its reachable marking. If $\alpha, \beta \in \mathit{Enbld}_{\mathcal{NP}}(M)$ and $\alpha \leq \beta$ then $\mathit{satlocalities}_M(\alpha) \subseteq \mathit{satlocalities}_M(\beta)$.*

Proof. Suppose $l \in \mathit{satlocalities}_M(\alpha) \setminus \mathit{satlocalities}_M(\beta)$.
From $l \in \mathit{satlocalities}_M(\alpha)$ we have that $l \in \ell(\alpha)$ and:

$$\forall t : \; l \in \ell(t) \implies \alpha + t \notin \mathit{enbld}_{\mathcal{NP}}(M) \; . \tag{2}$$

From $l \notin \mathit{satlocalities}_M(\beta)$ we have that either $l \notin \ell(\beta)$, or $l \in \ell(\beta)$ and there is $\widetilde{t}$ such that:

$$l \in \ell(\widetilde{t}) \quad \wedge \quad \beta + \widetilde{t} \in \mathit{enbld}_{\mathcal{NP}}(M) \; . \tag{3}$$

Only the latter is possible, because $l \in \ell(\alpha)$ and $\alpha \leq \beta$. From (2) and (3) we have that $\alpha + \widetilde{t} \notin \mathit{enbld}_{\mathcal{NP}}(M)$ and $\beta + \widetilde{t} \in \mathit{enbld}_{\mathcal{NP}}(M)$, which produces a contradiction with $\alpha + \widetilde{t} \leq \beta + \widetilde{t}$. □

Proposition 8. *Let $\mathcal{NP}$ be a τ/ℓ-net system with partially localised conflicts and M be its reachable marking. Moreover, let $\alpha \in \mathit{Enbld}_{\mathcal{NP}}(M)$ and $\widetilde{l} \in \mathit{satlocalities}_M(\alpha)$ be such that $\alpha|_l \not< \alpha|_{\widetilde{l}}$, for all $l \in \mathit{satlocalities}_M(\alpha) \setminus \{\widetilde{l}\}$. Then $\alpha|_{\widetilde{l}} \in \mathit{Enbld}^{\mathit{min}}_{\mathcal{NP}}(M)$.*

Proof. From Proposition 4 we have that $\alpha|_{\widetilde{l}} \in \mathit{Enbld}_{\mathcal{NP}}(M)$. Suppose there is a non-empty step $\beta \in \mathit{Enbld}_{\mathcal{NP}}(M)$ such that $\beta < \alpha|_{\widetilde{l}}$. From Proposition 7 and $\alpha|_{\widetilde{l}} \leq \alpha$, it follows that

$$\mathit{satlocalities}_M(\beta) \subseteq \mathit{satlocalities}_M(\alpha|_{\widetilde{l}}) \subseteq \mathit{satlocalities}_M(\alpha) \; .$$

As $\beta \neq \varnothing$ and $\beta \in \mathit{Enbld}_{\mathcal{NP}}(M)$ we obtain

$$\mathit{satlocalities}_M(\beta) \neq \varnothing \; .$$

We have that $\widetilde{l} \notin \mathit{satlocalities}_M(\beta)$ as $\beta < \alpha|_{\widetilde{l}}$. Let $\widehat{l} \in \mathit{satlocalities}_M(\beta)$. Since $\mathit{satlocalities}_M(\beta) \subseteq \mathit{satlocalities}_M(\alpha)$, we have $\widehat{l} \in \mathit{satlocalities}_M(\alpha)$. Hence, by Proposition 4, $\alpha|_{\widehat{l}} \in \mathit{Enbld}_{\mathcal{NP}}(M)$. By the assumption we made, $\alpha|_{\widehat{l}} \not< \alpha|_{\widetilde{l}}$ as $\widehat{l} \neq \widetilde{l}$. So, there is $\widehat{t}$ such that $\alpha|_{\widehat{l}}(\widehat{t}) \geq \alpha|_{\widetilde{l}}(\widehat{t}) > \beta(\widehat{t})$, producing a contradiction with $\widehat{l} \in \mathit{satlocalities}_M(\beta)$. □

Let $\mathcal{NP}$ be a τ/ℓ-net system with partially localised conflicts and M be its reachable marking. Then:

$$\mathit{maxind}^M_t \overset{\mathrm{df}}{=} \max\{\alpha(t) \mid \alpha \in \mathit{Enbld}_{\mathcal{NP}}(M)\} \; ,$$

for every net transition t which is resource enabled at M.

Proposition 9. *Let $\mathcal{NP}$ be a τ/ℓ-net system with partially localised conflicts and M be its reachable marking. Moreover, let t and u be distinct transitions which are resource enabled at M and share a locality $\widetilde{l}$. Then exactly one of the following holds:*

- *There is no step $\alpha \in Enbld_{\mathcal{NP}}(M)$ such that $\widetilde{l} \in satlocalities_M(\alpha)$ and*
$$maxind_t^M + maxind_u^M = \alpha(t) + \alpha(u) \tag{4}$$
and, for all $l \in satlocalities_M(\alpha) \setminus \{\widetilde{l}\}$, we have $\alpha|_l \not< \alpha|_{\widetilde{l}}$.
- *There is $\alpha \in Enbld^{min}_{\mathcal{NP}}(M)$ such that $t, u \in \alpha$.*

Proof. We have $\widetilde{l} \in \ell(t) \cap \ell(u)$. Suppose that there is $\alpha \in Enbld_{\mathcal{NP}}(M)$ such that (4) holds and $\widetilde{l} \in satlocalities_M(\alpha)$ and for all $l \in satlocalities_M(\alpha) \setminus \{\widetilde{l}\}$, $\alpha|_l \not< \alpha|_{\widetilde{l}}$. Since t and u are resource enabled at M and (4) holds, we have $\alpha(t) \geq 1$ and $\alpha(u) \geq 1$. On the other hand, $\widetilde{l} \in \ell(t) \cap \ell(u)$. Then, $t, u \in \alpha|_{\widetilde{l}}$. We can see that all the conditions of Proposition 8 are satisfied for α and $\widetilde{l}$, and so we obtain that $\alpha|_{\widetilde{l}} \in Enbld^{min}_{\mathcal{NP}}(M)$. □

Unique and minimal step covers It is not possible to reverse the inclusion in Proposition 6, i.e., there can be $\alpha \in Enbld^{min}_{\mathcal{NP}}(M)$ such that:

$$\bigcap\{\ell(t) \mid t \in \alpha\} \subseteq satlocalities_M(\alpha)$$

does no hold. As an example, we can take a PT-net with two concurrent transitions, a and b, each having one pre-place marked with a single token and no post-places, satisfying $\ell(a) = \{l, l'\}$ and $\ell(b) = \{l\}$. Then the step $\alpha = \{a\}$ belongs to $Enbld^{min}_{\mathcal{NP}}(M_0)$ yet:

$$\bigcap\{\ell(t) \mid t \in \alpha\} = \{l, l'\} \ \not\subseteq\ \{l'\} = satlocalities_{M_0}(\alpha) .$$

Looking at the last example, one can make a comment about the advantages of allowing a single transition to have more than one locality. In such a situation, different localities can define different modes of engagement/co-operation. For transition a, the locality l could be interpreted as defining a 'co-operative mode', while l' a 'self-sufficient' mode. In this way, some localities may force big sets of transitions to work in synchrony, while other localities may allow smaller sets to be synchronised, or even single transitions to be executed alone. Intuitively, we can model different 'circles of co-operations' for net transitions.

If we take again the last two transitions, and this time consider the step $\{a, b\}$, then one may observe that there is certain ambiguity as to which localities have been active during its execution, as both $L = \{l\}$ and $L' = \{l, l'\}$ could be taken. We will now investigate the role of such sets of localities.

Definition 7 (step covers). *A* (locality) cover *of a step α is a set of localities L such that:*

$$supp(\alpha) = \bigcup\{supp(\alpha|_l) \mid l \in L\} ,$$

and it is minimal *if no proper subset of L is a locality cover of α. Moreover, a minimal locality cover L is* unique *if there is no other minimal locality cover L' for α such that $\{\alpha|_l \mid l \in L\} = \{\alpha|_{l'} \mid l' \in L'\}$.*

Example 1. Let $\ell(a) = \{l, l'\}$ and $\ell(b) = \{l\}$. Then $L = \{l, l'\}$ is not a minimal cover for $\alpha = \{a\}$ as $L' = \{l'\}$ is also a cover.

Example 2. Let $\ell(a) = \ell(b) = \{l, l'\}$. Then there are two minimal covers for $\alpha = \{a, b\}$, $L = \{l\}$ and $L' = \{l'\}$, which are not unique.

Example 3. Let $\ell(a) = \{l, l'\}$, $\ell(b) = \{l\}$ and $\ell(c) = \{l'\}$. Then $L = \{l, l'\}$ is a unique cover for $\alpha = \{a, b, c\}$.

Example 4. Let $\ell(a) = \{l_1, l_3\}$, $\ell(b) = \{l_1, l_4\}$, $\ell(c) = \{l_2, l_3\}$ and $\ell(d) = \{l_2, l_4\}$. Then $\alpha = \{a, b, c, d\}$ has two unique minimal covers: $L = \{l_1, l_2\}$ and $L' = \{l_3, l_4\}$.

Equipped with the concept of a unique minimal cover, we can reverse the inclusion in Proposition 6.

Proposition 10. *Let $\mathcal{NP}$ be a τ/ℓ-net system with partially localised conflicts and M be its reachable marking. If $\alpha \in \mathit{Enbld}^{min}_{\mathcal{NP}}(M)$ and all its minimal covers are unique, then:*

$$\mathit{satlocalities}_M(\alpha) = \bigcap\{\ell(t) \mid t \in \alpha\} \ .$$

Proof. We need to show the $(\supseteq)$ inclusion as the opposite one follows from Proposition 6. Suppose that $\widetilde{l} \in \bigcap\{\ell(t) \mid t \in \alpha\} \setminus \mathit{satlocalities}_M(\alpha)$.

From $\widetilde{l} \in \bigcap\{\ell(t) \mid t \in \alpha\}$ we have $\alpha = \alpha|_{\widetilde{l}}$. Since $\alpha \in \mathit{Enbld}^{min}_{\mathcal{NP}}(M)$, we have $\mathit{satlocalities}_M(\alpha) \cap \ell(t) \neq \varnothing$, for all $t \in \alpha$. Suppose $\widehat{l} \in \mathit{satlocalities}_M(\alpha) \cap \ell(\widetilde{t})$, for some $\widetilde{t} \in \alpha$. Notice that $\widehat{l} \neq \widetilde{l}$ as $\widetilde{l} \notin \mathit{satlocalities}_M(\alpha)$. From Proposition 4, $\alpha|_{\widehat{l}} \in \mathit{Enbld}_{\mathcal{NP}}(M)$.

We have $\alpha|_{\widehat{l}} \leq \alpha = \alpha|_{\widetilde{l}}$. We cannot have $\alpha|_{\widehat{l}} = \alpha|_{\widetilde{l}}$, because then both $\{\widetilde{l}\}$ and $\{\widehat{l}\}$ would be two different minimal covers for α, and so neither of them be unique. Hence $\alpha|_{\widehat{l}} < \alpha$, producing a contradiction with the minimality of α. □

Thus in terms of selecting clusters in the construction outlined earlier on, if C has been selected at a state q then, for every $\alpha \in \mathit{minSteps}_q$, we must have $C \in \mathit{satclusters}_q(\alpha)$ iff $\alpha|_C = \alpha$.

5 Concluding remarks

In this paper, we only initiated the investigation of intricate relationships between localities, conflicts and step covers. In the future research we plan to develop stronger results on this topic, aiming at an efficient synthesis procedure of τ-nets with localities with unknown locality mappings.

Acknowledgement We would like to thank Piotr Chrzastowski-Wachtel for his suggestion to investigate nets with overlapping localities. We are also grateful to the reviewers for their detailed and helpful comments. This research was supported by the Rae&Epsrc Davac Verdad projects, and Nsfc Grants 60910004 and 2010CB328102.

References

1. Badouel, E., Bernardinello, L., Darondeau, Ph.: The Synthesis Problem for Elementary Net Systems is NP-complete. Theoretical Computer Science **186** (1997) 107–134
2. Badouel, E., Darondeau, Ph.: Theory of Regions. In: Reisig, W., Rozenberg, G. (eds.): Lectures on Petri Nets I: Basic Models, Advances in Petri Nets. Lecture Notes in Computer Science **1491**. Springer-Verlag, Berlin Heidelberg New York (1998) 529–586
3. Bernardinello, L.: Synthesis of Net Systems In: Marsan, M.A. (ed.): Application and Theory of Petri Nets 1993. Lecture Notes in Computer Science **691**. Springer-Verlag, Berlin Heidelberg New York (1993) 89–105
4. Darondeau, P., Koutny, M., Pietkiewicz-Koutny, M., Yakovlev, A.: Synthesis of Nets with Step Firing Policies. Fundamenta Informaticae **94** (2009) 275–303
5. Desel, J., Reisig, W.: The Synthesis Problem of Petri Nets. Acta Informatica **33** (1996) 297–315
6. Dasgupta, S., Potop-Butucaru, D., Caillaud, B., Yakovlev, A.: Moving from Weakly Endochronous Systems to Delay-Insensitive Circuits. Electronic Notes in Theoretical Computer Science **146** (2006) 81–103
7. Desel, J., Reisig, W.: The Synthesis Problem of Petri Nets. Acta Informatica **33** (1996) 297–315
8. Kleijn, J., Koutny, M.: Processes of Membrane systems with Promoters and Inhibitors. Theoretical Computer Science **404** (2008) 112–126
9. Kleijn, H.C.M., Koutny, M., Rozenberg, G.: Towards a Petri Net Semantics for Membrane Systems. In: Freund, R., Paun, G., Rozenberg, G., Salomaa, A. (eds.): WMC 2005. Lecture Notes in Computer Science **3850**. Springer-Verlag, Berlin Heidelberg New York (2006) 292–309
10. Koutny, M., Pietkiewicz-Koutny, M.: Transition Systems of Elementary Net Systems with Localities. In: Baier, C., Hermanns, H. (eds.): CONCUR 2006. Lecture Notes in Computer Science **4137**. Springer-Verlag, Berlin Heidelberg New York (2006) 173–187
11. Koutny, M., Pietkiewicz-Koutny, M.: Synthesis of Elementary Net Systems with Context Arcs and Localities. Fundamenta Informaticae **88** (2008) 307–328
12. Koutny, M., Pietkiewicz-Koutny, M.: Synthesis of Petri Nets with Localities. Scientific Annals of Computer Science **19** (2009) 1–23
13. Mukund, M.: Petri Nets and Step Transition Systems. International Journal of Foundations of Computer Science **3** (1992) 443–478
14. Nielsen, M., Rozenberg, G., Thiagarajan, P.S.: Elementary transition systems. Theoretical Compututer Science **96** (1992) 3–33
15. Păun, G.: Membrane Computing, An Introduction. Springer-Verlag, Berlin Heidelberg New York (2002)
16. Pietkiewicz-Koutny, M.: The Synthesis Problem for Elementary Net Systems with Inhibitor Arcs. Fundamenta Informaticae **40** (1999) 251–283

Incremental Process Mining

Marc Solé[1] and Josep Carmona[2]

[1] Computer Architecture Department, UPC msole@ac.upc.edu
[2] Software Department, UPC jcarmona@lsi.upc.edu

Abstract. The problem of synthesis of Petri nets from transition systems or languages has many applications, ranging from CAD for VLSI to medical applications, among others. The most common algorithms to accomplish this task are based on the theory of regions. However, one of the problems of such algorithms is its space requirements: for real-life or industrial instances, some of the region-based algorithms cannot handle in memory the internal representation of the input or the exploration lattice required. In this paper, the incremental derivation of a basis of regions and the later partitioned basis exploration is presented, which allows the splitting of large inputs.

1 Introduction

The introduction of the theory of regions [1] in the early nineties enabled a new area of research that strives for transforming language or state-based representations into event-based ones. This transformation, known as *synthesis*, was initially devoted to derive a Petri net whose reachability graph was bisimilar or isomorphic to the input transition system. A variant of this problem, known as *mining*, has weaker requirements: the language of the derived Petri net must be a superset (maybe proper) of the language of the input transition system [2]. The theory of this paper provides algorithms for mining.

Many research has been carried out since the introduction of regions, specially of theoretical nature, which has brought a better understanding of the theory [3–6], and has provided meaningful extensions to make it more general [7–10].

As a consequence of the aforementioned theoretical work, tools based on the theory of regions started to be available by the end of the nineties [11, 12] and still new ones are developed nowadays [9, 10]. These tools are the outcome of bridging the gap between the theory and practice, and many of them are used extensively in the academic domain, whereas few are used in industry. The reasons for the limited success of these tools in industry might be:

1. The algorithms involved are complex, i.e. in general polynomial with the size of the input [3], that might be prohibitive for large inputs, and the use of efficient data structures like BDDs or heuristics only alleviates the problem.
2. No high-level techniques are provided to cope with the inherent complexity of the problem.

$$\begin{tikzcd}
\log L_1 \arrow[r, "\text{In [15]}"] & \mathsf{TS}\ A_1 \arrow[r, "\text{In [6,14]}"] & \text{Basis } B_1 \arrow[dr, "\text{Sect. 3}"] & & & \\
 & \mathsf{TS}\ A_2 \arrow[r] & \text{Basis } B_2 \arrow[r] & \bigcup \arrow[r] & \text{Basis } \bigcup_i B_i \arrow[r, "\text{Sect. 4}"] & \mathsf{PN}\ N \\
\vdots & \vdots & \vdots & & & \\
\mathsf{PN}\ N_n \arrow[r, "\mathsf{RG}(N_n)"] & \mathsf{TS}\ A_n \arrow[r] & \text{Basis } B_n \arrow[uur] & & &
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
 & \mathsf{TS}\ A_1 \arrow[r] & \text{Basis } B_1 \arrow[dr] & & & \\
\mathsf{TS}\ A \arrow[ur] \arrow[r] \arrow[ddr] & \mathsf{TS}\ A_2 \arrow[r] & \text{Basis } B_2 \arrow[r] & \bigcup \arrow[r] & \text{Basis } \bigcup_i B_i \arrow[r] & \mathsf{PN}\ N \\
 & \vdots & \vdots & & & \\
 & \mathsf{TS}\ A_n \arrow[r] & \text{Basis } B_n \arrow[uur] & & &
\end{tikzcd}$$

(b)

Fig. 1. Incremental Process Mining.

In this work we provide the theoretical basis for deriving high-level strategies that may allow to handle large specifications. More concretely, as space requirements are typically the bottleneck for some of the tools listed above, in this paper we present an incremental technique that allows splitting the input into several smaller parts. Moreover, we show how the theory of regions can be extended algorithmically to combine the regions of each part in order to derive the regions of the whole input.

The theory presented will be oriented for the problem of mining: given a set of objects describing behaviors (like a Petri net, a transition systems or an event log) $O_1, O_2, \ldots, O_n$, we want to obtain a Petri net N such that $\mathcal{L}(N) \supseteq \bigcup_{i=1}^{n} \mathcal{L}(O_i)$. The traditional way to solve this problem is to generate a transition system A_i for each O_i, join all these A_i to create a single transition system, and then apply a synthesis technique to derive a Petri net from a transition system [12, 13].

In this paper we explore a different approach. Instead of working with a monolithic transition system, we use the fact that a transition system can be represented by a basis of regions, such that any other region is a linear combination of the regions in the basis (a recent publication shows an efficient technique to accomplish this task [14]). Then, bases can be combined to obtain a region basis for the whole system, from which we can derive the Petri net N.

The general picture of the approach of this paper is outlined on Fig. 1(a). Some arcs are labeled with an indication of the section or the reference where the conversion/operation is explained. Basically the first step is to convert inputs that are not transition systems into a transition system, and then compute a region basis from which a PN can be generated. Another possible application is also shown in Fig. 1(b), where a large TS is split into smaller subsystems of manageable size, with the only restriction that all the subsystems must include the initial state and be connected.

1.1 Related work

In [16] an incremental approach was suggested based on the observation that any region of the union of two transition systems can be expressed as the union of regions of those systems. As in the approach presented in this paper, the approach in [16] trades space for time, since it must first compute all the regions and then obtain the minimal ones, a slower process than finding directly the minimal regions if the whole transition system fits into memory. The main drawback of this method is that the complete set of regions of each component transition system must be stored (either in memory or disk) in order to compute the set of regions of the union. In this work we propose a faster methodology based on the fact that the complete set of regions can be succintly represented by a basis of regions.

1.2 Organization

We start by giving the necessary background in Sect. 2. The process of combining a set of bases to produce a unique basis for the whole system is explained in Sect. 3. A description of the generation of a PN from a region basis is given in Sect. 4 and, finally, Sect. 5 concludes this paper.

2 Background

2.1 Finite Transition Systems and Petri Nets

Definition 1 (Transition system). *A* transition system *(TS) is a tuple $\langle S, \Sigma, T, \mathsf{s}_0 \rangle$, where S is a set of* states*, Σ is an alphabet of* actions*, $T \subseteq S \times \Sigma \times S$ is a set of* (labelled) transitions*, and $\mathsf{s}_0 \in S$ is the* initial state*.*

We use $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}'$ as a shortcut for $(\mathsf{s}, \mathsf{e}, \mathsf{s}') \in T$, and we denote its transitive closure as $\xrightarrow{*}$. A state s' is said to be *reachable from state* s if $\mathsf{s} \xrightarrow{*} \mathsf{s}'$. We extend the notation to transition sequences, i.e., $\mathsf{s}_1 \xrightarrow{\sigma} \mathsf{s}_{n+1}$ if $\sigma = \mathsf{e}_1 \ldots \mathsf{e}_n$ and $(\mathsf{s}_i, \mathsf{e}_i, \mathsf{s}_{i+1}) \in T$. We denote $\#(\sigma, \mathsf{e})$ the number of times that event e occurs in σ. Let $A = \langle S, \Sigma, T, \mathsf{s}_0 \rangle$ be a TS. We consider connected TSs that satisfy the following axioms: i) S and Σ are finite sets, ii) every event has an occurrence and iii) every state is reachable from the initial state. The *language* of a TS A, $\mathcal{L}(A)$, is the set of transition sequences feasible from the initial state.

For two TSs A_1 and A_2, when $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$, we will say that A_2 is an *over-approximation* of A_1.

Definition 2 (Union of TSs). *Given two TSs $A_1 = \langle S_1, \Sigma_1, T_1, \mathsf{s}_0 \rangle$ and $A_1 = \langle S_2, \Sigma_2, T_2, \mathsf{s}_0 \rangle$, the union of A_1 and A_2 is the TS $A_1 \cup A_2 = \langle S_1 \cup S_2, \Sigma_1 \cup \Sigma_2, T_1 \cup T_2, \mathsf{s}_0 \rangle$.*

Clearly, the TS $A_1 \cup A_2$ is an over-approximation of the TSs A_1 and A_2, i.e. $\mathcal{L}(A_1) \cup \mathcal{L}(A_2) \subseteq \mathcal{L}(A_1 \cup A_2)$.

Definition 3 (Petri net [17]). *A Petri net (PN) is a tuple (P, T, W, M_0) where the sets P and T represent disjoint finite sets of places and transitions, respectively, and $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the weighted flow relation. The initial marking $M_0 \in \mathbb{N}^P$ defines the initial state of the system.*

A transition $t \in T$ is *enabled* in a marking M if $\forall p \in P : M(p) \geq W(p, t)$. Firing an enabled transition t in a marking M leads to the marking M' defined by $M'(p) = M(p) - W(p, t) + W(t, p)$, for $p \in P$, and is denoted by $M \xrightarrow{t} M'$. The set of all markings reachable from the initial marking M_0 is called its Reachability Set. The *Reachability Graph* of N, denoted $\mathsf{RG}(N)$, is a transition system in which the set of states is the Reachability Set, the events are the transitions of the net and a transition (M_1, t, M_2) exists if and only if $M_1 \xrightarrow{t} M_2$. We use $\mathcal{L}(N)$ as a shortcut for $\mathcal{L}(\mathsf{RG}(N))$.

2.2 Generalized Regions

The theory of regions [1, 5] provides a way to derive a Petri net from a transition system. Intuitively, a region corresponds to a place in the derived Petri net. In the initial definition, a region was defined as a subset of states of the transition system satisfying a homogeneous relation with respect to the set of events. Later extensions [7, 18, 10] generalize this definition to multisets, which is the notion used in this paper.

Definition 4 (Multiset, k-bounded Multiset, Subset). *Given a set S, a multiset r of S is a mapping $r : S \to \mathbb{Z}$. The number $r(s)$ is called the* multiplicity *of s in r. Multiset r is k-bounded if all its multiplicities are less or equal than k. Multiset r_1 is a* subset *of r_2 ($r_1 \subseteq r_2$) if $\forall s \in S \ : \ r_1(s) \leq r_2(s)$.*

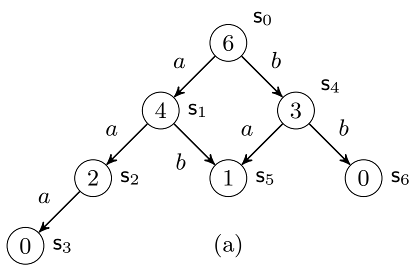
We define the following operations on multisets:

Definition 5 (Multiset operations).

Maximum power	$\mathrm{pow}(r) = \max_{s \in S} r(s)$
Minimum power	$\mathrm{minp}(r) = \min_{s \in S} r(s)$
Scalar product	$(k \cdot r)(s) = k \cdot r(s)$, *for* $k \in \mathbb{Z}$
Scalar sum	$(r + k)(s) = r(s) + k$, *for* $k \in \mathbb{Z}$
Union	$(r_1 \cup r_2)(s) = \max(r_1(s), r_2(s))$
Sum	$(r_1 + r_2)(s) = r_1(s) + r_2(s)$
Subtraction	$(r_1 - r_2)(s) = r_1(s) - r_2(s)$

The operations described above have algebraic properties, e.g., $r + r = 2 \cdot r$ and $r_1 - k \cdot r_2 = r_1 + (-k) \cdot r_2$.

Definition 6 (Gradient). *Let $\langle S, \Sigma, T, \mathsf{s}_0 \rangle$ be a TS. Given a multiset r and a transition $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T$, its* gradient *is defined as $\delta_r(\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}') = r(\mathsf{s}') - r(\mathsf{s})$. If all the transitions of an event $\mathsf{e} \in \Sigma$ have the same gradient, we say that the event e has* constant gradient*, whose value is denoted as $\delta_r(\mathsf{e})$.*



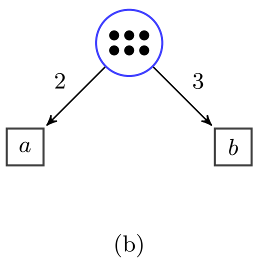


Fig. 2. (a) Region in a TS: $r(\mathsf{s}_0) = 6, r(\mathsf{s}_1) = 4, \ldots, r(\mathsf{s}_6) = 0$, (b) corresponding place in the Petri net.

Definition 7 (Region). *A* region *r is a multiset defined in a TS, in which all the events have constant gradient.*

Example 1. Fig. 2(a) shows a TS. The numbers within the states correspond to the multiplicity of the multiset r shown. Multiset r is a region because both events a and b have constant gradient, i.e. $\delta_r(\mathsf{a}) = -2$ and $\delta_r(\mathsf{b}) = -3$. There is a direct correspondence between regions and places of a PN. The gradient of the region describes the flow relation of the corresponding place, and the multiplicity of the initial state indicates the number of initial tokens [10]. Fig. 2(b) shows the place corresponding to the region shown in Fig. 2(a).

We say that region r is *normalized* if $\mathrm{minp}(r) = 0$. Similarly, it is *non-negative* if $\mathrm{minp}(r) \geq 0$. Any region r can become normalized by subtracting $\mathrm{minp}(r)$ from the multiplicity of all the states.

Definition 8 (Normalization). *We denote by $\downarrow\! r$ the normalization of a region r, so that $\downarrow\! r = r - \mathrm{minp}(r)$.*

It is useful to define a normalized version of the sum operation between regions, since it is closed in the class of normalized regions.

Definition 9 (Normalized sum). *Let r_1 and r_2 be normalized regions, we denote by $r_1 \oplus r_2$ their* normalized sum, *so that $r_1 \oplus r_2 = \downarrow\!(r_1 + r_2)$.*

Definition 10 (Gradient vector). *Let r be a region of a TS whose set of events is $\Sigma = \{\mathsf{e}_1, \mathsf{e}_2, \ldots, \mathsf{e}_n\}$. The* gradient vector of r, *denoted as $\Delta(r)$, is the vector of the event gradients, i.e. $\Delta(r) = (\delta_r(\mathsf{e}_1), \delta_r(\mathsf{e}_2), \ldots, \delta_r(\mathsf{e}_n))$.*

Proposition 1. *Gradient vectors have the following properties:*

$$\begin{array}{cc}
\Delta(r_1 + r_2) = \Delta(r_1) + \Delta(r_2) & \Delta(k \cdot r) = k \cdot \Delta(r) \\
\Delta(r + k) = \Delta(r) & \Delta(r_1 - r_2) = \Delta(r_1) - \Delta(r_2) \\
\Delta(r_1 \oplus r_2) = \Delta(r_1) + \Delta(r_2) &
\end{array}$$

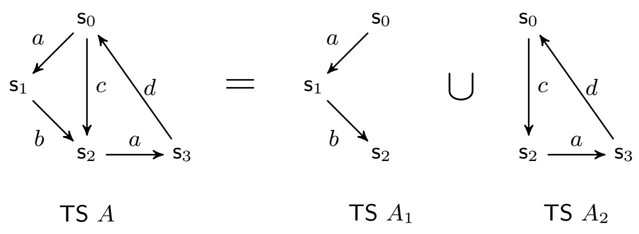


Fig. 3. TS A is split into two subsystems A_1 and A_2.

Regions can be partitioned into classes using their gradient vectors.

Definition 11 (Canonical region). *Two regions r_1 and r_2 are said to be equivalent if their gradient is the same,* i.e. $r_1 \equiv r_2 \Leftrightarrow \Delta(r_1) = \Delta(r_2)$. *Given a region r, the equivalence class of r, is defined as $[r] = \{r_i | \ r_i \equiv r\}$. A canonical region is the normalized region of an equivalence class,* i.e. $\downarrow r$.

Example of canonical regions are provided in Fig. 4, where two TSs are shown in which some regions have been shadowed. For instance, the canonical region $r_0 = \{s_1, s_2\}$ has gradient vector $\Delta(r_0) = (1,0)$. A PN built from the set of minimal canonical regions has the same language as a PN built using all the regions [5], thus it yields the smallest overapproximation with respect to the language of the TS [10].

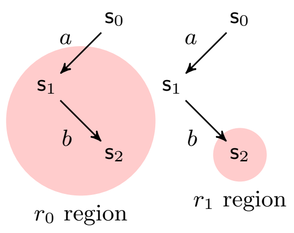
Definition 12 (Subregion, Empty region, Minimal canonical region). *r_1 is a* subregion *of r_2, denoted as $r_1 \sqsubseteq r_2$, if, for any state s, $\downarrow r_1(s) \leq \downarrow r_2(s)$. We denote by $\emptyset$ the region in which all states have zero multiplicity. A minimal canonical region r satisfies that for any other region r', if $r' \sqsubseteq r$ then $r' \equiv \emptyset$.*

3 Combining region bases

In this section we detail how region bases of different TSs can be joined, yielding a region basis of their union. We will illustrate the theory with the running example shown in Fig. 3. In this case, we assume A is a very large TS that cannot be easily handled, hence it is split into two smaller subsystems, namely A_1 and A_2, so that $A = A_1 \cup A_2$.

3.1 Basis of regions

Definition 13 (Region basis). *Given a TS, a region basis $B = \{r_1, r_2, \ldots, r_n\}$ is a minimal subset of the canonical regions of TS such that any region r can be expressed as a linear combination of B (*i.e. $r = \sum_{i=1}^{n} c_i \cdot r_i$*, with $c_i \in \mathbb{Q}$, $r_i \in B$).*



Event	$\Delta(r_0)$	$\Delta(r_1)$
a	1	0
b	0	1

Event	$\Delta(q_0)$	$\Delta(q_1)$
a	-1	-1
c	1	0
d	0	1

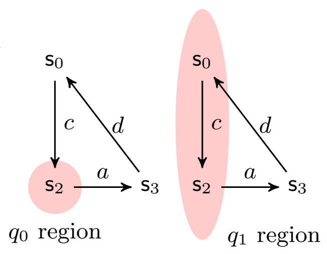


Fig. 4. Region bases for A_1 and A_2.

The set of canonical regions of a TS, together with the normalized sum operation, forms a free Abelian group [18]. Consequently, there exists a *basis* (*i.e.* subset of the group) such that every element in the group can be rewritten as a unique linear combination of the basis elements.

Region bases are interesting because, as the following theorem states, their size is usually small.

Theorem 1 ([18]). *Let $\langle S, \Sigma, T, s_0 \rangle$ be a TS. The size of the region basis is less or equal to* $\min(|\Sigma|, |S| - 1)$.

3.2 Region compatibility

Given two systems described by their region basis, we want to obtain the region basis of their union TS. The work more closely related is [16], in which it is described how the set of regions in the joined system can be obtained from the regions of the component systems. This is achieved by introducing the concept of compatible (standard) regions. In this section we first review and extend this concept of compatibility to generalized regions.

Definition 14 (Compatible TSs). *Two TSs A_1 and A_2 are compatible if they have the same initial state.*

Def. 14 is more general than the one in [16], where the number of shared states is restricted to one (the initial state). For instance, systems A_1 and A_2 of Fig. 3 are compatible according to this definition, but not using the definition in [16], since they share states s_0 and s_2.

Definition 15 (Compatible regions, offset). *Two regions r_1 and r_2 from two compatible TSs $A_1 = \langle S_1, \Sigma_1, T_1, s_0 \rangle$ and $A_2 = \langle S_2, \Sigma_2, T_2, s_0 \rangle$ are said to be* compatible *if:*

– $\forall e \in \Sigma_1 \cap \Sigma_2, \delta_{r_1}(e) = \delta_{r_2}(e)$, i.e. *they have the same gradient for all common events, and,*

– $\exists k \in \mathbb{Z} : \forall s \in S_1 \cap S_2, r_2(s) - r_1(s) = k$, i.e. *the difference in multiplicity of each shared state is equal to a constant, that we call the* offset *between* r_1 *and* r_2*, denoted as* $\text{off}(r_1, r_2)$.

Two compatible regions are said to be directly compatible *if* $\text{off}(r_1, r_2) = 0$*, a fact that we denote as* $r_1 \leftrightarrow r_2$*. Conversely, if* $\text{off}(r_1, r_2) \neq 0$*, we say that the regions are* indirectly compatible *and we use the following notation* $r_1 \leftrightsquigarrow r_2$.

An immediate consequence of Def. 15 is that, if there is only a single shared state, then any two regions with the same gradient for all common events are compatible. This is, for instance, the case when TSs represent execution trees and only the initial state is shared among them.

Two compatible regions can be made directly compatible by adding the offset to one of them.

Definition 16 (Directly compatible region). *Given two compatible regions* r_1 *and* r_2 *with* $\text{off}(r_1, r_2) > 0$*, the directly compatible region of* r_1 *with respect to* r_2 *is* $r_1{\uparrow^{r_2}} = r_1 + \text{off}(r_1, r_2)$.

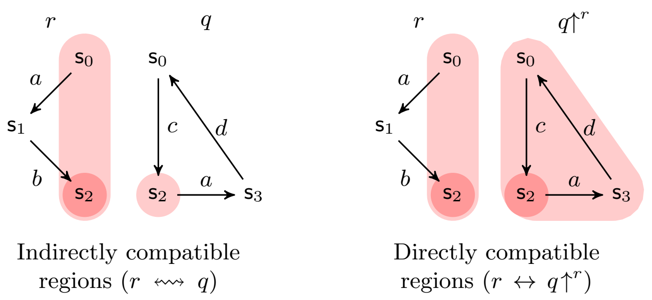
Definition 17 (Union of compatible regions). *Given two compatible regions* r_1 *and* r_2*, defined over sets of states* S_1 *and* S_2*, respectively, with* $\text{off}(r_1, r_2) > 0$*, their union, denoted* $r_1 \sqcup r_2$*, is*

$$(r_1 \sqcup r_2)(s) = \begin{cases} (r_1{\uparrow^{r_2}})(s) & \text{if } s \in S_1 \\ r_2(s) & \text{otherwise} \end{cases}$$

Proposition 2 (Union of compatible regions is a region of union system). *Given two compatible regions* r_1 *and* r_2 *of two compatible TSs* A_1 *and* A_2*,* $r_1 \sqcup r_2$ *is a region of* $A_1 \cup A_2$*. For each shared event, its gradient is equal to the gradient in* r_1 *or* r_2*, which are equal. For non-shared events of* A_1 *(*A_2*), their gradient is the gradient in* A_1 *(*A_2*).*

Proof. Assume $\text{off}(r_1, r_2) \geq 0$. Region $r = r_1 \sqcup r_2$, where $r_1{\uparrow^{r_2}} = r_1 + \text{off}(r_1, r_2)$. The latter entails that, in a shared state s, the multiplicity is the same for $r_1{\uparrow^{r_2}}$ and r_2, which is the multiplicity assigned to r. For non-shared states, on the other hand, the multiplicity assigned in r is the multiplicity of the region whose TS contains the state. Thus any arc (either going from a shared to non-shared state or any other combination) has a constant gradient, equal to the gradient of that event in the corresponding region. Result transfers to r_1 because $\Delta(r_1) = \Delta(r_1{\uparrow^{r_2}})$. □

Example 2. In Fig. 5 we show one region of each subsystem of our running example. They are compatible, since the gradient of the single shared event a is the same in both regions. More specifically, they are indirectly compatible, since their offset is different than 0.



r
q
s_0
a
s_1
b
s_2
c
d
s_3
Indirectly compatible regions ($r \rightsquigarrow q$)
r
$q{\uparrow}^r$
Directly compatible regions ($r \leftrightarrow q{\uparrow}^r$)

Fig. 5. Region r of A_1 and q of A_2, are indirectly compatible.

Property 1. Given two compatible non-negative regions r_1 and r_2. If they are directly compatible then $r_1 \sqcup r_2$ is a normalized region, if and only if, one of them is normalized. If they are not directly compatible, but both of them are normalized, then again $r_1 \sqcup r_2$ is a normalized region.

Proof. If one of them is normalized and they are directly compatible no modification of multiplicities is performed, so the state with 0 multiplicity will remain untouched and since regions are non-negative, then minp$(r_1 \sqcup r_2) = 0$. Conversely, if $r_1 \sqcup r_2$ is a normalized region and $r_1 \leftrightarrow r_2$, then all multiplicities of either r_1 or r_2 are greater or equal to 0, and since there is at least one 0 multiplicity, at least one of them must be normalized. If both are normalized but are not directly compatible, only one of them modifies its multiplicities and we end up in the same situation. □

3.3 Incremental algorithm for obtaining a basis

Given two TSs, A_1 and A_2. Let $\{r_1, \ldots, r_n\}$ be the region basis of A_1 and $\{q_1, \ldots, q_m\}$ the basis of A_2. The set of all regions of the union system $A_1 \cup A_2$ whose language satisfies $\mathcal{L}(A_1 \cup A_2) \supseteq \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$ is obtained by finding all non-trivial solutions for x_i variables that satisfy:

$$\forall \mathsf{e}_i \in \Sigma_1 \cap \Sigma_2 : \sum_{1 \le j \le n} \delta_{r_j}(\mathsf{e}_i) \cdot x_j = \sum_{1 \le k \le m} \delta_{q_k}(\mathsf{e}_i) \cdot x_{n+k}$$

i.e. for all common events their gradient must be the same on both systems.

This system of equations can be rewritten in matrix form as $M \cdot \boldsymbol{x} = \mathbf{0}$, where $\boldsymbol{x}$ is the column vector of variables x_i and M is the following matrix:

$$M = \begin{pmatrix} \delta_{r_1}(\mathsf{e}_1) \ldots \delta_{r_n}(\mathsf{e}_1) & -\delta_{q_1}(\mathsf{e}_1) \ldots -\delta_{q_m}(\mathsf{e}_1) \\ \delta_{r_1}(\mathsf{e}_2) \ldots \delta_{r_n}(\mathsf{e}_2) & -\delta_{q_1}(\mathsf{e}_2) \ldots -\delta_{q_m}(\mathsf{e}_2) \\ \vdots & \vdots \\ \delta_{r_1}(\mathsf{e}_c) \ldots \delta_{r_n}(\mathsf{e}_c) & -\delta_{q_1}(\mathsf{e}_c) \ldots -\delta_{q_m}(\mathsf{e}_c) \end{pmatrix}$$

where c is the number of common events in the system, *i.e.* $c = |\Sigma_1 \cap \Sigma_2|$. We call this matrix the *gradient compatibility matrix* between A_1 and A_2.

Compatibility of regions demands not only the gradients of common events to be the same, but also that the offset is the same for all shared states. To enforce such condition, assume that we shift all the regions in the bases $\{r_1, \dots, r_n\}$ and $\{q_1, \dots, q_m\}$ so that for all r_i and q_j we have that $r_i(\mathsf{s}_0) = q_j(\mathsf{s}_0) = 0$. Now any region obtained by combining the regions in the bases will have a 0 multiplicity in the initial state. Thus, if the offset is the same in all shared states, their multiplicity must coincide in all remaining shared states.

If there is a path between s_0 and shared state s in which the same events (and the same number of times but no matter in which order) are fired in both TSs A_1 and A_2, then the condition is automatically satisfied. Only in the case where all paths are different, we say that shared state s is *in conflict*, and then we must explicitly enforce the equality of the multiplicity of s in both systems.

This condition can be expressed in matrix form using a row for each shared state in conflict (different than s_0 since this state is never in conflict). For a shared state s in conflict, its corresponding row will be of the form $(r_1(\mathsf{s}) \dots r_n(\mathsf{s}) - q_1(\mathsf{s}) \dots q_m(\mathsf{s}))$, where all regions r_i and q_j have been shifted, as said before, so that $r_i(\mathsf{s}_0) = q_j(\mathsf{s}_0) = 0$. Let us name as C the matrix containing such rows, we will call it the *shared state conflict matrix* between A_1 and A_2. Now since the multiplicity of all these states must be the same, their subtraction must be 0. Thus, $C \cdot \boldsymbol{x} = \boldsymbol{0}$.

Theorem 2. *Let A_1 and A_2 be two compatible TSs with region bases $\{r_1, \dots, r_n\}$ and $\{q_1, \dots, q_m\}$ respectively. Let M be their gradient compatibility matrix, C be the shared state conflict matrix, and $\boldsymbol{x}$ the column vector of variables x_1 to x_{n+m}. Consider an assignment to the variables in $\boldsymbol{x}$ such that $\binom{M}{C} \cdot \boldsymbol{x} = \boldsymbol{0}$ and some $x_i \neq 0$. This assignment identifies a region $r \sqcup q = \sum_{1 \leq i \leq n} r_i x_i \sqcup \sum_{1 \leq j \leq m} q_j x_{n+j}$ in $A_1 \cup A_2$.*

Proof. A non-trivial solution $\boldsymbol{x}$ defines two compatible regions, namely $r = \sum_{1 \leq i \leq n} r_i x_i$ and $q = \sum_{1 \leq j \leq m} q_j x_{n+j}$, in A_1 and A_2 respectively. These two regions are compatible according to Definition 15 if $M \cdot \boldsymbol{x} = \boldsymbol{0}$, because for common events the gradient is the same and the offset is the same in all shared states if $C \cdot \boldsymbol{x} = 0$. By Proposition 2, $r \sqcup q$ is a region of $A_1 \cup A_2$. □

So the problem reduces to finding the solutions to a homogeneous linear system. Note that the system does not require to have solutions in the integer domain. In fact all the solutions are in $\mathbb{Q}$, since all the gradients are integers. Homogeneous linear systems have one trivial solution (*i.e.* $\boldsymbol{0}$) and infinite non-trivial solutions. Let $\boldsymbol{y_i}$ be all the non-redundant solution vectors, then any possible solution of the system can be obtained by linear combination of these solution vectors, since adding or subtracting $\boldsymbol{0}$ from $\boldsymbol{0}$ does not change its value. These $\boldsymbol{y_i}$ are a basis of the nullspace of $\binom{M}{C}$, and any solution $\boldsymbol{x}$ can be written as a unique linear combination $\boldsymbol{x} = \sum_i \lambda_i \boldsymbol{y_i}$, with $\lambda_i \in \mathbb{Q}$.

There are several well-known methods to obtain a basis for the nullspace [19], one of the easiest is to put the matrix in reduced row echelon form, determine the

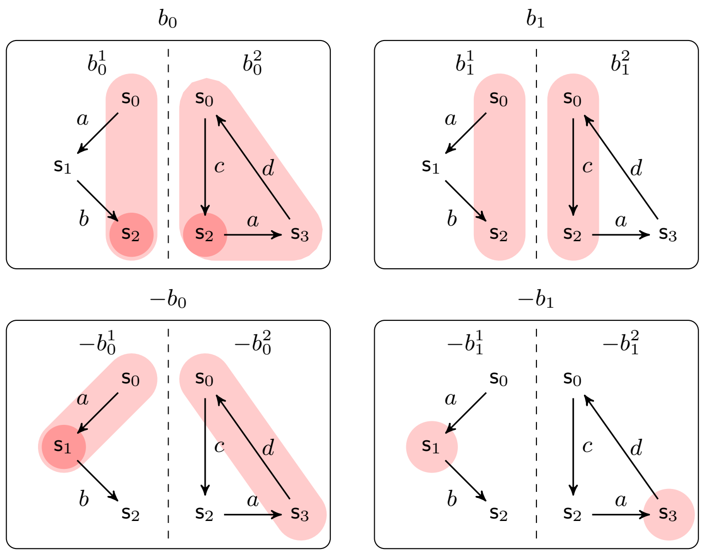


Fig. 6. Region basis $\{b_0, b_1\}$ (and their coregions $\{-b_0, -b_1\}$) for system $A_1 \cup A_2$. Regions are partitioned so that $b_0 = b_0^1 \cup b_0^2$, where b_0^i is the part of b_0 in A_i.

free variables, and then, for each free variable x_i, derive a vector of the basis by assigning 1 to x_i and 0 to the rest of free variables. The $\boldsymbol{y_i}$ columns correspond to the combination of regions of both system that have the same gradient, and the resulting combined region is a region basis for the union TS.

Example 3. We will compute the region basis of the union of TSs A_1 and A_2 shown in Fig. 3. Two possible region basis for these systems are $\{r_0, r_1\}$ and $\{q_0, q_1\}$, show in Fig. 4. The matrix M in this case is

$$M = \begin{pmatrix} 1 \; 0 \; 1 \; 1 \end{pmatrix}$$

where columns (from left to right) correspond to regions r_0, r_1, q_0, q_1 and there is only a single row that corresponds to the only shared event between A_1 and A_2, namely event a.

The set of shared states is $\{s_0, s_2\}$, thus the shared state conflict matrix C has only one row because only state s_2 is reachable from s_0 by firing a different multiset of events. Consequently $C = \begin{pmatrix} r_0(s_2) \; r_1(s_2) \; -q_0(s_2) \; -q_1(s_2) \end{pmatrix}$. With matrices M and C we can now build

$$\begin{pmatrix} M \\ C \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & -1 & 0 \end{pmatrix} \xrightarrow[\text{echelon form}]{\text{reduced row}} \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & -2 & -1 \end{pmatrix}$$

which is an indeterminate system with 2 degrees of freedom. We can write $x_1 = 2x_2 + x_3$ and $x_0 = -x_2 - x_3$, so by changing the values of variables x_2 and x_3

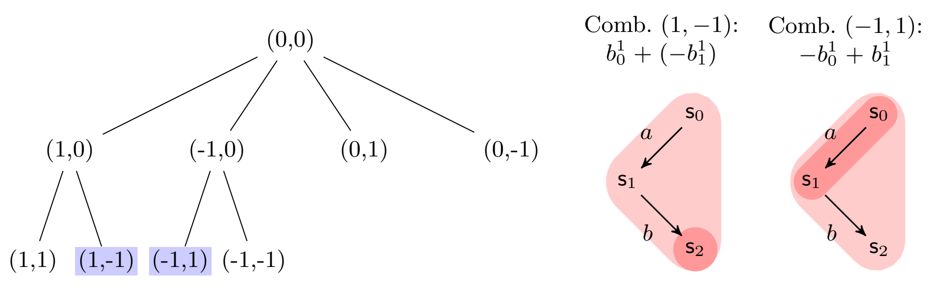


Fig. 7. Exploration of the region space. Each node represents a combination of the basis $\{b_0, b_1\}$ that will be explored. The combinations shaded in blue are the ones for which A_1 yields a non-normalized region (shown on right), thus only these combinations would be checked in A_2.

we can generate all the possible solutions. Given these parameters the region solution will be $((-x_2 - x_3) \cdot r_0 + (2x_2 + x_3) \cdot r_1) \sqcup (x_2 \cdot q_0 + x_3 \cdot q_1)$. Using the parameter values $(1, 0)$, and $(0, 1)$ for vector (x_2, x_3) we do obtain the following regions in the joined system: $b_0 = (-r_0 + 2r_1) \sqcup q_0$ and $b_1 = (-r_0 + r_1) \sqcup q_1$. The set $\{b_0, b_1\}$ forms a region basis for the $A_1 \cup A_2$ system (see Fig. 6).

4 Generating a PN from a basis

In [14] an algorithm was presented that allows finding minimal regions by careful exploration of the region space defined by the region basis. The fundamental idea is that the regions in the basis are initially assumed to be minimal, and then combinations of these regions can only yield a minimal region if the resulting region is non-normalized, since otherwise the region is a superregion of any of its component regions.

However this approach cannot be directly used if the number of states in the monolithic TS is too high to easily perform region operations in memory. The alternative we propose in this paper is to partition the region operations into the different component TSs, so that each time only the information of one of the systems is accessed.

To achieve this partitioning, consider the region basis $\{b_0, \ldots, b_n\}$, we denote by b_i^j the part of region b_i that belongs to subsystem j. All the regions in the basis are assumed to be normalized, but the different b_i^j could be non-normalized, since they are defined so that, given i, all b_i^j are directly compatible (cf. Def. 15). For instance region b_0 in Fig. 6 is normalized, however b_0^2 it is not.

Consequently a combination of the basis $\sum_i c_i \cdot b_i$ yields a non-normalized region only if, for all subsystem j, $\sum_i c_i \cdot b_i^j$ is a non-normalized region (by

Partial order $\subseteq$ in A_1

$$\begin{tikzcd}
 & (0,0) \arrow[dl] \arrow[dr] & \\
(1, -1) \arrow[d] & & (0, -1) \arrow[d] \\
(0, 1) \arrow[d] & & (-1, 1) \arrow[d] \\
(1, 0) & & (-1, 0)
\end{tikzcd}$$

Partial order $\subseteq$ in A_2

$$\begin{tikzcd}
 & (0,0) \arrow[dl] \arrow[dr] & \\
(-1, 1) \arrow[dd] \arrow[dr] & & (0, -1) \arrow[dl] \arrow[dd] \\
 & (-1, 0) \arrow[dd] & \\
(0, 1) \arrow[dr] & & (1, -1) \arrow[dl] \\
 & (1, 0) &
\end{tikzcd}$$

Fig. 8. Partial orders between combinations of regions in each subsystem, according to the subset relation on multisets ($\subseteq$).

Property 1)[3]. Thus, the strategy would be to test the basis combinations in the first subsystem, keeping only the combinations producing non-normalized regions. Then only these combinations will be tested in the second subsystem, discarding the ones that yield a normalized region, and the process will continue until all subsystems have been checked.

Finally, with only the surviving combinations, the subregion test will be performed to guarantee that only the minimal regions are found. Again this test can be distributed among the subsystems, since $\sum_i c_i \cdot b_i \subseteq \sum_i c'_i \cdot b_i$ if, and only if, for all subsystem j, $\sum_i c_i \cdot b_i^j \subseteq \sum_i c'_i \cdot b_i^j$.

We will illustrate all this process using our running example. In Fig. 7 we can see a tree of combinations of the $\{b_0, b_1\}$ basis. Each node is a tuple (c_0, c_1) describing the coefficients used to obtain a region r as $c_0 \cdot b_0 + c_1 \cdot b_1$. The second level of the tree corresponds to the regions in the basis and their coregions (as shown in Fig. 6). To bound the search space, assume we arbitrarily fix that the coefficients are only allowed to take values in the set $\{-1, 0, 1\}$.

From the four possible combinations in the third level, two of them yield already normalized regions in A_1, namely combinations $(1, 1)$ and $(-1, -1)$. On the other hand, combinations $(1, -1)$ and $(-1, 1)$ correspond to non-normalized regions in A_1, as shown in the figure. Consequently only these two combinations will be checked in subsystem A_2. In this case, both regions, namely, $b_0^2 + (-b_1^2)$ and $-b_0^2 + b_1^2$, are also non-normalized in A_2. Thus these combinations correspond to non-normalized regions in $A_1 \cup A_2$, which means that they might be minimal regions (once normalized).

[3] Since Property 1 only holds for non-negative regions, negative c_i coefficients are treated as markers for summing the normalized coregions of b_i. For instance $2b_1 - 3b_2$ will be actually computed as $2 \cdot b_1 + 3 \cdot {\downarrow}(-b_2)$. This way all regions are always non-negative and Property 1 can be safely used.

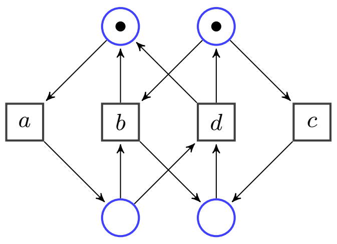


Fig. 9. Mined PN.

At this point we must check for minimal regions. The list of candidates includes all the regions in the basis (and their coregions), that is all the combinations in the second level, as well as combinations $(1, -1)$ and $(-1, 1)$ that have been found during the exploration. To check for minimality, we create a partial order among all these combinations in each subsystem (see Fig. 8). A region is not minimal if there is a combination, different than $(0, 0)$, that appears before it in all the partial orders. In our example, combinations $(1, 0)$ and $(-1, 0)$ are not minimal. Conversely, for instance $(-1, 1)$ is minimal because, although combination $(0, -1)$ precedes it in A_1 (i.e. $-b_1^1 \subseteq -b_0^1 + b_1^1$), it is not longer true in A_2 (i.e. $-b_1^2 \not\subseteq -b_0^2 + b_1^2$).

With the minimal regions found we build the mined PN of Fig. 9.

For a set of subsystems $A_1, \ldots, A_n$ the algorithm can be summarized as follows:

- Take the first subsystem (A_1) and explore region space until either combinations are exhausted (according to the user defined bounds on the coefficients) or the border of non-normalized regions is found.
- Pass to next subsystem the set of combinations of non-normalized regions and check for normality in this other subsystem.
  - If region is normalized, then discard, but mark sibling combinations to be explored in the current subsystem (since all sibling combinations in the first subsystem will remain to be non-normalized).
  - On the other hand, if region is non-normalized and we are not in the last subsystem, then add it to the list of regions that conform the border of non-normalized regions that will be passed to next subsystem. If we are already in the last subsystem, then add the combination to the list of candidate minimal regions, normalize it, and then check the normalization status in all subsystems. Sibling combinations are scheduled to be explored in the first subsystem whose subpart of the region is normalized.

5 Conclusions

In this paper we have extended the theory developed in [14] to devise an incremental algorithm for process mining. Given that space requirements are often the real bottleneck of some of the region-based techniques in the literature, methods like the one presented in this paper might represent a crucial step into making the theory applicable in industrial scenarios.

References

1. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures. Part I, II. Acta Informatica **27** (1990) 315–368
2. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9) (2004) 1128–1142
3. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. Lecture Notes in Computer Science **915** (1995) 364–383
4. Hoogers, P.W., Kleijn, H.C.M., Thiagarajan, P.S.: An event structure semantics for general petri nets. Theor. Comput. Sci. **153**(1&2) (1996) 129–170
5. Desel, J., Reisig, W.: The synthesis problem of Petri nets. Acta Inf. **33**(4) (1996) 297–315
6. Badouel, E., Darondeau, P.: Theory of regions. In Reisig, W., Rozenberg, G., eds.: Petri Nets. Volume 1491 of LNCS., Springer (1998) 529–586
7. Mukund, M.: Petri nets and step transition systems. Int. Journal of Foundations of Computer Science **3**(4) (1992) 443–478
8. Darondeau, P.: Deriving unbounded petri nets from formal languages. In Sangiorgi, D., de Simone, R., eds.: CONCUR. Volume 1466 of Lecture Notes in Computer Science., Springer (1998) 533–548
9. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of petri nets from finite partial languages. Fundam. Inform. **88**(4) (2008) 437–468
10. Carmona, J., Cortadella, J., Kishinevsky, M.: New region-based algorithms for deriving bounded Petri nets. IEEE Transactions on Computers **59**(3) (2009)
11. Caillaud, B.: Synet : A synthesizer of distributable bounded Petri-nets from finite automata. http://www.irisa.fr/s4/tools/synet/ (2002)
12. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets from finite transition systems. IEEE Trans. on Computers **47**(8) (1998) 859–882
13. Carmona, J., Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: A symbolic algorithm for the synthesis of bounded Petri nets. In: Application and Theory of Petri Nets and Other Models of Concurrency. (2008)
14. Solé, M., Carmona, J.: Process mining from a basis of state regions. In: Application and Theory of Petri Nets and other Models of Concurrency. LNCS, Springer (2010) 226–245
15. van der Aalst, W., Rubin, V., Verbeek, H., van Dongen, B., Kindler, E., Günther, C.: Process mining: a two-step approach to balance between underfitting and overfitting. Software and Systems Modeling (2009)
16. Dongen, B.F.V., Busi, N., Pinna, G.M., van der Aalst, W.: An iterative algorithm for applying the theory of regions in process mining. In: Proceedings of the Workshop on Formal Approaches to Business Processes and Web Services (FABPWS'07). (2007) 36–55
17. Murata, T.: Petri Nets: Properties, analysis and applications. Proceedings of the IEEE (April 1989) 541–580
18. Bernardinello, L., Michelis, G.D., Petruni, K., Vigna, S.: On the synchronic structure of transition systems. In Desel, J., ed.: Structures in Concurrency Theory, Proc. of the Int. Workshop on Structures in Concurrency Theory. (1995) 69–84

19. Kalman, D.: Basic null space calculations. The College Mathematics Journal **15**(1) (1984) 42–47

PetriFlow: A Petri Net Based Framework for Modelling and Control of Workflow Processes

Martin Riesz, Martin Sečkár, and Gabriel Juhás

Faculty of Electrical Engineering and Information Technology,
Slovak University of Technology,
Ilkovičova 3, 812 19 Bratislava, Slovak Republic
{martin.riesz,xseckar,gabriel.juhas}@stuba.sk

Abstract. The paper presents an architecture of a Petri net based framework for modelling and control of workflow processes. It focuses on the PNEditor module and briefly discusses workflow engine based on the models designed using the PNEditor. Then the paper describes method of synthesis *Separating feasible places* and an algorithm for reducing the number of places in the resulting Petri net.

Key words: PetriFlow, PNEditor, workflow management, synthesis

1 Introduction

There are many different Petri net tools and Petri net based tools for modelling workflow processes, such as CPN ([1]), Viptool ([2], [3]), Yasper ([4]) or ProM ([5]), to mention just some of them. The question arises why to implement another one. Most of them are determined to create models and some of them to analyze the models. However, models are only the first step in a business process management systems (BPMS). The main advantage of BPMS is that the models can be used to control the workflow process according to the designed model using a workflow engine.

The problem of the most existing editors (let us just mention Viptool as a prominent example), is that they were implemented with different aims, mostly to analyze the model, but they do not provide all the information needed for the generation of a deployable application, such as resource management.

Another problem is with the case generation. Usually, a model obtained via a Petri net modelling tool can be understood as a general definition of a model of a process, while the single cases can be understood as instances of the process. Using an analogy with object-oriented programming, a model can be understood as a class, while single cases can be understood as objects of that class. In Petri net based modelling tools, the realization of cases is often done using coloured Petri nets ([6]). But in such tools, colours are used both for distinguishing cases from each other as well as for modelling the data of the cases.

Another important feature for a successful application of models is a hierarchy, which enables not only to model on different level of abstraction but also to deploy reusable components. This is important for a practical use, as the business consultants, which design the models often do not offer them of sufficient level of details. Many tools offer different kinds of hierarchical nets, but because they also mostly implement a semantical framework with aim to preserve some properties, they are mostly too complex and/or too restrictive to generate a deployable code.

2 PetriFlow: A Brief Introduction

For the above mentioned reason, we develop a new framework for modelling and control of workflow processes based on Petri nets. Presently, it consists of two modules, PNEditor and PNEngine.

PNEditor enables to design a model, which is basically a place/transition nets enriched with the feature for distinguishing between static and dynamic places ([7]), where the static places correspond to static variables (they exists once for a process), while the dynamic places are constructed for each case. Moreover PNEditor enables modelling with subnets, where the subnets represents only a visual tool for designer, i.e. on semantical level, PNEditor works with a flat place/transition nets. Another important feature of PNEditor is that it provides a resource management on a very simple and intuitive way. It enables to create roles, where a role is basically a subset of the set of transitions, determining which transitions (which tasks) can be performed by users having the role. Further is PNEditor able to synthesize Petri nets from process logs.

The development version of the PetriFlow PNEditor can be downloaded via: `http://pneditor.matmas.net/`

PNEngine is a light version of a workflow engine based on the models provided by PNEditor. User registered in PNEngine is able to upload processes and thus becoming their owner. The owner of the process can then assign other users to roles defined in the process. Only user assigned to a given role is able to fire transitions contained in the role. Further, PNEngine enables to create new cases for a given process and to control the cases processing according to the business logic given by the Petri net modelling the process. The business logic layer is implemented in J2EE, the connection with the database and persistence of the cases is realized using Hibernate. The user interface is realized via Java Server Pages. The PNEngine is running on a Tomcat server. It enables the users to perform an activity from the task list for actually processed cases, i.e. to fire enabled transitions of the correspondent copy of the net, via a web browser. After a user performs an activity from the task list for a given case, the corresponding transition is fired, the new marking is computed and the task list is updated. Different cases of the same process are able to communicate over static places.

3 PetriFlow PNEditor

PNEditor offers usual features of a graphical editor for designing place/transition nets. It enables to draw place/transition nets, i.e. labeled places, transitions, weighted arcs and markings with multiple tokens per place. The further functionality is saving the net to a file, the definition of roles, subnets (nested nets), saving of predefined subnets to files and their reuse as reusable components, replacement of subnets, definition of static places, which exists once per process, and other standard features, such as unlimited undo-redo actions.

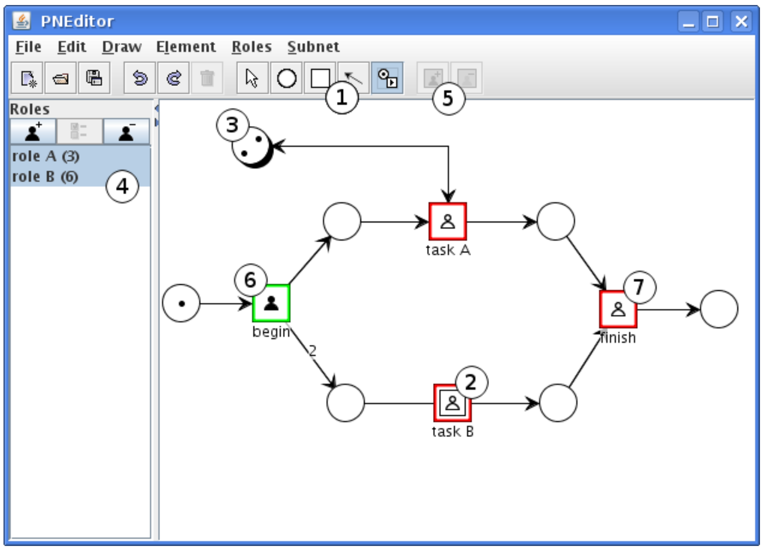


Fig. 1. Illustration of the key features of the PetriFlow PNEditor.

For a better illustration, Fig. 1 gives an overview of the functionality of the PetriFlow PNEditor module:

1. Drawing tool selection: from left: object selection tool, inserting places, inserting transitions, arc drawing, adding/removing tokens/transition firing
2. Square with double border represents a subnet
3. Place with shadow represents a static place
4. Panel with roles: buttons from left: add a role, edit role properties, delete a role.
   – role A contains set of transitions: begin, task A, finish (total of 3)
   – role B contains set of transitions: begin and all nested transitions in subnet task B (total of 6)
5. Buttons for adding or removing transitions from the currently selected roles
6. Both roles are selected so the information icons are displayed on top of the transitions in the diagram: black person icon on the transition describes the situation in which all the selected roles contain this transition
7. White person icon on the transition describes the situation in which only some selected roles contain this transition

4 Subnets in PetriFlow PNEditor

Usually, business modeller models a task as atomic transitions. On a more detailed level, typically a task can be started, finished, paused, continued or cancelled. Each of such tasks can be illustrated with a part of Petri net given in Fig. 2, which can be understood as a subnet on a more detailed level and should be used as a reusable component.

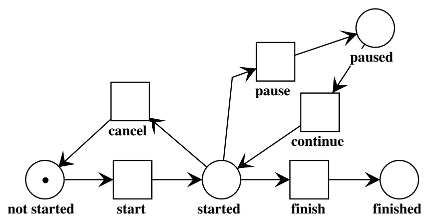


Fig. 2.

The place "not started" in Fig. 2 describes a state, in which the task has not yet been started and "finished" describes the state reached after the task is finished.

It would be very practical to model workflow processes using such reusable components and in general to give modeller an option to design and use his own reusable components possibly without any semantical restriction.

Another situation in which reusable components represents a desirable feature is in case of complex processes assembled from a relatively independent units with exactly defined inputs and outputs. The PNEditor supports subnets allowing each subnet to have more nested subnets so that the whole hierarchy can be build up in a place/transition net.

In case the user creates a workflow model where the tasks will be represented by transitions, PNEditor gives a choice of replacing a transition with a subnet. The subnets can be replaced by existing stored subnets. In this way, individual transitions can be converted to custom subnets representing reusable components.

In order to explain the concept of subnets, we have to recall some basic definitions of place/transition nets (for more details see e.g. [8]). Given a place/transition net $N = (P, T, F, W)$, where P is a finite set of places, T is a finite number of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs (i.e. the flow relation) and $W : F \to \mathbb{N}_0$ is the weight function ($\mathbb{N}_0$ denoting nonnegative integers). We say that a subnet of N is any net $N' = (P', T', F', W')$ where $P' \subseteq P$, $T \subseteq T$, the flow relation $F' = F \cap ((P' \times T') \cup (T' \times P'))$ and $W' = W|F'$. Moreover, we consider only proper subnets, i.e. subnets satisfying the following condition for each $p \in P$ and each $t \in T'$: $((p, t) \in F \vee (t, p) \in F) \Rightarrow p \in P'$. For clarity of the text let us define the interface of a proper subnet as the set of places $p \in P'$ which are connected with a transition which does not belong to

T'. In the PNEditor, the interface places are graphically expressed using dashed places. On one abstraction level, a subnet is visualized via interface places connected with a square with double border via reference edges. These edges can have one of two appearances:

1. dotted edge - the interface place is not connected with any transition in the subnet.
2. dashed edge - the interface place is connected in the subnet with one or more transitions

In case the interface place is connected in the subnet with exactly one transition, the reference edge takes the direction of the arc. Otherwise the reference edge is undirected, i.e. it is displayed without an arrow.

Neighbourhood of a place $p \in P$ w.r.t. the net N is a subnet $N_p = (\{p\}, T_p, F_p, W_p)$ of N with the set of places formed by the place itself and the set of transitions $T_p = \{t \in T | (t,p) \in F \vee (p,t) \in F\}$ formed by the the union of the preset and the postset of the place p in the net N, i.e. by surrounding transitions of p in net N.

Recall that two place/transition nets are isomorphic, when there exists a bijective mapping between the sets of places and a bijective mapping between the sets of transitions, which preserves arcs and their weights. We say that a place p in a place/transition net is said unique place of the net, if there is no place p' in the net with the isomorphic neighbourhood.

In the PNEditor, identities of the interface places are not saved when storing a subnet to make it a reusable component, i.e. a subnet is stored just as an ordinary net with an additional information which places form its interface. When replacing one subnet by another stored subnet, the interface places of the replaced subnet are identified with the interface places of the stored replacing net according to the following rules:

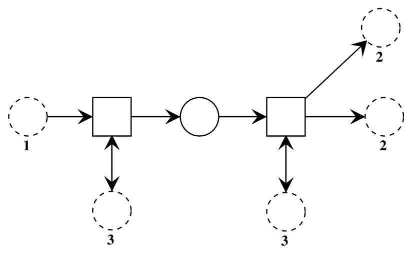


Fig. 3. Interface places labelled with the same number have isomorphic neighbourhood. Only the interface place labelled with number 1 is unique.

1. In the first place, only unique interface places are identified: A unique interface place p of the replaced subnet is considered to be equal to a unique

   interface place p' of the stored replacing subnet, if the neighbourhood of p w.r.t. the replaced subnet is isomorphic to the neighbourhood of p' w.r.t. the replacing stored net.
2. In the second step, if there exists exactly one unique interface place of the replaced subnet and exactly one unique interface place of the replacing stored net satisfying that their neighbourhoods w.r.t. the respective subnets are not isomorphic, then these interface places are considered to be equal. This correspond to a predicate, that if it is unambiguously possible, then the interface places should be identified.
3. Remaining interface places of replacing subnet replacing stored net are changed to ordinary places and remaining interface places of replaced net become interface places of the replacing net. It means that it is left to the user to identify manually by further editing which remaining interface places of the replaced subnet equal to the remaining interface places of the replacing net.

An example of the use of the subnet concept in the PNEditor is illustrated in Fig. 4:

1. The transition is created and converted to subnet
2. Visualization of the inside of the subnet
3. The subnet is modified and saved to a file
4. New subnet is created, selected command for replacing subnet
5. The result of 2 identical subnets

Thus, behind the visualization of a hierarchical process model in the PNEditor using the subnet concept is a single flat place/transition net.

5 Synthesis in PetriFlow PNEditor

We could design process models manually – which can be tedious and error-prone. There is also the possibility to collect logs from real-time processes and let an algorithm do the work for us. Workflow management systems such as the PNEngine can also be used to collect the logs. We just need simple p/t net with all transitions always enabled that will represent expecting activities.

There are multiple methods of Petri net synthesis already invented [9]. In the PNEditor we implemented a region based method *Separating feasible places* as described in [10].

5.1 Preliminaries

As usual we use the following notations. For details see [10].

An *alphabet* is a finite set A. The set of all *strings (words)* over an alphabet A is denoted by A^*. The *empty word* is denoted by λ. A subset $L \subseteq A^*$ is called *language over* A. For a word $w \in A^*$, $|w|$ denotes the *length of* w and $|w|_a$ denotes the number of occurrences of $a \in A$ in w. Given two words v, w, we

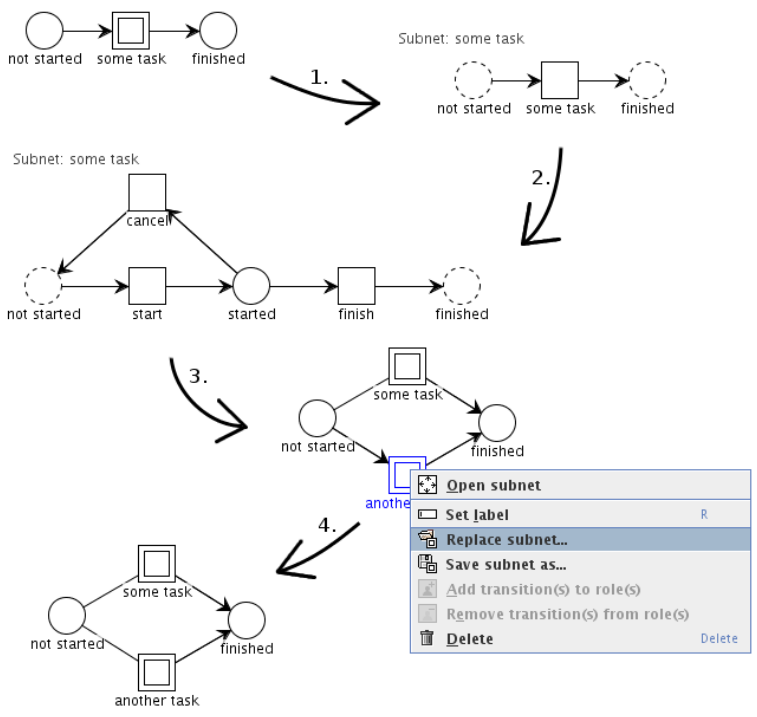


Fig. 4. Illustration of the subnet concept in PetriFlow PNEditor

call v *prefix* of w if there exists a word u such that $vu = w$. A language L is *prefix-closed*, if for every $w \in L$ each prefix of w also belongs to L.

Let T be a finite set of *activities* and C be a finite set of *cases*. And *event* is an element of $T \times C$. And *event log* is an element of $(T \times C)^*$.

Given a case $c \in C$ we define the function $p_c : T \times C \to T$ by $p_c(t, c') = t$ if $c = c'$ and $p_c(t, c') = \lambda$ else. Given an event log $\sigma = e_1 \ldots e_n \in (T \times C)^*$ we define the *process language* $L(\sigma)$ of σ by $L(\sigma) = \{p_c(e_1) \ldots p_c(e_i) | i \le n, c \in C\} \subseteq T^*$.

A net is a triple $N = (P, T, F)$, where P is a set of *places*, T is a finite set of *transitions* satisfying $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*. Let $x \in P \cup T$ be an element. The *preset* $\bullet x$ is the set $\{y \in P \cup T | (y, x) \in F\}$, and the *post-set* $x\bullet$ is the set $\{t \in P \cup T | (x, y) \in F\}$.

A *marking* of a p/t net $N = (P, T, F, W)$ is a function $m : P \to \mathbb{N}_0$ assigning $m(p)$ tokens to a place $p \in P$. A *marked p/t net* is a pair (N, m_0), where N is a p/t net, and m_0 is a marking of N, called *initial marking*.

A transition $t \in T$ is *enabled to occur* in a marking m of a p/t net N if $m(p) \ge W(p, t)$ for every place $p \in \bullet t$. If transition t is enabled to occur in a marking m, then its *occurrence* leads to the new marking m' defined by $m'(p) = m(p) - W(p, t) + W(t, p)$ for every place $p \in P$. That means t consumes $W(p, t)$ tokens from p and *produces* $W(t, p)$ tokens in p. We write $m \xrightarrow{t} m'$ to denote that t is enabled to occur in m and that its occurrence leads to m'. A finite sequence of transitions $w = t_1 \ldots t_n$, $n \in \mathbb{N}$ is called an *occurrence sequence enabled in m and leading to m_n* if there exists a sequence of markings $m_1, \ldots, m_n$ such that $m \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} m_n$. The set of all occurrence sequences enabled in the initial marking m_0 of a marked p/t net (N, m_0) forms a language over T and is denoted by $L(N, m_0)$.

Let (N, m_p), $N = (\{p\}, T, F_p, W_p)$ be a marked p/t net with only one place p (F_p, W_p, m_p are defined according to the definition of p). The place p is called *feasible* (w.r.t. $L(\sigma)$), if $L(\sigma) \subseteq L(N, m_p)$, otherwise *non-feasible*.

Denoting $T = \{t_1, \ldots, t_n\}$, a *region* of $L(\sigma)$ is a tuple $\mathbf{r} = (r_0, \ldots, r_{2n}) \in \mathbb{N}^{2n+1}$ satisfying for every $ct \in L(\sigma)$ ($c \in L(\sigma)$, $t \in T$):

$$r_0 + \sum_{i=1}^{n} (|c|_{t_i} \cdot r_i - |ct|_{t_i} \cdot r_{n+i}) \ge 0. \tag{1}$$

Every region $\mathbf{r}$ of $L(\sigma)$ defines a place p_r via $m_0(p_r) := r_0$, $W(t_i, p_r) := r_i$ and $W(p_r, t_i) := r_{n+i}$ for $1 \leqslant i \leqslant n$. The place p_r is called corresponding place to $\mathbf{r}$.

Given language L over T, $WC(L) = \{w \in L, t \in T : wt \notin L\}$ is called a set of wrong continuations of L over T.

Let $\mathbf{r}$ be a region of $L(\sigma)$ and let $WC \subseteq WC(L(\sigma))$ is a set of wrong continuations. The region $\mathbf{r}$ is a *separating region (w.r.t. WC)* if for every $wt \in WC$:

$$r_0 + \sum_{i=1}^{n} (|w|_{t_i} \cdot r_i - |wt|_{t_i} \cdot r_{n+i}) < 0. \tag{2}$$

A separating region $\mathbf{r}$ w.r.t. a set of wrong continuations $WC \subseteq WC(L(\sigma))$ can be calculated (if it exists) as a non-negative integer solution of a homogeneous linear inequation system with integer coefficients of the form

$$\mathbf{A}_{L(\sigma)} \cdot \mathbf{r} \geq \mathbf{0}$$
$$\mathbf{B}_{WC} \cdot \mathbf{r} < \mathbf{0}.$$

The matrix $\mathbf{A}_{L(\sigma)}$ consists of rows $\mathbf{a}_{ct} = (a_{ct,0}, \ldots, a_{ct,2n})$ for all $ct \in L(\sigma)$, satisfying $\mathbf{a}_{ct} \cdot \mathbf{r} \geq \mathbf{0} \Leftrightarrow$ (1). This is achieved by setting for each $ct \in L(\sigma)$:

$$a_{ct,i} = \begin{cases} 1 & \text{for } i = 0, \\ |c|_{t_i} & \text{for } i = 1, \ldots, n, \\ -|ct|_{t_{i-n}} & \text{for } i = n+1, \ldots, 2n. \end{cases}$$

The matrix $\mathbf{B}_{WC}$ consists of rows $\mathbf{b}_{wt} = (b_{wt,0}, \ldots, b_{wt,2n})$ for all $wt \in WC$, satisfying $\mathbf{b}_{wt} \cdot \mathbf{r} < \mathbf{0} \Leftrightarrow$ (2). This is achieved by setting for each $wt \in WC$:

$$b_{wt,i} = \begin{cases} 1 & \text{for } i = 0, \\ |w|_{t_i} & \text{for } i = 1, \ldots, n, \\ -|wt|_{t_{i-n}} & \text{for } i = n+1, \ldots, 2n. \end{cases}$$

The linear inequation system mentioned can be solved using linear programming ([11]) with linear objective function to minimize the resulting separating region, i.e. to generate minimal arc weights and a minimal initial marking.

5.2 Method of Separating Feasible Places

Given an event log σ with set of activities T we search for a preferably small finite marked p/t net (N, m_0) such that $L(\sigma) \subseteq L(N, m_0)$ and $L(N, m_0) \backslash L(\sigma)$ is small. According to the method of *Separating feasible places* we first create a p/t net with all transitions T but no arcs or places. This way, any occurrence sequence is enabled. Then we keep adding feasible places, until each wrong continuation of $WC(L(\sigma))$ is prohibited. Each feasible place is created according to one wrong continuation $wt \in WC(L(\sigma))$. We only calculate separating region w.r.t. $\{wt\}$ if wt is not already prohibited by already added places, because one place can prohibit multiple wrong continuations. For details see Algorithm 1.

5.3 Algorithm for Reducing the Number of Places

In some cases (see Fig. 5) we observed more than necessary number of places in the resulting net, so we created an algorithm for reducing the number of places in the resulting net. This algorithm is also implemented in the PNEditor.

Given a finite set A, the symbol $|A|$ denotes cardinality of A.

Our solution to this problem was to first identify which wrong continuations are prohibited by which places. Each place can prohibit multiple wrong continuations. This information is easy to get: we temporarily remove places $P' \subseteq P$ from the marked p/t net (N, m_0), $N = (P, T, F, W)$ and if the net permits given

```
input  : An event log $\sigma$
output: $(N, m_0)$, $N = (P, T, F, W)$ such that $L(\sigma) \subseteq L(N, m_0)$
$L(\sigma) \leftarrow$ process language of $\sigma$;
$A \leftarrow$ empty matrix;
$(P, T, F, W, m_0) \leftarrow (\emptyset$, activities of $\sigma, \emptyset, \emptyset, \emptyset)$;
foreach $w \in L(\sigma)$ do
    add row $a_w$ to matrix $A$;
end
foreach $w \in WC(L(\sigma))$ do
    if $w \in L(N, m_0)$ then
        $\mathbf{r} \leftarrow$ integer solution of $\mathbf{A} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{b}_w \cdot \mathbf{r} < 0, \mathbf{r} \geq \mathbf{0}$ such that $\mathbf{r}$ is minimal;
        if such solution $\mathbf{r}$ exist then
            $p \leftarrow$ corresponding place to $\mathbf{r}$;
            $P \leftarrow P \cup \{p\}$;
        end
    end
end
$coveredWrongContinuations \leftarrow \{w \in WC(L(N, m_0))\}$;
foreach $p \in P$ do
    $P \leftarrow P \setminus \{p\}$;
    $undo \leftarrow False$;
    foreach $w \in coveredWrongContinuations$ do
        if $w \in L(N, m_0)$ then
            $undo \leftarrow True$;
            break;
        end
    end
    if $undo$ then
        $P \leftarrow P \cup \{p\}$;
    end
end
```

Algorithm 1: The method of *Separating feasible places*: We add places that permit all correct continuations and prohibit at least one wrong continuation. In case a given wrong continuation is already prohibited by an already added place we do not need to create new one for the wrong continuation – it would be unnecessary. We slightly modified the existing algorithm – we moved the cleaning of unnecessary places after computing initial marked p/t net. See the original algorithm in [10]. If all wrong continuations are prohibited without given place then the place is unnecessary.

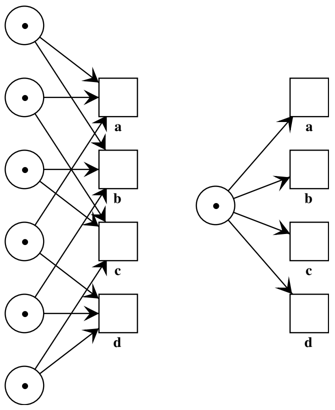


Fig. 5. On the left is a synthesized net using method of *Separating feasible places* and on the right is what we think an optimal solution.

wrong continuation $w \in WC(L(N, m_0))$ then we can say w is prohibited by the places P'.

The original method of *Separating feasible places* constructed each place from exactly one wrong continuation and all correct continuations. We will be constructing new places in the same way except we will be using one or more wrong continuations simultaneously.

First, we pick two places $p_1, p_2 \in P$. We determine which wrong continuations $WC_{p_1,p_2} \subseteq WC$ are prohibited by these places. Then we construct a new place p_3 from WC_{p_1,p_2} using the original method.

Now we compare what is "better": the two places p_1, p_2 or the one place p_3. We need to define what "better" actually is. We assumed that if the net has overall fewer places, fewer arcs then it is better. We decided to measure the complexity of the net $N = (P, T, F, W)$ as:

$$complexity(N) = |P| + \sum_{p \in P, t \in T} W(p,t) + W(t,p).$$

If the net has lower complexity without p_1, p_2 and with p_3, we replace p_1, p_2 with p_3, else we pick another pair of p_1, p_2 and repeat the cycle until we tried every combination.

When we make a replace, we run the algorithm again until no replace is made. The algorithm terminates because we have finite number of places.

We decided to test just two places at a time because we sought a fast algorithm. Testing every possible subset of the places would not be practical as it would have exponential time complexity according to the number of places.

Let N^* be a set of all possible p/t nets. Let *neighbour* be a function $N^* \times P \times P \to \{0, 1\}$. For a given p/t net $N = (P, T, F, W)$ and $p_1, p_2 \in P$ is $neighbour(N, p1, p2) = 1$ when $\exists t \in T : p_1 \in \bullet t \wedge p_2 \in \bullet t \vee p_1 \in t \bullet \wedge p_2 \in t \bullet$, otherwise $neighbour(N, p1, p2) = 0$.

Further we observed that each combination of places $p_1, p_2 \in P$, that were later merged to one place p_3, were in the same preset or post-set of some transition, i.e. $neighbour(N, p_1, p_2) = 1$. We used this observation to improve average case performance of the algorithm. Instead of testing each possible pair $p_1, p_2 \in P$, for each $p1 \in P$ we pick $p_2 \in P$ such that $neighbour(N, p_1, p_2) = 1$. For details see Algorithm 2.

Experimental results were positive (see Fig. 6 and Fig. 7).

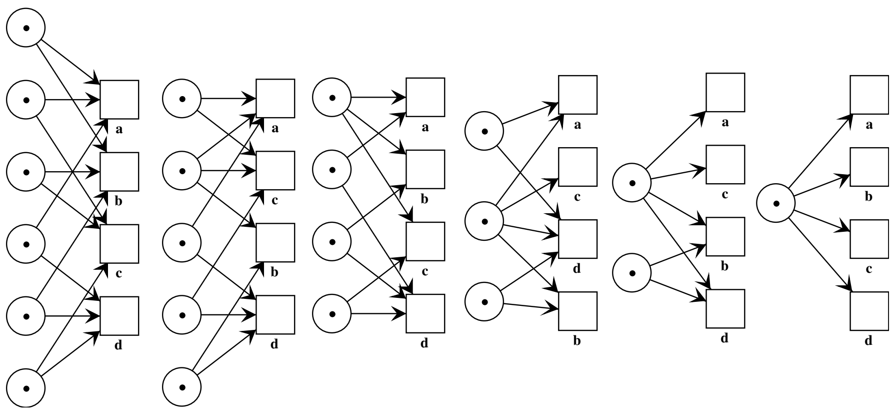


Fig. 6. Multiple steps of the algorithm for reducing complexity. On the left is input net and on the right is the output.

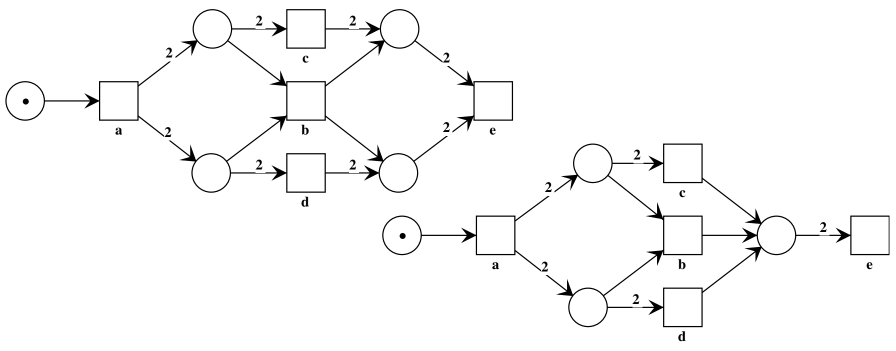


Fig. 7. On the left is synthesized net (N', m'_0) using method of *Separating feasible places* only. On the right is output (N, m_0) of our algorithm where input is the net on the left. Both nets have the same set of all occurrence sequences enabled in the initial marking, i. e. $L(N, m_0) = L(N', m'_0)$.

```
input  : Marked p/t net $(N', m_0'), N' = (P', T', F', W')$
output: Marked p/t net $(N, m_0), N = (P, T, F, W)$ such that
        $L(N, m_0) = L(N')$, $|P| \leq |P'|$ and $complexity(N) \leq complexity(N')$
$(P, T, F, W, m_0) \leftarrow (P', T', F', W', m_0')$;
$A \leftarrow$ empty matrix;
foreach $w \in L(N, m_0)$ do
    add row $a_w$ to matrix $A$;
end
$oldNumPlaces \leftarrow |P|$;
while $True$ do
    foreach $p1, p2 \in P : p1 \neq p2 \wedge neighbour(N, p1, p2)$ do
        $coveredWC \leftarrow \emptyset$;
        foreach $p \in \{p_1, p_2\}$ do
            $P \leftarrow P \setminus \{p\}$;
            foreach $w \in WC(L(N, m_0))$ do
                if $w \in L(N, m_0)$ then
                    $coveredWC \leftarrow coveredWC \cup \{w\}$;
                end
            end
            $P \leftarrow P \cup \{p\}$;
        end
        $B \leftarrow$ empty matrix;
        foreach $w \in coveredWC$ do
            add row $b_w$ to matrix $B$;
        end
        $\mathbf{r} \leftarrow$ integer solution of $\mathbf{A} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{B} \cdot \mathbf{r} < 0, \mathbf{r} \geq \mathbf{0}$ such that $\mathbf{r}$ is minimal;
        if such solution $\mathbf{r}$ exist then
            $oldComplexity \leftarrow complexity(N)$;
            $p_3 \leftarrow$ corresponding place to $\mathbf{r}$;
            $P \leftarrow P \setminus \{p_1, p_2\} \cup \{p_3\}$;
            $newComplexity \leftarrow complexity(N)$;
            $P \leftarrow P \cup \{p_1, p_2\} \setminus \{p_3\}$;
            if $newComplexity < oldComplexity$ then
                $P \leftarrow P \setminus \{p_1, p_2\}$;
                if $\sum_{t \in T}(W(p_3, t) + W(t, p_3)) > 0$ then
                    $P \leftarrow P \cup \{p_3\}$;
                end
                break;
            end
        end
    end
    if $|P| = oldNumPlaces$ then
        break;
    end
    $oldNumPlaces \leftarrow |P|$;
end
```

Algorithm 2: Algorithm for reducing complexity.

6 Conclusion

Although there are many methods for synthesis of Petri nets from logs (sequences, languages, partial languages, etc.), the main drawback when used in practice remains: the obtained nets are still too complicated in comparison with human made models. There are different reasons. Remember, that a net without places enables all sequences of transitions, and each place restricts behaviour by removing some sequences. Often, one reason for getting compicated models is that not each valid sequence is presented in the logs and therefore synthetised net obtain too much places restricting too much behaviour. Obviously, such cases cannot be solved by optimizing algorithms as presented in this paper. However, in the case that the logs are complete, optimization is a crucial step for acceptance of process mining in practice. The presented algorithm provides a simple step towards this direction. However, much of the work still has to be done in this area to nd the right mixture between accuracy of the resulting nets and their readability.

References

1. M. Beaudouin-Lafon, W. E. Mackay, M. Jensen, P. Andersen, P. Janecek, M. Lassen, K. Lund, K. Mortensen, S. Munck, A. Ratzer, K. Ravn, S. Christensen, K. Jensen: CPN/Tools: A Tool for Editing and Simulating Coloured Petri Nets *ETAPS Tool Demonstration Related to TACAS* In: *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2031, pp. 574–577, Springer-Verlag, 2001.
2. R. Bergenthum, J. Desel, G. Juhs, R. Lorenz: Can I Execute my Scenario in Your Net? VipTool tells you! In *Application and Theory of Petri Nets and Other Models of Concurrency.* LNCS 4024, pp. 381–390, Springer-Verlag, 2006.
3. J. Desel, G. Juhs, R. Lorenz and C. Neumair: Modelling and Validation with VipTool. In: *Business Process Management 2003*, LNCS 2678, pp. 380–389, Springer-Verlag, 2003.
4. K.M. van Hee, J. Keiren, R. Post, N. Sidorova, J.M. van der Werf: Designing case handling systems. In *Transactions on Petri Nets and Other Models of Concurrency I*, LNCS 5100, pp. 119–133, Springer, Berlin. 2008.
5. B. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, *Application and Theory of Petri Nets 2005, volume 3536 of Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
6. C.W. Gunther, W.M.P. van der Aalst: *Modeling the Case Handling Principles with Colored Petri Nets.* Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, October 2005, Department of Computer Science, University of Aarhus, PB-576, 211–230.
7. K. van Hee, N. Sidorova, M. Voorhoeve: *Resource-Constrained Workow Nets.* Fundamenta Informaticae, 2006.
8. W. Reisig: Petri nets: An introduction. Springer, 1985.
9. R. Bergenthum, J. Desel, S. Mauser: *Comparison of Different Algorithms to Synthesize a Petri Net from a Partial Language.* In: Lecture Notes in Computer Science: Transactions on Petri Nets and Other Models of Concurrency, pp. 216–243, 2009.

10. R. Bergenthum, J. Desel, R. Lorenz, S. Mauser: *Process Mining Based on Regions of Languages.* In: Lecture Notes in Computer Science: Business Process Management, pp. 375–383, 2007.
11. R. J. Vanderbei: *Linear Programming: Foundations and Extensions.* Kluwer Academic Publishers, 1996.

A Tool for the Synthesis of Asynchronous Speed-Independent Circuits

Ondrej Gallo, Tomáš Nečas, Fedor Lehocki

Faculty of Electrical Engineering and Information Technology,
Slovak University of Technology,
Ilkovičova 3, 812 19 Bratislava, Slovak Republic
ondrej.gallo@stuba.sk, xnecas@is.stuba.sk, fedor.lehocki@stuba.sk

Abstract. The present work is devoted to the development of software tool written in Java for synthesis of asynchronous speed-independent circuits. A special type of Petri nets - Signal Transition Graph, was used for the synthesis. Using the algorithm based on the theory of regions, a logic function is derived from this graph. In order to reduce the complexity of the resulting asynchronous circuit the number of gates should be minimized by optimization of logical function with a Quine-McCluskey algorithm.

Keywords: Synthesis Asynchronous Speed-Independent Circuit, Signal Transition Graph, State Graph.

1 Introduction

Asynchronous circuits have gained in importance along with the expansion of the production of high integration circuits. By that time, mostly synchronous systems were designed. Decreasing the dimensions and increasing the operating frequency, however, made the synchronisation of individual function blocks on a chip more difficult. The time delay is generally caused by the increased cycle of timing signal. This delay (for the individual function blocks) differs locally to such a degree that it results in the synchronisation failure of the individual subsystems and the malfunction of the circuit. This issue can be solved by adding a special regulation circuit providing a constant clock frequency in the whole chip. Such a circuit would occupy relatively much space on a chip (approx. 10%) and consume much power (approximately 40% of the input power). This would result in the increasing cost and energy demand of such chips. The application of asynchronous circuits provides a different approach. These circuits do not need a clock signal because their operation is controlled by events and not by time. As no clock signal generation is required and no regulation circuitry is needed, such circuits are smaller and consume less energy. The only issue that is common for both synchronous and asynchronous circuits is the presence of so-called hazardous states.

The aim of this work is to provide a tool for the synthesis of asynchronous circuits that generates a circuit diagram as a result of the circuit behavior. The field of digital circuit synthesis is complex and provides many solution approaches. One of them is represented by the application of the Petri nets formalism as a description tool for the behavior of an arbitrary digital circuit. Application of algorithms for STG synthesis (e.g. based on the region theory) a logical function is derived (using Quine-McCluskey algorithm) and presented in the form of a circuit diagram.

2 Basic definitions

The signal transition graph (STG) [1] is a specially labelled Petri net that is defined as a heptad (P, T, F, M_0, N, s_0, λ_T), where P is set of places, T is the set of transitions, F is the flow relation $F \subseteq (P \times T) \cup (T \times P)$, M_0 is the initial marking of Petri net, $N = I \cup O$ is a set of signals, I is a set of input signals and O is a set of output signals. s_0 represents the initial value for each signal in its initial state and λ_T: $T \rightarrow N \times \{+, -\}$ is the transition labelling function.

The state graph SG [1] is basically a reachability graph for the STG. Compared with the reachability graph, states in the state graph are labelled more functionally. It can be designed only in the case that the reachability graph is bounded, i.e. the number of reachable labels is finite. This graph in its final form is being used for the derivation of logical functions in the synthesis of asynchronous circuits. The state graph is formally defined as a triplet (S, δ, λ_S). S is a set of all the states, $\delta \subseteq S \times T \times S$ is a set of state transitions and $\lambda_S : S \rightarrow (N \rightarrow \{0,1\})$ is a function of state labelling.

Each state is labelled by a binary vector ($s(0), s(1), ..., s(n)$), where each signal s(i) $i \in \{0, 1, ... n\}$ can acquire values from the set $\{0, 1\}$. Although for a better illustration of the graphical representation of possible states, individual signals can also acquire possible values from the set {0, 1, R, F}. This form of labelling also provides information as to whether the respective signal is excited or not. The excited signal represents a change in its value from 0 to 1 or vice versa. This is illustrated by the characters R or F. The R character denotes the signal value in the SG being equal to 0 but where its value has changed to 1 in the subsequent state s(i). The signal is able to reach the next state because the respective transition could be started. This transition is labelled u_i+ by using the labelling function $\lambda_T(t)$. Similarly, the F character denotes the signal value being equal to 1 and to 0 in the subsequent state. The excited state can be formally defined as follows:

$$\exists(s_i, t, s_j) \in \delta \,.\, \lambda_T(t) = u_i+ \vee \lambda_T(t) = u_i-. \tag{1}$$

Each signal is labelled in the STG as a transition representing the front edge of the signal u_i+ or the decay of the signal u_i–. As stated hereinabove, STG is a special Petri net fulfilling the following properties:

Input free-choice: The starting sequence of the respective transition (signal) is controlled by the so-called mutual exclusion of input signals. It is indicated by a special transition in the transition graph.

Boundedness: This property of the transition graph provides that the state graph (to be defined later in the text) shall acquire a finite number of states. The transition graph is single-bounded when just a single label is in each place. The transition graph must be a safe Petri net.

Liveness: The STG must be free from deadlocks.

State consistency: All transitions providing front edge and a decay of the signal must strictly alter between + and – in any execution of the STG.

Complete state coding (CSC): This property is checked in SG. This means that a pair of states of *S* has a unique state coding defined by the labelling function λ_S, or it does not have the unique state coding but it does contain the same output signal excited in each state. If this property is not fulfilled, a new signal or signals are to be inserted into the SG.

Persistency: If a transition is enabled, it is fired and the label is transferred from the place ahead of the transition to the place or places behind, provided that this start shall not be deactivated by another transition. This property must be provided for the input and internal signals. The persistency of the input signals must be ensured by the environment of the designed circuit.

3 Software description

The model of the circuit behavior represents an input for our tool (ACDesigner [2]). In the process of the circuit's design, designers mostly prefer the timing diagram of the circuit's behavior. In this diagram, all input and output signals, and causalities between the signals, are recorded. Thus if a timing diagram is available a special Petri net (Signal Transition Graph) can be formed, representing the input for our software. By means of the algorithm presented by Cortadella et al. [3], we derive an asynchronous speed–independent circuit whose behavior is characterized by the STG. Speed independence is a property ensuring the correct circuit operation considering that all logical gates have unbounded delay [1]. The graphical representation of the circuit synthesis process is shown in Fig. 1.

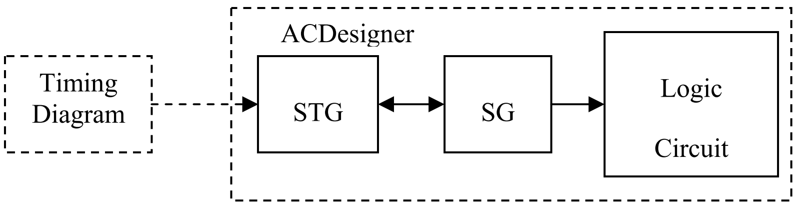


Fig. 1. Graphical representation of the particular steps in synthesis.

After loading STG in the software, the required properties (boundedness, consistency, persistency, input free-choice, liveness, and complete state coding) are verified and the SG is derived. In some cases, the state graph does not comply with the CSC property. The solution of this issue is based on the insertion of new states into the SG according to the algorithm published in [4] and [5]. An example of the SG fulfilling the CSC (new signal *csc0*) property is shown in Fig. 2b. SG is an intermediate product in the process of synthesis and it is not visualised in our tool.

The state graph is divided into regions and intersections of some regions. The best regions covering conflicts are selected (for each iteration only one conflict is selected). After selection of the conflict the environment is added (consisting of neighbor regions) and the cost function is calculated [4]. This function serves as a basis to determine direction of the SG search to find the most suitable place for the insertion of the new signal. After reduction of the number of possible insertion points, the best one is selected for which the least complicated circuit is formed. This iteration should also be repeated several times, because single signal insertion may not solve all the conflicts, and new ones could even appear. The algorithm, however, converges to a solution that was confirmed experimentally in the reference [4]. If SG fulfils the CSC property, a logic function is derived for each output and new (internal) signal. The minimisation of the logic function, which is required with respect to the resulting number of gates, is the next step. The Quine-McCluskey algorithm [6] was applied in our software to minimise the logic function. At this level, requirements can also be laid on the application of particular gate types. In our case, we focused on the use of the standard 2-input gates of AND, OR type and on the NOT gate.

A Petri net in PNML format [7], which can be designed by using other tools, e.g. PNEditor [8] or VipTool [9], is the input file in this ACDesigner version. The net must contain a label indentifying the signal type (input or output signal). In a Petri net, signals are represented by transitions. Therefore, they must be labelled as the input or output signals. In Fig. 2a, a demonstration of the Petri net is shown including the labelling of transitions with the keywords "in" and "out". The name is arbitrary provided that it ends either with the + or − sign. For the designer, it is an indication of the signal change from 0 to 1 (+ sign) or from 1 to 0 (− sign), respectively.

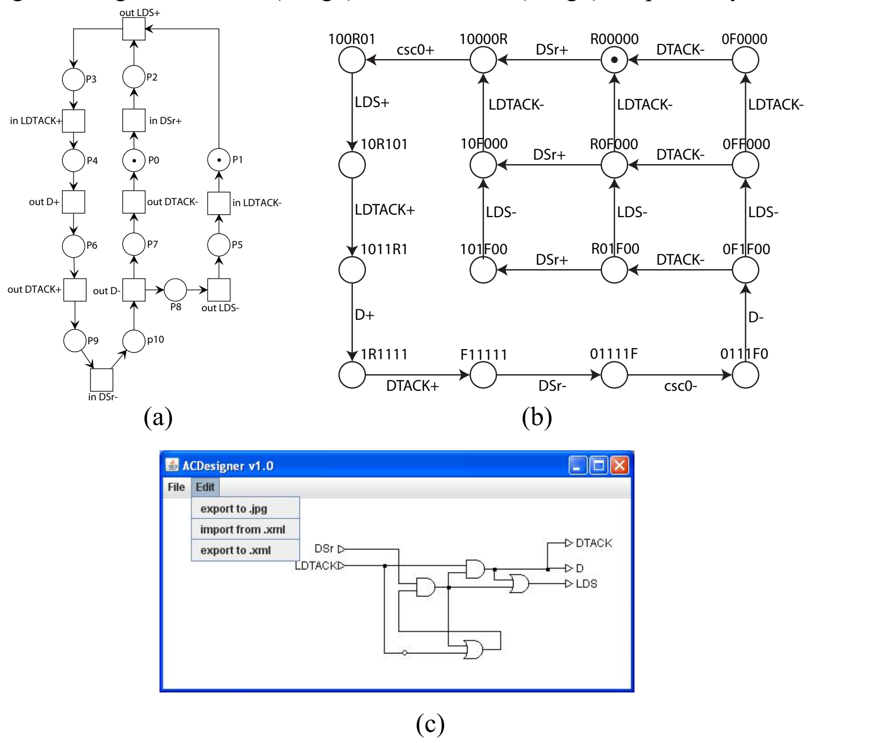


Fig. 2. (a) Input format of Petri nets - STG, (b) SG fulfilling the CSC property, binary vector is <*DSr*, *DTACK*, *LDTACK*, *LDS*, *D, csc0*>, (c) resulting logical circuit.

The software output is shown in Fig. 2c. The user can save the generated circuit in two formats, either as XML or as JPG.

4 Conclusion

The algorithm of the synthesis of asynchronous speed-independent circuits was implemented for the first time in the petrify tool [10]. Due to the absence of the graphical visualisation of the resulting circuit, we decided to programme our own tool, which would include this functionality. The software is written in Java ver. 1.6, which ensures platform independence. The implementation of more methods of logical circuits' synthesis will be the next step. These methods should take advantage of a non-standard memory element (C-element). The utilization of this element may prevent the occurrence of several hazardous states. Moreover, it has a very fast memory and can be easily implemented on a chip. The extension of support to multiple input and output formats is another important step in our development. One of the possible input formats could be the timing diagram, which is easy understandable to many designers. If a logical circuit is designed, it must be verified by the simulation process. This option is provided by another professional simulation software (e.g. Protel, PSPICE, and CADENCE) that requires its own specific file format. From that point of view, our tool will be extended with respective additional functionalities.

References

1. Chris J. Myers, Asynchronous Circuit Design, John Wiley & Sons, July, 2001.
2. ACDesigner: A tool for synthsis of asynchronous circuits: http://gordan.matmas.net/acdesigner/
3. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, Hardware and Petri Nets: Application to Asynchronous Circuit Design, Application and Theory of Petri Nets 2000, Lecture Notes in Computer Science vol. 1825, pp. 1-15, Springer Verlag, June 2000.
4. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, A region-based theory for state assignment in speed-independent circuits, IEEE Trans. on CAD, vol. 16, no. 8, August 1997, pp. 793-812.
5. J. Cortadella, M. Kishinevsky, L. Lavagno and A. Yakovlev. Deriving Petri nets from finite transition systems. Universitat Politecnica de Catalunya, Tech. Rep. UPC-DAC-96-19, June 1996.
6. Brian Holdsworth, R. Clive Woods, Digital logic design 4th edition, Newnes, 2002.
7. The Petri Net Markup Language: www.informatik.hu-berlin.de/top/pnml/
8. PNeditor: A tool for modeling Petri nets. https://pneditor.matmas.net/
9. VipTool: A tool for modeling Petri nets. http://viptool.ku-eichstaett.de
10. Petrify: A tool for synthesis of Petri nets and asynchronous controllers. http://www.lsi.upc.edu/~jordicf/petrify.

Petri Nets and Software Engineering (PNSE)

Introduction

For the successful realisation of complex systems of interacting and reactive software and hardware components the use of a precise language at different stages of the development process is of crucial importance. Petri nets are becoming increasingly popular in this area, as they provide a uniform language supporting the tasks of modelling, validation, and verification. Their popularity is due to the fact that Petri nets capture fundamental aspects of causality, concurrency and choice in a natural and mathematically precise way without compromising readability.

The use of Petri nets (P/T-nets, coloured Petri nets and extensions) in the formal process of software engineering, covering modelling, validation, and verification, is presented as well as their application and tools supporting the disciplines mentioned above.

This part contains contributions accepted for long and short presentation at the International Workshop on *Petri Nets and Software Engineering* (PNSE'10) in Braga, Portugal, June 22, 2010.

We received 16 high-quality contributions. The program committee has accepted four of them for full presentation. Furthermore the committee accepted five papers as short presentations. Three contributions were submitted and accepted as posters.

This CEUR proceedings version comprises just the full and short presentation papers. More information about the workshop, like the full online-proceedings, can be found at http://www.informatik.uni-hamburg.de/TGI/events/pnse10/

The program committee consisted of:
Wil van der Aalst (Eindhoven University, The Netherlands)
João Paulo Barros (Instituto Politécnico de Beja, Portugal)
Didier Buchs (University of Geneva, Switzerland)
Piotr Chrząstowski-Wachtel (University of Warsaw, Poland)
Gianfranco Ciardo (University of California at Riverside, USA)
José-Manuel Colom (University of Zaragoza, Spain)
Jörg Desel (Catholic University Eichstätt-Ingolstadt, Germany)
Raymond Devillers (Université Libre de Bruxelles, Belgium)
Marlon Dumas (University of Tartu, Estonia)
Michael Duvigneau (University of Hamburg, Germany) (Chair)
Berndt Farwer (University of Durham, UK)
João M. Fernandes (Universidade de Minho, Portugal)
Jorge C. A. de Figueiredo (Federal University de Campina Grande, Brazil)
Giuliana Franceschinis (University of Piemonte Orientale / University of Torino, Italy)
Guy Gallasch (University of South Australia, Australia)
Luís Gomes (Universidade Nova de Lisboa, Portugal)
Nicolas Guelfi (University of Luxembourg, Luxembourg)
Stefan Haar (ENS Cachan, France)
Xudong He (Florida International University, USA)
Thomas Hildebrandt (University of Copenhagen, Denmark)
Vladimir Janousek (University of Brno, Czech Republic)
Gabriel Juhás (Slovak University of Technology Bratislava, Slovakia)
Peter Kemper (College of William and Mary, USA)
Astrid Kiehn (Indraprastha Institute of Information Technology Delhi, India)
Ekkart Kindler (Technical University of Denmark, Denmark)
Hanna Klaudel (Université d'Evry-Val d'Essonne, France)
Michael Köhler-Bußmeier (University of Hamburg, Germany)

The international program committee was supported by the valued work of Michał Knapik, Levi Lucio, Elisabetta Mangioni and Tarek Melliti as additional reviewers. Their work is highly appreciated.

Furthermore, we would like to thank the organizational teams of the Universidade do Minho and the Instituto Politécnico de Beja, Portugal, for their general organizational support.

Without the enormous efforts of authors, reviewers, PC members and the organizational teams this workshop wouldn't provide such an interesting collection of contribution.

Thanks!

Michael Duvigneau
Daniel Moldt

Combining Petri Nets and UML for Model-based Software Engineering

João M. Fernandes

Dep. Informática / Centro Algoritmi
Universidade do Minho
4710-057 Braga
Portugal

Abstract

UML is by far the most widely used modelling language used nowadays in software engineering, due to its large scope and its wide tool support. This software standard offers many diagrams that cover all typical perspectives for describing and modelling the software systems under consideration. Among those diagrams, UML includes diagrams (activity diagram, state machine diagram, use case diagrams, and the interaction diagrams) for describing the behaviour (or functionality) of a software system. Petri nets constitute a well-proven formal modelling language, suitable for describing the behaviour of systems with characteristics like concurrency, distribution, resource sharing, and synchronisation. Thus, one may question why not combining some UML diagrams with Petri nets for effectively supporting the activities of the software engineer. The usage of Petri nets for/in Software Engineering was addressed by several well-known researchers, like, for example, Reisig [6], Pezzè [1], Machado [5], and Kindler [4].

In this invited paper, we discuss some alternatives to introduce Petri nets into a UML-based software development process. In particular, we describe how Coloured Petri Net (CPN) models can be used to describe the set of scenarios associated with a given use case. We describe three different alternatives that can be adopted to achieve that purpose.

The first approach, initially presented in [7], suggests a set of rules that allow software engineers to transform the behaviour described by a UML 2.0 sequence diagram into a CPN model. Sequence diagrams in UML 2.0 are much richer than those in UML 1.x, namely by allowing several traces to be combined in a unique diagram, using high-level operators over interactions. The main purpose of the transformation is to allow the development team to construct animations based on the CPN model that can be shown to the users or the clients in order to reproduce the expected scenarios and thus validate them. Thus, non-technical stakeholders are able to discuss and validate the captured requirements. The usage of animation is an important topic in this context, since it permits the user to discuss the system behaviour using the problem domain language.

In the second approach, discussed in [3], we assume that developers specify the functionality of the system under consideration with use cases, each of which is described by a set of UML 2.0 sequence diagrams. For each use case, there should exist at least one sequence diagram that represents and describes its main scenario. Other sequence diagrams for the same use case are considered to be variations of the main sce-

nario. The transformation approach allows the development team to interactively play or reproduce any possible run of the given scenarios. In particular, the natural characteristics of the CPN modelling language facilitate the representation of the hierarchy and concurrency constructs of sequence diagrams.

The third alternative, considered in [2], is an improvement with respect to the previous approach and is targeted to reactive systems. We identify and justify two key properties that the CPN model must have, namely: (1) controller-and-environment-partitioned, which means constituting a description of both the controller and the environment, and distinguishing between these two domains and between desired and assumed behaviour; (2) use case-based, which means constructed on the basis of a given use case diagram and reproducing the behaviour described in accompanying scenario descriptions. We have demonstrated how this CPN model is useful for requirements engineering, since it provides a solid basis for addressing behavioural issues early in the development process, for example regarding concurrent execution of use cases and handling of failures.

References

1. G. Denaro and M. Pezzè. Petri Nets and Software Engineering. In *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 439–466. Springer, 2004. DOI 10.1007/b98282.
2. J.M. Fernandes, J.B. Jørgensen, and S. Tjell. Requirements engineering for reactive systems: Coloured petri nets for an elevator controller. In *14th Asia-Pacific Software Engineering Conference (APSEC 2007)*, pages 294–301. IEEE CS Press, December 2007. DOI 10.1109/APSEC.2007.81.
3. J.M. Fernandes, S. Tjell, J.B. Jørgensen, and O. Ribeiro. Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net. In *6th Int. Workshop on Scenarios and State Machines (SCESM 2007), within ICSE 2007*. IEEE CS Press, May 2007. DOI 10.1109/SCESM.2007.1.
4. Ekkart Kindler. Model-Based Software Engineering and Process-Aware Information Systems. *Transactions on Petri Nets and Other Models of Concurrency*, 5460:27–45, 2009. DOI 10.1007/978-3-642-00899-3_2.
5. R. J. Machado, K. B. Lassen, S. Oliveira, M. Couto, and P. Pinto. Requirements Validation: Execution of UML Models with CPN Tools. *International Journal on Software Tools for Technology Transfer*, 9(3–4):353–369, 2007. DOI 10.1007/s10009-007-0035-0.
6. W. Reisig. Petri Nets in Software Engineering. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 63–96. Springer, 1987. DOI 10.1007/3-540-17906-2_22.
7. O. Ribeiro and J. M. Fernandes. Some Rules to Transform Sequence Diagrams into Coloured Petri Nets. In K. Jensen, editor, *7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN 2006)*, pages 237–256, October 2006.

The Resource Allocation Problem in Software Applications: A Petri Net Perspective*

Juan-Pablo López-Grao[1] and José-Manuel Colom[2]

[1] Dpt. of Computer Science and Systems Engineering (DIIS)
[2] Aragonese Engineering Research Institute (I3A)
University of Zaragoza, Spain
Email: {jpablo,jm}@unizar.es

Abstract. Resource Allocation Systems (RAS) have been intensively studied in the last years in the domain of Flexible Manufacturing Systems (FMS). The success of this research line has been based on the identification of particular subclasses of Petri Nets that correspond to a RAS abstraction of this kind of systems. In this paper we take a parallel road to that travelled through for FMS, but for the case of software applications. The considered applications present concurrency and deadlocks can happen due to the allocation of shared resources. We will evince that the existing subclasses of Petri Nets used to study this kind of deadlock problems are insufficient, even for very simple software systems. From this starting point we propose a new subclass of Petri Nets that generalizes the previously known RAS subclasses and we present a taxonomy of anomalies that can be found in the context of software systems.

1 Introduction

Among the most recurrent patterns in a wide disparity of engineering disciplines, the competition for shared resources between concurrent processes takes a prominent position. The reader might think of examples in the context of distributed systems, operations research, manufacturing plants, etc. The perspective of discrete event systems theory proves appropriate and powerful as a framework in which provide solutions to the so-called resource allocation problem [1]. Systems of this kind are often called Resource Allocation Systems (RAS) [2, 3].

RAS are usually conceptualized around two distinct entities, processes and resources, thanks to a prior abstraction process which is inherent in the discipline. The resource allocation problem refers to satisfying successfully the requests for resources made by the processes, ensuring that no process ever falls in a deadlock. A set of processes is deadlocked when they indefinitely wait for resources that are already held by other processes of the same set [4].

RAS can be categorized both on the type of processes (sequential, non-sequential) and resources (serially reusable, consumable) [5]. Hereafter, we will

* This work has been partially supported by the European Community's Seventh Framework Programme under Project DISC (Grant Agreement n. INFSO-ICT-224498) and the project CICYT-FEDER DPI2006-15390.

focus on Sequential RAS with serially reusable resources. This means that a process can increase or decrease the quantity of free resources during its execution. However, the process will contervail that operation before terminating, i.e. resources are used in a conservative way.

Although other models of concurrency have also been considered [6], Petri nets [7] have arguably taken a leading role among the family of formal models used for dealing with the resource allocation problem [8, 9]. One of the strengths of this approach is the smooth mapping between the main entities of RAS and the basic elements of Petri net models. A resource type can be modelled using a place: the number of instances of it being modelled with tokens. Meanwhile, sequential processes are modelled with tokens progressing through state machines. Arcs from resource places to transitions (from transitions to resource places) represent the acquisition (return) of some resources by a process. Petri nets thus provide a natural formal framework for the analysis of RAS, besides benefiting from the goods of compositionality.

This fact is well notorious in the domain of Flexible Manufacturing Systems (FMS), where Petri net models for RAS have widely succeeded since the seminal work of Ezpeleta et al. was introduced [8]. This is mostly due to a careful selection of the subclass of Petri nets used to model these FMS, based upon two solid pillars. First, the definition of a rich syntax from a physical point of view, which enables the natural expression of a wide disparity of plant configurations. And second, the contribution of sound scientific results which let us characterize deadlocks from the model structure, as well as provide a well-defined methodology to automatically correct them in the real system.

Nowadays, there exists a plethora of Petri net models for modelling RAS in the context of FMS, which often overcome some of the syntactical limitations of the S^3PR class [8]. S^4PR net models [10, 11] generalize the earlier, while allowing multiple simultaneous allocations of resources per process. S^*PR nets [12] extend the expressive power of the processes to that of state machines: hence internal cycles in their control flow is allowed. However, deadlocks in S^*PR net models are not fully comprehended from a structural perspective. Other classes such as NS-RAP [9], ERCN-merged nets [13] or PNR nets [14] extend the capabilities of S^3PR/S^4PR models beyond Sequential RAS by way of lot splitting or merging operations.

Most analysis and control techniques in the literature are based on the computation of a structural element which univocally characterizes deadlocks in many RAS models: the so-called *bad siphon*. A bad siphon is a siphon which is not the support of a p-semiflow. If bad siphons become (sufficiently) emptied, their output transitions die since the resource places of the siphon cannot regain tokens anymore, thus revealing the *deadly embrace*. Control techniques thus rely on the insertion of monitor places [15], i.e. controllers in the real system, which limit the leakage of tokens from the bad siphons.

Although there exist obvious resemblances between the resource allocation problem in FMS and that of parallel or concurrent software, previous attempts to bring these well-known RAS techniques into the field of software engineering have been, to the best of our knowledge, either too limiting or unsuccessful. Gadara nets [16] constitute the most recent attempt, yet they fall in the over-restrictive side in the way the resources can be used, as a result of inheriting the design philosophy applied for FMS. In this work, we will analyze why the net classes and results introduced in the context of FMS fail when brought to the field of concurrent programming.

Section 2 presents a motivating example and discusses the elements that a RAS net model should desirably feature in order to successfully explore the resource allocation problem within the software enginering discipline. Taking into account those considerations, section 3 introduces a new Petri net class, called PC^{2}R. Section 4 relates the new class to those defined in previous works and establishes useful net transformations which forewarn us about new behavioural phenomena. Section 5 introduces some of these anomalies which highlight the fact that previous theoretical results in the context of FMS are insufficient in the new framework. Finally, section 6 summarizes the results of the paper.

2 The RAS view of a software application

Example 1 presents a humorous variation of Dijkstra's classic problem of the dining philosophers. We will adopt and adapt the beautiful writing by Hoare at [17] for its enunciation.

Example 1. The pragmatic dining philosophers. "Five philosophers spend their lives thinking and eating. The philosophers share a common dining room where there is a circular table surrounded by five chairs, each belonging to one philosopher. A microwave oven is also available. In the center of the table there is a large bowl of spaghetti which is frequently refilled (so it cannot be emptied), and the table is laid with five forks. On feeling hungry, a philosopher enters the dining room, sits in his own chair, and picks up the fork on the left of his place. Then he touches the bowl to feel its temperature. If he feels the spaghetti got too cold, he will leave his fork and take the bowl to the microwave. Once it is warm enough, he will come back to the table, sit on his chair and leave the bowl on the table after recovering his left fork (please bear in mind that the philosopher is *really* hungry by now). Unfortunately, the spaghetti is so tangled that he needs to pick up and use the fork on his right as well. If he can do it before the bowl gets cold again, he will serve himself and start eating. When he has finished, he puts down both forks and leaves the room."

According to the classic RAS nomenclature, each philosopher is a sequential process, and the five forks plus the bowl are serially reusable resources which are shared among the five processes. From a software perspective, each philosopher can be a process or a thread which will be executed concurrently.

Algorithm 1 introduces the code for each philosopher. Notationally, we modelled the acquisition / release of resources by way of the `wait()` / `signal()` operations, respectively. Both of them have been generalized for the acquisition of multiple resources (separated by commas when invoking the function). Finally, the `trywait()` operation is a non-blocking wait operation. If every resource is available at the time `trywait()` is invoked, then it will acquire them and return `TRUE`. Otherwise, `trywait()` will return `FALSE` without acquiring any resource. For the sake of simplicity, it is assumed that the conditions with two or more literals are evaluated atomically.

Algorithm 1 - Code for Philosopher i (where $i \in \{1, 2, 3, 4, 5\}$)

```
      var
           fork: array [1..5] of semaphores; // shared resources
           bowl: semaphore;                  // shared resource
      begin
        do while (1)
           THINK;
           Enter the room;
(T1)       wait(fork[i]);
           do while (not(trywait(bowl, fork[i+1 mod 5]))
                   or the spaghetti is cold)
(T2)         if (trywait(bowl)
                   and the spaghetti is cold) then
(T3)            signal(fork[i]);
                Go to the microwave;
                Heat up spaghetti;
                Go back to table;
(T4)            wait(fork[i]);
(T5)            signal(bowl);
             end if;
(T6)       loop;
           Serve spaghetti;
(T7)       signal(bowl);
           EAT;
(T8)       signal(fork[i], fork[i+1 mod 5]);
           Leave the room;
        loop;
```

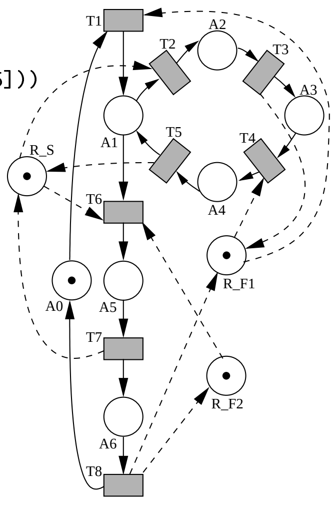


Fig. 1. Philosopher 1.

Figure 1 depicts the net for algorithm 1, with $i = 1$, after abstracting the relevant information from a RAS perspective. Figure 2 renders the composition of the five philosopher nets via fusion of the common shared resources. Note that if we remove the dashed arcs from figure 2, then we can see five disjoint strongly connected state machines plus six isolated places.

The five state machines represent the control flow for each philosopher. Every state machine is composed of seven states (each state being represented by a place). Tokens in a state machine represent concurrent processes/threads which share the same control flow. In this case, the unique token in each machine is located at the so-called *idle place*. This means that, at the initial state, every philosopher is thinking (outside the room). In general, the idle place can be seen as a mechanism which enforces a structural bound: the number of concurrent *active threads* (i.e. non-idle) is limited. Here, at most one philosopher of type i can be inside the room, for each $i \in \{1, 2, 3, 4, 5\}$.

The six isolated places are called *resource places*. A resource place represents a certain resource type, and the number of tokens in it represents the quantity of free instances of that resource type. In this case, every resource place is monomarked. Thus, at the initial state there is one fork of type i, for every $i \in \{1, 2, 3, 4, 5\}$, plus one bowl of spaghetti (modelled by way of the resource place at the centre of the figure).

Finally, the dashed arcs represent the acquisition or release of resources by the active threads when they change their execution state. Every time a transition is fired, the total amount of resources available is altered. Please note, however, that moving one isolated token of a state machine (by firing its transitions) until the token reaches back the idle state, leaves the resource places marking unaltered. Thus, the resource usage is conservative.

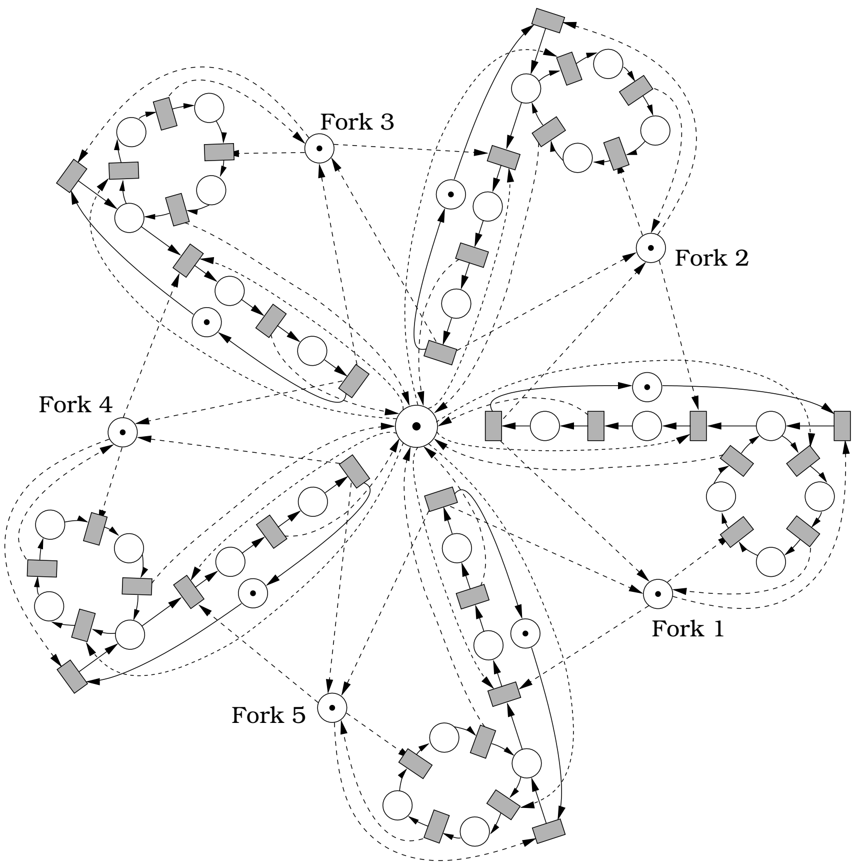


Fig. 2. The dining philosophers are thinking. Arcs from/to P_R are dashed for clarity.

At this point, we will discuss some capabilities that (in our humble opinion) a RAS model should have so as to support the modelling of concurrent programs.

Although acyclic sequential state machines are rather versatile as models for sequential processes in the context of FMS (as the success of the S^3PR and S^4PR classes prove), this is clearly too constraining even for very simple software systems. Considering Böhm and Jacopini's theorem [18], however, we can assume that every non-structured sequential program can be refactored into a structured one using `while-do` loops. Meanwhile, calls to procedures and functions can be substituted by way of inlining techniques. Let us also remind that `fork`/`join` operations can also be unfolded into isolated concurrent sequential processes, as evidenced in [9]. As a result, we can restrict process models to state machines in which decisions and iterations (in the form of `while-do` loops) are supported, but not necessarily every kind of internal cycle.

Another significant difference between FMS and software systems from a RAS perspective is that resources in the latter are not necessarily physical (e.g., a file) but can also be logical (e.g., a semaphore). This has strong implications in the degree of freedom allowed for allocating those resources: we will return to this issue a little later.

In this domain, a resource is an object that is shared among concurrent processes/threads and must be used in mutual exclusion. Since the number of resources is limited, the processes will compete for the resource and will use it in a non-preemptive way. This particular allocation scheme can be imposed by the resources' own access primitives, which may be blocking. Otherwise, the resource can be protected by a binary semaphore/mutex/lock (if there is only one instance of that resource type) or by a counting semaphore (multiple instances). Note that this kind of resources can be of assorted nature (e.g., shared memory locations, storage space, database table rows) but the required synchronization scheme is inherently similar.

On the other side, it is well-known that semaphores used in that aim can be also seen as non-preemptive resources which are used in a conservative way. For instance, a counting semaphore that limits the number of connections to a database can be interpreted in that way from a RAS point of view. Here processes will wait for the semaphore when attempting to establish a database connection, and will release it when they decide to close the aforementioned connection.

However, semaphores also perform a relevant role as an interprocess signaling facility, which can also be a source of deadlocks. In this work, our goal is the study of the resource allocation problem, so this functionality is out of scope. We propose fixing deadlock problems due to resource allocation issues firstly, and later apply other techniques for amending those due to message passing.

Due to their versatility, semaphore primitives are interesting for studying how resources can be allocated by a process/thread. For instance, XSI semaphores (also known as System V semaphores) have a multiple wait primitive (`semop` with `sem_op<0`). An example of multiple resource allocation appears in algorithm 1. Besides, an XSI semaphore can be decremented atomically in more than one. Both POSIX semaphores (through `sem_trywait`) and XSI semaphores (through `semop` with `sem_op<0` and `sem_flag=IPC_NOWAIT`) have a non-blocking wait primitive. Again, algorithm 1 could serve as an example. Finally, XSI semaphores also feature inhibition mechanisms (through `semop` with `sem_op=0`), i.e. processes can wait for a zero value of the semaphore.

As we suggested earlier, the fact that resources in software engineering do not always have a physical counterpart is a very peculiar characteristic with consequences. In this context, processes do not only consume resources but also can *create* them. A process will destroy the newly created resources before its termination. For instance, a process can create a shared memory variable (or a service!) which can be allocated to other processes/threads. Hence the resource allocation scheme is no longer *first-acquire-later-release*, but it can be the other way round too. Nevertheless, all the resources will be used in a conservative way by the processes (either by a create-destroy sequence or by a wait-release sequence). As a side effect, and perhaps counterintuitively, there may not be free resources during the system startup (as they still must be created), yet being the system live.

Summing up, for successfully modelling RAS in the context of software engineering, a Petri net model should have at least the following abstract properties:

1. The control flow of the processes should be represented by state machines with support for decisions (`if-then-else` blocks) and nested internal cycles (`while-do` blocks).
2. There can be several resource types and multiple instances of each one.
3. State machines can have multiple tokens (representing concurrent threads).
4. Processes/threads use resources in a conservative way
5. Acquisition/release arcs can have non-ordinary weights (e.g., a semaphore value can be atomically incremented/decremented in more than one unit)
6. Atomic multiple acquisition/release operations must be allowed
7. Processes can have decisions dependent of the allocation state of resources (due to the non-blocking wait primitives, as in figure 2)
8. Processes can lend resources. As a side effect, there could exist processes that depend on resources which must be created/lent by other processes (hence they cannot finish if executed in isolation)

3 PC^{2}R nets

In this section, we will present a new Petri net class, which fulfills the requirements advanced in section 2: the class of Processes Competing for Conservative Resources (PC^{2}R). This class generalizes other subclasses of the S^{n}PR family while respecting the design philosophy on these. Hence, previous results are still valid in the new framework. However, PC^{2}R nets can deal with more complex scenarios which were not yet addressed from the domain of S^{n}PR nets.

Definition 1 presents a subclass of state machines which is used for modelling the control flow of the processes in isolation. Iterations are allowed, as well as decisions within internal cycles, in such a way that the control flow of structured programs can be fully supported. Non-structured processes can still be refactored into them as discussed in Section 2.

Definition 1. *An* iterative state machine $\mathcal{N} = \langle \{p_0\} \cup P, T, C\rangle$ *is a strongly connected state machine such that either every cycle contains* p_0 *or* P *can be partitioned into two subsets* P_1, P_2*, with a place* $p \in P_2$ *such that:*

1. *The subnet generated by* $\langle \{p\} \cup P_1, {}^\bullet P_1 \cup P_1{}^\bullet\rangle$ *is a strongly connected state machine in which every cycle contains* p*, and*
2. *The subnet generated by* $\langle \{p_0\} \cup P_2, {}^\bullet P_2 \cup P_2{}^\bullet\rangle$ *is an iterative state machine.*

In figure 1, if we remove the resource places R_F1, R_F2 and R_S then we obtain an iterative state machine, with $P_1 = \{A2, A3, A4\}$, $P_2 = \{A1, A5, A6\}$, $p_0 = A0$ and $p = A1$. The definition of iterative state machines is instrumental for introducing the class of PC^2R nets.

PC^2R nets are modular models. Two PC^2R nets can be composed into a new PC^2R model via fusion of the common shared resources. Please note that a PC^2R net can simply be one process modelled by an iterative state machine along with the set of resources it uses. Hence the whole net model can be seen as a composition of the modules for each process. We will formally define the class in the following:

Definition 2. *Let* $I_\mathcal{N}$ *be a finite set of indices. A* PC^2R *is a connected generalized pure P/T net* $\mathcal{N} = \langle P, T, C\rangle$ *where:*

1. $P = P_0 \cup P_S \cup P_R$ *is a partition such that: (a)* [idle places] $P_0 = \{p_{0_1}, ..., p_{0_{|I_\mathcal{N}|}}\}$*; (b)* [process places] $P_S = P_1 \cup ... \cup P_{|I_\mathcal{N}|}$*, where* $\forall i \in I_\mathcal{N}$*:* $P_i \neq \emptyset$ *and* $\forall i, j \in I_\mathcal{N}$*:* $i \neq j$*,* $P_i \cap P_j = \emptyset$*; (c)* [resource places] $P_R = \{r_1, ..., r_n\}, n > 0$.
2. $T = T_1 \cup ... \cup T_{|I_\mathcal{N}|}$*, where* $\forall i \in I_\mathcal{N}, T_i \neq \emptyset$*, and* $\forall i, j \in I_\mathcal{N}, i \neq j, T_i \cap T_j = \emptyset$.
3. *For all* $i \in I_\mathcal{N}$ *the subnet generated by restricting* $\mathcal{N}$ *to* $\langle \{p_{0_i}\} \cup P_i, T_i\rangle$ *is an iterative state machine.*
4. *For each* $r \in P_R$*, there exists a unique minimal p-semiflow associated to* r*,* $Y_r \in \mathbb{N}^{|P|}$*, fulfilling:* $\{r\} = \|Y_r\| \cap P_R, (P_0 \cup P_S) \cap \|Y_r\| \neq \emptyset$*, and* $Y_r[r] = 1$.
5. $P_S = \bigcup_{r \in P_R}(\|Y_r\| \setminus \{r\})$.

Please note that the support of the Y_r p-semiflows (point 4 of definition 2) may include P_0: this is new with respect to S^4PR nets. Such a resource place r is called a *lender* resource place. If r is a lender, then there exists a process which creates (*lends*) instances of r. In our model, processes can start their execution creating resource instances, but *before* acquiring any other resource. Otherwise, it could happen that the support of a minimal p-semiflow would contain more than one resource place (thus infriging condition 4 of definition 2).

The class supports iterative processes, multiple resource acquisitions, non-blocking wait operations and resource lending. Inhibition mechanisms are not natively supported (although some cases can still be modelled with $\mathrm{PC^2R}$ nets).

The next definition generalizes the notion of acceptable initial marking introduced for the $\mathrm{S^4PR}$ class. In software systems all processes/threads are initially inactive and start from the same point (the `begin` operation). Hence, all of the corresponding tokens are in the idle place in the initial marking (the process places being therefore empty). Note that lender resource places may be empty for an acceptable initial marking. Figure 2 shows a P^2CR net with an acceptable initial marking which does not belong to the S^4PR class.

Definition 3. *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, C\rangle$ be a PC^{2}R. An initial marking m_0 is acceptable for $\mathcal{N}$ iff $||m_0|| = P_0 \cup P_R$ and $\forall p \in P_S, r \in P_R\ :\ Y_r^T \cdot m_0 \geq Y_r[p]$.*

4 Some transformations and related classes

In [19], we introduced a new class of Petri net models for RAS, called SPQR (Systems of Processes Quarreling over Resources). SPQR nets feature an appealing syntactical simplicity and expressive power though they are very challenging from an analytical point of view. They can be roughly described as RAS nets in which the process subnets are acyclic and the processes can lend resources in any possible (conservative) manner. Every PC^{2}R can be transformed into a Structurally Bounded SPQR net (SB SPQR net).

The transformation rule is based on the idea of converting every while-do block in an acyclic process which is activated by a lender resource place. This lender place gets marked once the thread reaches the while-do block. The token is removed at the exit of the iteration. This transformation must be applied starting by the most intern loops, proceeding in decreasing nesting order. Figure 3 depicts the transformation rule. The rule preserves the language accepted by the net (and thus liveness) since it basically consists in the addition of a implicit place (place $P1$ in the right hand net of figure 3, since R_P1 can be seen as a renaming of $P1$ in the left hand net).

Figure 4 illustrates the transformation of the net of example 1 but restricted to two philosophers into the corresponding SB SPQR.

Thanks to such transformations, the SB SPQR class can express the widest range of systems in the Sequential RAS Petri net family. Figure 5 introduces the inclusion relations between a variety of Petri net classes for Sequential RAS.

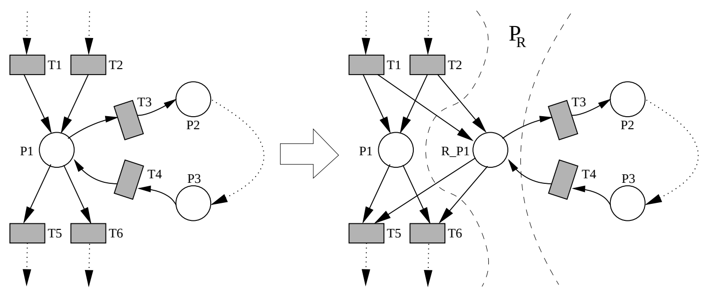


Fig. 3. Transforming PC^{2}Rs into SB SPQRs: From iterative to acyclic processes

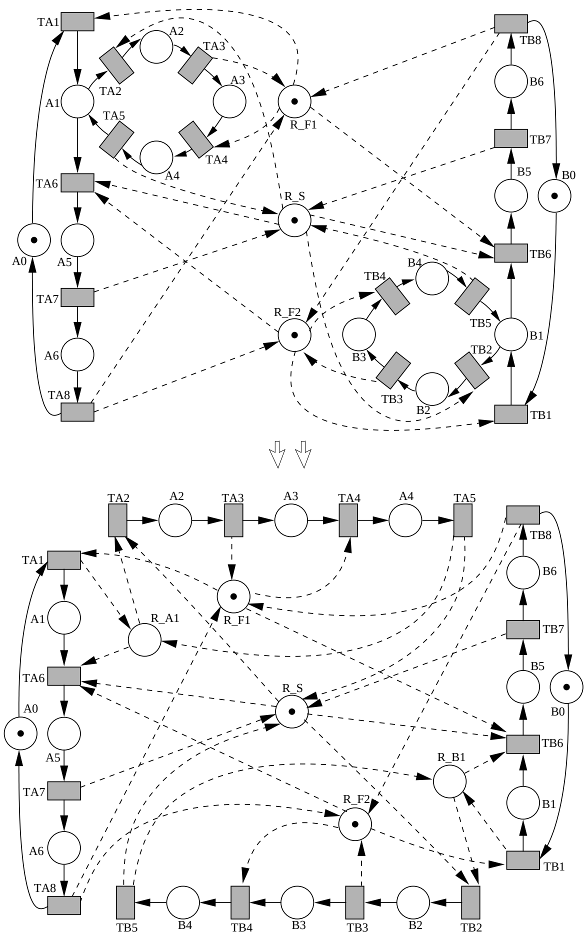


Fig. 4. From PC^{2}R to SB SPQR: Two pragmatic dining philosophers

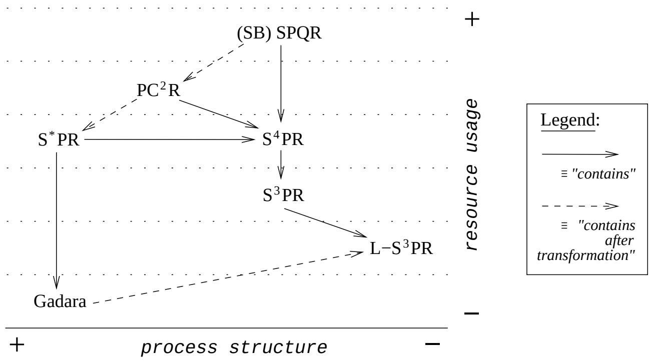


Fig. 5. Inclusion relations between Petri net classes for RAS

5 Some bad properties through examples

The bad news about the discussion in sections 2 and 3 is that siphon-based control techniques for RAS do not work in general for concurrent software, even ignoring (i.e., not using) the *resource lending* feature introduced by PC^{2}R nets.

Let us have a look back at example 1 and its related algorithm 1. It is not difficult to see that, if every philosopher enters the room, sits down and picks up the fork on the left of himself, the philosophers will be trapped in a livelock. Every philosopher can eventually take the bowl of spaghetti and heat it up in the microwave. This pattern can be repeated infinitely, but it is completely useless, since no philosopher will ever be able to have dinner.

This behaviour is obviously reflected in the corresponding net representation at figure 2. Let us construct a firing sequence σ containing only the first transition of each state machine (i.e., the output transition of its idle place). The firing order of these transitions is irrelevant. Now let us fire such a sequence, and the net falls in a livelock. The internal cycles are still firable in isolation, but no idle place can ever be marked again. Unfortunately, the net has several bad siphons, but none of them is empty or insufficiently marked in the livelock. In other words, for every reachable marking in the livelock, there exist output transitions of the siphons which are firable. As a result, the siphon-based non-liveness characterization for earlier net classes (such as S^{4}PR [10]) is not sufficient in the new framework.

A similar pattern can be observed in the upper net of figure 4. There exist three bad siphons, which are $D_1 = \{A2, A3, A4, A5, A6, B2, B4, B5, B6, R\_F2, R\_S\}$, $D_2 = \{A2, A4, A5, A6, B2, B3, B4, B5, B6, R\_F1, R\_S\}$ and $D_3 = \{A2, A4, A5, A6, B2, B4, B5, B6, R\_F1, R\_F2, R\_S\}$. Besides, every transition in the set $\Omega = \{TA2, TA3, TA4, TA5, TB2, TB3, TB4, TB5\}$ is an output transition of D_1, D_2 and D_3. After firing $TA1$ and $TB1$ from the initial marking, the state $A1 + B1 + R\_S$ is reached. This marking belongs to a livelock with other six markings. The reader can check that, unfortunately, there exists a firable transition in Ω for every marking in the livelock. A similar phenomenon can be observed for the SB SPQR net at the bottom of figure 4.

In general, livelocks are not a new phenomenon in the context of Petri net models for RAS. Even for $L - S^3PR$ nets, which are the simplest models in the family, deadlock freeness does not imply liveness [20]. However, deadlocks and livelocks always could be related to the existence of a siphon which was 'dry'. Unfortunately, this no longer holds. Another well-known result for simpler subclasses was that liveness equalled reversibility for nets with acceptable initial markings. For PC^{2}R, this is also also untrue, as figure 6 proves.

We believe that the transformation of PC^{2}R nets into SB SPQR can be useful to understand the phenomena from a structural point of view. Intuitively speaking, the concept of *lender resource* seems a simple yet powerful instrument which still remains to be fully explored. Still, SB SPQRs can present very complex behaviour. For instance, acceptably marked SB SPQR nets do not even hold the directness property [21] (which e.g. was true for S^{4}PR nets). Figure 7 shows a marked net which has no home states in spite of being live. This and

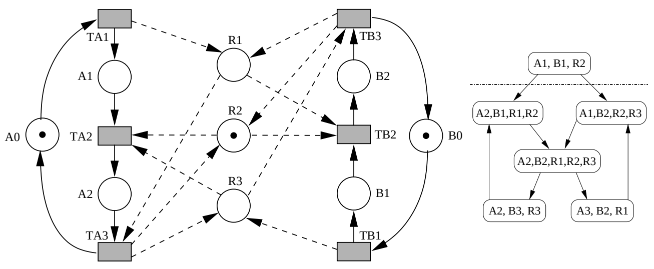


Fig. 6. An acceptably marked PC^{2}R which is live but not reversible

other properties are profoundly discussed (along with their implications) in a previous work [19].

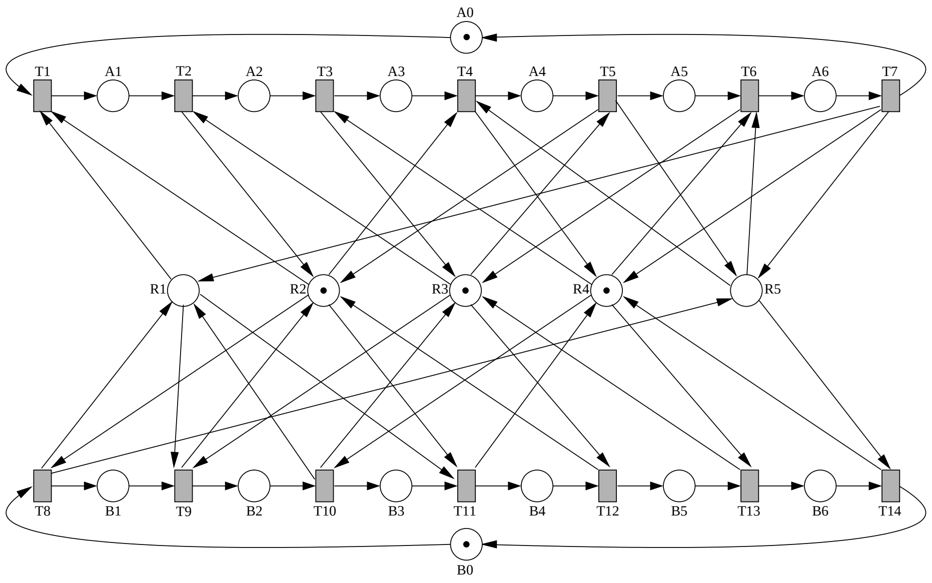


Fig. 7. A marked SB SPQR which is live but has no home states

6 Conclusion and future work

Although there exist a variety of Petri net classes for RAS, many of these definition efforts have been directed to obtain powerful theoretical results for the analysis and synthesis of this kind of systems. Nevertheless, we believe that the process of abstraction is a central issue in order to have useful models from a real-world point of view, and therefore requires careful attention. In this work, we have followed that path and constructed a requirements list for obtaining an interesting Petri net subclass of RAS models applied to the software engineering domain. Considering that list, we defined the class of PC^{2}R nets, which fulfills those requirements while respecting the design philosophy on the RAS view of systems. We also introduced some useful transformation and class relations so as to locate the new class among the myriad of previous models. Finally we observed that the problem of liveness in the new context is non-trivial and presented some cases of bad behaviour which will be subject of subsequent work.

A Petri Nets: Basic definitions

A *place/transition net* (P/T net) is a 3-tuple $\mathcal{N} = \langle P, T, W\rangle$, where W is a total function $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$, being P, T non empty, finite and disjoint sets. Elements belonging to the sets P and T are called respectively *places* and *transitions*, or generally nodes. P/T nets can be represented as a directed bipartite graph, where places (transitions) are graphically denoted by circles (rectangles): let $p \in P$, $t \in T$, $u = W(p,t)$, $v = W(t,p)$, there is a directed arc, labelled u (v), beginning in p (t) and ending in t (p) iff $u \neq 0$ $(v \neq 0)$.

The *preset* (*poset*) or set of input (output) nodes of a node $x \in P \cup T$ is denoted by ${}^\bullet x$ $(x^\bullet)$, where ${}^\bullet x = \{y \in P \cup T \mid W(y,x) \neq 0\}$ $(x^\bullet = \{y \in P \cup T \mid W(x,y) \neq 0\})$. The preset (poset) of a set of nodes $X \subseteq P \cup T$ is denoted by ${}^\bullet X$ $(X^\bullet)$, where ${}^\bullet X = \{y \mid y \in {}^\bullet x, x \in X\}$ $(X^\bullet = \{y \mid y \in x^\bullet, x \in X\}$

An *ordinary P/T net* is a net with unitary arc weights (i.e., W can be defined as a total function $(P \times T) \cup (T \times P) \rightarrow \{0,1\}$). If the arc weights can be non-unitary, the P/T net is also called *generalized*. A *state machine* is an ordinary net such that for every transition $t \in T$, $|{}^\bullet t| = |t^\bullet| = 1$. An *acyclic state machine* is an ordinary net such that for every transition $t \in T$, $|{}^\bullet t|, |t^\bullet| \leq 1$, and there is no circuit in it.

A self-loop place $p \in P$ is a place such that $p \in p^{\bullet\bullet}$. A *pure P/T net* (also self-loop free P/T net) is a net with no self-loop places. In pure P/T nets, the net can be also defined by the 3-tuple $\mathcal{N} = \langle P, T, C\rangle$, where C is called the *incidence matrix*, $C[p,t] = W(p,t) - W(t,p)$. Nets with self-loop places can be easily transformed into pure P/T nets without altering most significant behavioural properties, such as liveness, as shown in figure 8.

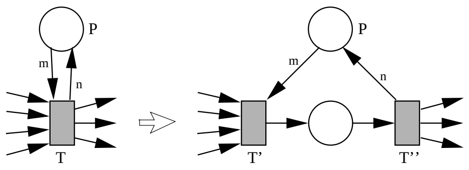


Fig. 8. Removing self-loop places

A *p-flow* is a vector $Y \in \mathbb{Z}^{|P|}$, $Y \neq \mathbf{0}$, which is a left annuler of the incidence matrix, $Y \cdot C = \mathbf{0}$. The support of a p-flow is denoted $\|Y\|$, and its places are said to be covered by Y. A *p-semiflow* is a non-negative p-flow, i.e. a p-flow such that $Y \in \mathbb{N}^{|P|}$. The P/T net $\mathcal{N}$ is *conservative* iff every place is covered by a p-semiflow. A *minimal p-semiflow* is a p-semiflow such that the g.c.d of its non-null components is one and its support $\|Y\|$ is not an strict superset of the support of another p-semiflow.

A set of places $D \subseteq P$ is a *siphon* iff every place $p \in {}^{\bullet}D$ holds $p \in D^{\bullet}$. The support of a p-semiflow is a siphon but the opposite does not hold in general.

Let $\mathcal{N} = \langle P, T, W \rangle$ be a P/T net, and let $P' \subseteq P$ and $T' \subseteq T$, where $P', T' \neq \emptyset$. The P/T net $\mathcal{N}' = \langle P', T', W' \rangle$ is the subnet generated by P', T' iff $W'(x, y) \Leftrightarrow W(x, y)$, for every pair of nodes $x, y \in P' \cup T'$.

A *marking* m of a P/T net $\mathcal{N}$ is a vector $\mathbb{N}^{|P|}$, assigning a finite number of marks $m[p]$ (called *tokens*) to every place $p \in P$. Tokens are represented by black dots within the places. The *support* of a marking, $\|m\|$, is the set of places which are marked in m, i.e. $\|m\| = \{p \in P \mid m[p] \neq 0\}$. We define a *marked P/T net* (also P/T net system) as the pair $\langle \mathcal{N}, m_0 \rangle$, where $\mathcal{N}$ is a P/T net, and m_0 is a marking for $\mathcal{N}$, also called *initial marking*. $\mathcal{N}$ is said to be the structure of the system, while m_0 represents the system state.

Let $\langle \mathcal{N}, m_0 \rangle$ be a marked P/T net. A transition $t \in T$ is *enabled* (also *firable*) iff $\forall p \in {}^{\bullet}t \ : \ m_0[p] \geq W(p, t)$, which is denoted by $m_0[t\rangle$. The *firing* of an enabled transition $t \in T$ changes the system state to $\langle \mathcal{N}, m_1 \rangle$, where $\forall p \in P \ : \ m_1[p] = m_0[p] + C[p, t]$, and is denoted by $m_0[t\rangle m_1$. A *firing sequence* σ from $\langle \mathcal{N}, m_0 \rangle$ is a non-empty sequence of transitions $\sigma = t_1\, t_2 \ldots t_k$ such that $m_0[t_1\rangle m_1[t_2\rangle \ldots m_{k-1}[t_k\rangle$. The firing of σ is denoted by $m_0[\sigma\rangle t_k$. A marking m is *reachable* from $\langle \mathcal{N}, m_0 \rangle$ iff there exists a firing sequence σ such that $m_0[\sigma\rangle m$. The *reachability set* $RS(\mathcal{N}, m_0)$ is the set of reachable markings, i.e. $RS(\mathcal{N}, m_0) = \{m \mid \exists\, \sigma \ : \ m_0[\sigma\rangle m\}$.

A transition $t \in T$ is *live* iff for every reachable marking $m \in RS(\mathcal{N}, m_0)$, $\exists m' \in RS(\mathcal{N}, m)$ such that $m'[t\rangle$. The system $\langle \mathcal{N}, m_0 \rangle$ is *live* iff every transition is live. Otherwise, $\langle \mathcal{N}, m_0 \rangle$ is *non-live*. A transition $t \in T$ is *dead* iff there is no reachable marking $m \in RS(\mathcal{N}, m_0)$ such that $m[t\rangle$. The system $\langle \mathcal{N}, m_0 \rangle$ is a *total deadlock* iff every transition is dead, i.e. no transition is firable. A *home state* m_k is a marking such that it is reachable from every reachable marking, i.e. $\forall m \in RS(\mathcal{N}, m_0) \ : \ m_k \in RS(\mathcal{N}, m)$. The net system $\langle \mathcal{N}, m_0 \rangle$ is *reversible* iff m_0 is a home state.

References

1. Lautenbach, K., Thiagarajan, P.S.: Analysis of a resource allocation problem using Petri nets. In Syre, J.C., ed.: Proc. of the 1st European Conf. on Parallel and Distributed Processing, Toulouse, Cepadues Editions (1979) 260–266
2. Colom, J.M.: The resource allocation problem in flexible manufacturing systems. In van der Aalst, W-M-P. and Best, E., ed.: Proc. of the 24th Int. Conf. on Applications and Theory of Petri Nets. Volume 2679 of LNCS., Eindhoven, Netherlands, Springer–Verlag (June 2003) 23–35

3. Li, Z.W., Zhou, M.C.: Deadlock Resolution in Automated Manufacturing Systems: A Novel Petri Net Approach. Springer, New York, USA (2009)
4. Coffman, E.G., Elphick, M., Shoshani, A.: System deadlocks. ACM Computing Surveys **3**(2) (1971) 67–78
5. Reveliotis, S.A., Lawley, M.A., Ferreira, P.M.: Polynomial complexity deadlock avoidance policies for sequential resource allocation systems. IEEE Transactions on Automatic Control **42**(10) (1997) 1344–1357
6. Fanti, M.P., Maione, B., Mascolo, S., Turchiano, B.: Event-based feedback control for deadlock avoidance in flexible production systems. IEEE Transactions on Robotics and Automation **13**(3) (1997) 347–363
7. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (1989) 541–580
8. Ezpeleta, J., Colom, J.M., Martínez, J.: A Petri net based deadlock prevention policy for flexible manufacturing systems. IEEE Transactions on Robotics and Automation **11**(2) (April 1995) 173–184
9. Ezpeleta, J., Recalde, L.: A deadlock avoidance approach for non–sequential resource allocation systems. IEEE Transactions on Systems, Man and Cybernetics. Part–A: Systems and Humans **34**(1) (January 2004)
10. Tricas, F., García-Valles, F., Colom, J.M., Ezpeleta, J.: A Petri net structure-based deadlock prevention solution for sequential resource allocation systems. In: Proc. of the 2005 Int. Conf. on Robotics and Automation (ICRA), Barcelona, Spain, IEEE (April 2005) 272–278
11. Park, J., Reveliotis, S.A.: Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. IEEE Transactions on Automatic Control **46**(10) (2001) 1572–1583
12. Ezpeleta, J., Tricas, F., García-Vallés, F., Colom, J.M.: A banker's solution for deadlock avoidance in FMS with flexible routing and multiresource states. IEEE Transactions on Robotics and Automation **18**(4) (August 2002) 621–625
13. Xie, X., Jeng, M.D.: ERCN-merged nets and their analysis using siphons. IEEE Transactions on Robotics and Automation **29**(4) (1999) 692–703
14. Jeng, M.D., Xie, X.L., Peng, M.Y.: Process nets with resources for manufacturing modeling and their analysis. IEEE Transactions on Robotics **18**(6) (2002) 875–889
15. Hu, H.S., Zhou, M.C., Li, Z.W.: Liveness enforcing supervision of video streaming systems using non-sequential Petri nets. IEEE Transactions on Multimedia **11**(8) (December 2009) 1446–1456
16. Wang, Y., Liao, H., Reveliotis, S., Kelly, T., Mahlke, S., Lafortune, S.: Gadara nets: Modeling and analyzing lock allocation for deadlock avoidance in multithreaded software. In: Proc. of the 49th IEEE Conf. on Decision and Control, Atlanta, Georgia, USA, IEEE (December 2009) 4971–4976
17. Hoare, C.A.R.: Communicating sequential processes. Communications of the ACM **21**(8) (1978) 666–677
18. Harel, D.: On folk theorems. Communications of the ACM **23**(7) (1980) 379–389
19. López-Grao, J.P., Colom, J.M.: Lender processes competing for shared resources: Beyond the S^{4}PR paradigm. In: Proc. of the 2006 Int. Conf. on Systems, Man and Cybernetics, IEEE (October 2006) 3052–3059
20. García-Vallés, F.: Contributions to the structural and symbolic analysis of place/transition nets with applications to flexible manufacturing systems and asynchronous circuits. PhD thesis, University of Zaragoza, Zaragoza (April 1999)
21. Best, E., Voss, K.: Free choice systems have home states. Acta Informatica 21 (1984) 89–100

IRS-MT: Tool for Intelligent Resource Allocation.

Piotr Chrząstowski-Wachtel[1,2] and Jakub Rauch[1]

[1] Institute of Informatics, Warsaw University,
Banacha 2, PL 02-097 Warszawa, Poland
[2] Warsaw School of Social Sciences and Humanities,
Chodakowska 18/31, PL 03-815 Warszawa, Poland
`pch@mimuw.edu.pl, jakub.rauch@gmail.com`

Abstract. A tool for optimizing cost and time of a workflow execution with respect to allocation of multi-purpose resources is presented. The optimization is done as a result of simulations, which take into account the cost and time associated with each of the resources, when allocated to transitions in Petri nets representing a workflow. These attributes can be collected and updated based on the logs of the finished instances. The program suggests the best allocation procedures, giving the estimates of the performance of the whole run for all possible decisions.

1 Introduction

Modeling processes as workflows has become quite popular and proved its usefulness in practice. One of the problems associated with running a workflow is the resource management. By resources we mean all components required to run an activity. In our case, their necessity is described by certain requirements (e.g. skills or functions) associated with activities (transitions). With each resource we associate a bunch of skills, so we can choose, which of the resources are to be attached to certain activity at the workflow simulation run-time. The Petri net approach requires collecting all the resources necessary for a given transition before triggering an acitivity to run. One cannot reserve a resource, and keep it busy, while waiting for other resources necessary to run an activity. Only when all resources are available we can make a decision to run (fire) an activity.

It often happens that a resource is requested by different activities. A resource conflict occurs, when a single resource is shared by two enabled transitions. We must make a decision, which activity will use the resource first. We assume here that the resources are re-usable, and that after finshing a transition the resource can be used by another activity.

When we make a decision about the resource allocation, we should take into account several aspects. Usually we try to optimize some quality function, like time or cost of the workflow run. We assume here that during a workflow run we can perform several actions concurrently. Since some of them will be competing for resources, a proper allocation can improve the quality of workflow run. Engaging proper resources can diminish for instance the delays caused by lack of the only resource requested by a concurrent action and hence waiting for this resource to be released.

As described in [BPS09], there are many essential aspects of resources, which should be taken under consideration during workflow simulation. One of them is already mentioned: the multifunctionality of resources. Resources have attributes describing their skills. In other words these are the abilities to perform certain actions. Each resource is associated with the set of activities, in which it can be used. Other attributes taken under consideration are performance and cost associated with engaging the resource. So we know how fast a resource can do an activity and how much it costs to use it. By expense of a resource we mean here a value per time unit associated with involvement of the resource. A chief accountant probably has a driving license, but using him to drive the documents somewhere can be a waste of his time and precious skills. His cost per hour is much higher than that of a professional driver.

Since workflows can be quite complex, the authors do not see any analitycal approach, which would be effective in the optimization of runs. It is hard to predict all the possible outcomes of the allocations decisions, when resources are in many conflicts. Instead, we propose an experimental approach, which involves the simulation of many runs, taking into account different allocations and estimating the desired measures as a result of allocation decisions. In our prototype IRS-MT (Intelligent Resource Sharing-Modelling Tool) the manager can edit a structured workflow net and set the number of experiments. The program makes random allocations reporting times and costs of the runs. Based on this knowledge the manager can use the suggestions of the program and get a picture of possible outcomes of the decisions taken. The decisions can be made incrementally. After each decision the workflow run advances its state, and when we come to the next decision, a separate simulation will be made, basing on the actual state of the system.

2 Basic definitions

In our tool we consider resources (S), roles (R), requirements ($\mathbb{Q}$) and activities (A). Resources and roles are finite and defined a priori (by designer). For each resource we define a set of skills (roles), which it can use. This is denoted by $F_{SR} : S \to P(R)$ function. Additionally, for each role in a resource, its efficiency may differ. For every resource $s \in S$, we define the *efficiency* function $E_s : F_{SR}(s) \to \mathbb{R}_+$. This function describes how fast a resource performs each of its roles. The lower this value is, the higher is the efficiency (one is the base value). Additionaly, for every resource we define its use cost per time unit as a function $C : S \to \mathbb{N}$. On the other hand, for each activity in a workflow, a set of requirements needed to fire it, should be determined. Every requirement $Q \in \mathbb{Q}$ is a subset of roles. We define a function $F_{AQ}(a)$ which is a bag of requirements associated with the activity. It is important, that every activity has an expected duration time defined and its standard deviation value. Later, in the generation and simulation phase, only the available resources fulfilling these requirements will be considered for taking part in such activity. And so, $s \in S$ is *fulfilling* $q = \langle R_q \rangle$, iff $R_q \subseteq F_{SR}(s)$. It is important, that if a resource is involved in some activity, it cannot be used by any other activity (there is one exception, which will be covered later in this section).

Initially all the resources are free, available in a pool of idle resources. We assume, that all the resources are reusable, so at the beginning of activity execution the needed resources are collected and at the end all the freed resources will be returned to the pool of idle resources and will be available for further use. The graphical representation of relations between the introduced notions is depicted on Fig 1.

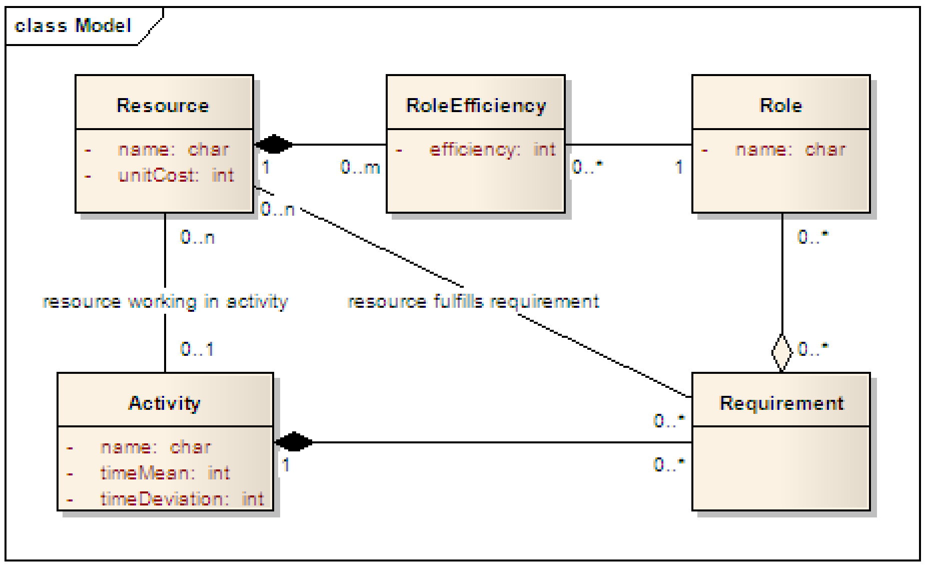


Fig. 1. Model

Our workflows are Petri nets created using five basic refinement patterns, proposed in [PCh03]: sequential-split of a place or transition, parallel-split of a place, choice-split of a transition and loop-split which attaches the spawned transition with a self-loop to a place. The additional rule for attaching a resource place is dual to the loop-split. It just glues a freshly created (resource) place to a transition by a self-loop. From the Petri net perspective we assume here that such place will contain initially a token for each physical resource available. We call the places created by such splits *resource places* or *activity places*. One cannot refine resource places. Even if the transition is refined, the resource will be allocated at the beginning of its execution and released, when it is done. Moreover, every resource place will carry the set of requirements defined for corresponding activity. If we match the resource places with activities accordingly, we will see, that the requirement assignments are defining the F_{AQ} function.

When we use activity refinement on a transition, which is inside some other activity, we will create a *nested activity.* By its *ancestor activity* we call every other activity, in which this one is nested. Such nested activities do not differ from any other activities, except for the fact, that these can use resources from its ancestors, as long, as the resource will not have one of its roles assigned to requirements in two different activities.

3 Tool overview

The main goal of this work is to provide a a tool, which can be helpful in improving the resource management. We concentrate on optimization of the workflow execution by providing the user with relevant statistical information and letting him make decisions on resource-to-activity assignments. To achieve this, a Petri net defining a workflow with activities and resources, will be created. For this purpose we introduce a *Workflow Designer.* It is the editor for building workflows using refinement patterns. In the editor we create activities, declare the set of requirements and approximate execution duration. On the other hand, we have *Resources Editor.* We use it for defining a pool of resources, assigning roles to them, and determining their effectiveness in each role. The set of defined roles can be modified by an always-visible editor *Roles Viewer.* All these three editors form the *static part* of the tool. Its more detailed description is presented in section 3.1.

After defining the model in the *static part*, we can proceed to the *dynamic part* of the tool. We will use *Generator*, to create sufficent number of random runs. When this is done, the *Simulator*, the *Report Viewer* and the *Bucket Editor* shows up. The first one displays a copy of our workflow, where resource places contain both the requirements and lists of currently available resources, fulfilling given requirements. Here, the tool provides us with information about expected time and cost of workflow completion for each of resource-to-requirement assignment we see. Basing on this knowledge we can decide, which resource should be assigned for current requirement. Every choice causes the expected values to be recalculated, so that we can run the workflow deciding, how resources should be used in the activities. The Report View presents detailed information about expected time and cost of workflow completion. It is synchronized with current simulator state, so all the values are always up-to-date. The Bucket Editor is a tool for storing, and presenting details of the runs, which were manually performed by user in the Simulator.

For example, let us consider the following, simple case. There is a package, which must be delivered to the destination within an hour. We know, that standard travel time by a scooter in current traffic would take around fifty minutes. We have two available scooter drivers: Evan and Gregory. Both of them can do the delivery, but Gregory has got his license for much longer time than Evan, and he shortens the expected delivery time by around 20 percent. Evan, on the other hand, is still afraid of driving fast and using tricky shortcuts, so his deliveries often take 20 percent longer than normal. However, Gregory's earnings are twice the earnings of Evan. This is the classic case, where no optimal resolution to the problem exists. It must be up to user's decision, whether time or cost is more important, and it is up to our tool to provide the user with information about expected cost and time consequences of each decision. We achieve it by simulating many thousands of runs and preparing the estimates with all aspects of the workflow taken into consideration, i.e.: activities order and dependencies, resources usage conflicts, possible time/cost variations.

3.1 Static Part

To present the tool in more details, we introduce other, more complex example. Let us suppose, that we have two rooms: A and B. We need to plaster and paint both of them. Additionaly, room A needs to be decorated. Here we assume, that a room cannot be painted, unless it is plastered and it cannot be decorated, unless it is painted. Expected time units, required to complete these tasks for each room are following:

- Room A — plastering: 20, painting: 10, decorating: 20;
- Room B — plastering: 10, painting: 20.

This process is presented on Fig. 2. We also need two resources, applicable for the work: *Steven* and *Tom*. *Steven* is an experienced *painter*, with a skill of *plastering*. *Tom*, on the other hand, is a *decorator*, who also can *paint*, but it takes him some more time that it does for *Steven*.

The purpose of the static part of the tool is to model this situation, so that it can be then simulated and later analysed in the dynamic part. We will now explain, how it can be achieved using available functionalities.

Resources Editor Defining resources consists of identifying the available set of people, machines and tools. For each of them, we can define a set of attributes like: use cost per time unit, collection of applicable roles (skills) and the effectiveness in each role. Every resource may have many skills, and many resources can have the same role. As it turns out, we often miss such knowledge during resources allocation planning and we do not take all the benefits from what a resource is capable of doing. Because of that, we can also miss the optimal resolution for given situation. Therefore we need to integrate all these aspects in the analysed context, so that we can consider consequences of particular situations with respect to most important factors. It is worth noticing, that in most popular Human Workflow management tools like Tibco [BS07] or Corel iGraphix, as well as in some academical tools like Yasper [YA06], during the workflow design and simulation phase, the roles are treated as resources. No resource, can have two skills. This, for modelling purposes, is a major limitation. It means, that one resource will never be requested for two activities with different required roles, which tightly limits analysed possibilities. In our tool there are no such boundaries.

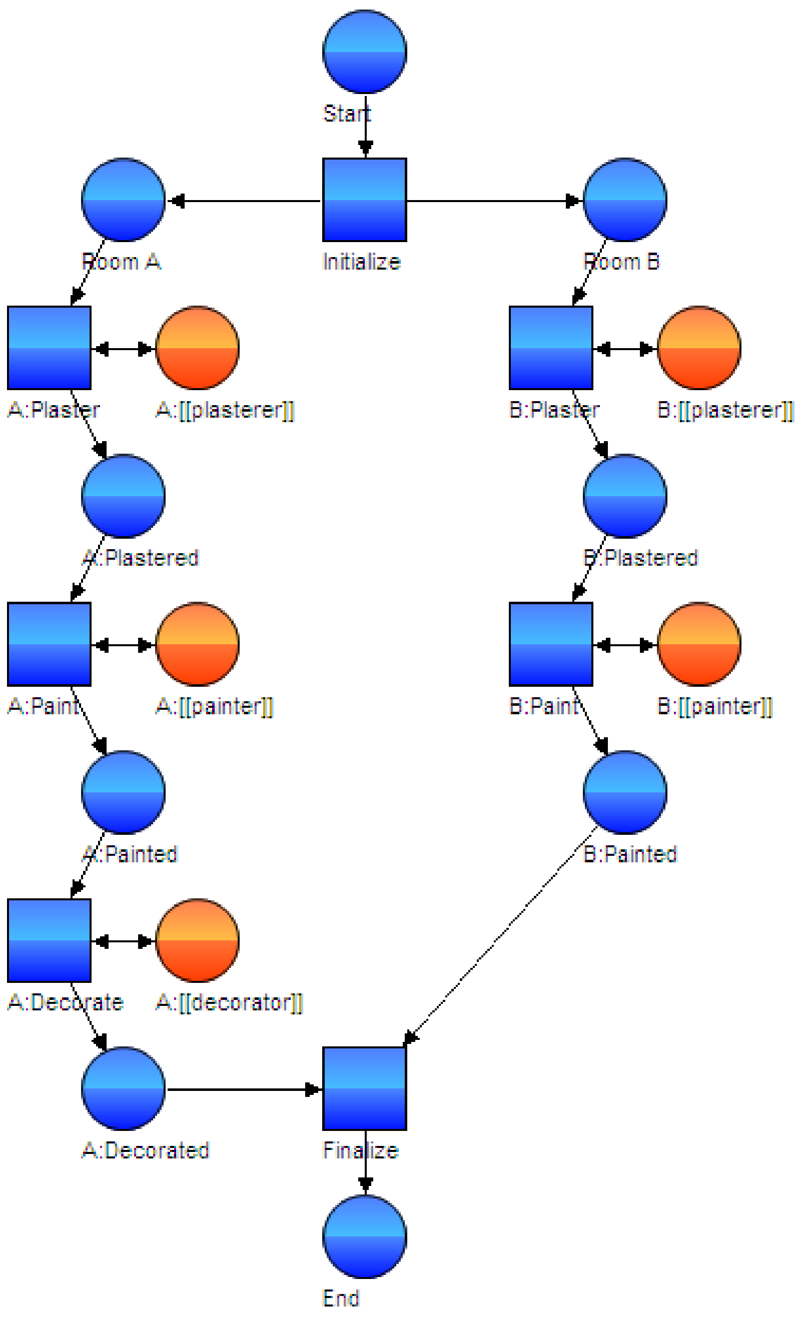
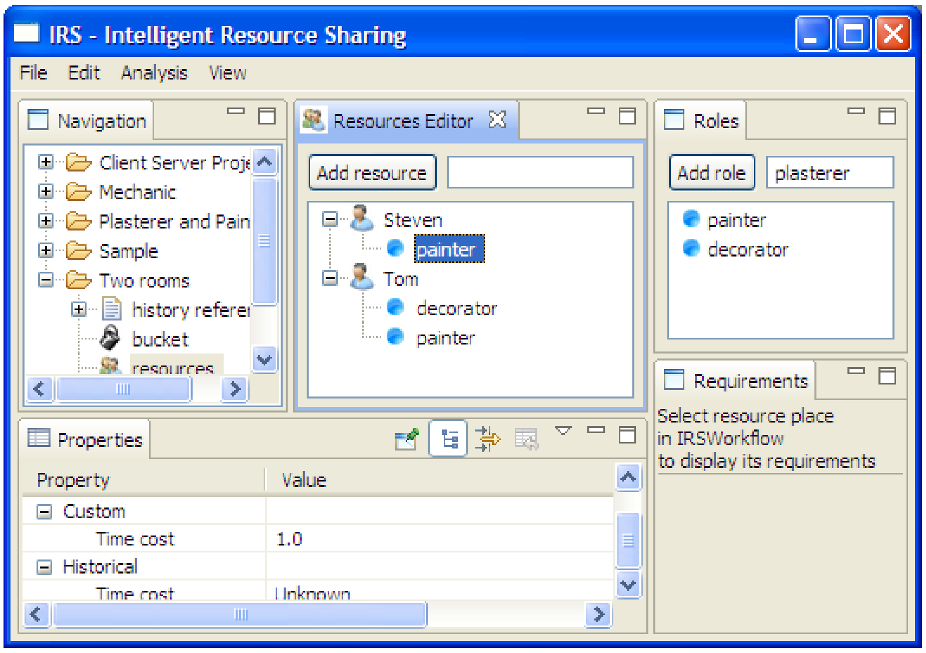


Fig. 2. Sample net



Select resource place
in IRSWorkflow
to display its requirements

Property	Value
Custom	
Time cost	1.0
Historical	
Time cost	Unknown

Fig. 3. Resources edition

Fig. 3 presents the pair of our resources: *Steven* and *Tom*. The interface is straightforward and it uses drag and drop features across almost every view. At the depicted state of modeling Steven lacks the plastering skill yet. In order to add it, we type *plasterer* in *Roles View*, press *Add role* button, drag the newly created role from this view and drop it over the *Steven* entry. New skill would get all of the necessary attributes initialized to default values. At the bottom of the window, we can see the *Properties* tab, which, as a context panel, allows us to modify attributes of currently selected object. The information about Steven's efficiency as a *painter* (*Time Cost* row) is displayed here. To be consistent with the descritpion, we should change this value from 1.0 to 0.8. This indicates that activity performed by Steven in a role of *painter* could take 20 percent shorter than normal. We repeat similar scenario for Tom, associating with him the role of painter and decorator and setting the efficiency coefficients for each of these roles. Expenses for each person should be defined also here.

Workflow Designer In the presented tool we can model sound workflows using refinement patterns presented in [PCh03]. To make this process easier it is possible to apply these patterns to any node the refinement tree, including the inner ones. Each node can also have its subtree truncated. The structural approach has been chosen to simplify both the design process and the inner application processing engine. In this tool we introduced the basic set of six refinements.

These might be extended in future by other patterns like communication or synchronization patterns described in [PCh03].

Editor offers additional operations for collapsing and expanding nodes, according to the information held in the refinement tree, so we can view our net at desired level of detail. The application displays all the nodes automatically in the viewer, but we can also move them around manually. The process from the example, has been created using refinement patterns, as shown on Fig. 4. Currently the *B:[[plasterer]]* resource place is selected, so that we can see, which requirements are defined for this requirement place in the bottom right tab *Requirements* (in our case all resource places will be labelled ⟨room name⟩:[[⟨required role⟩]]). As it was mentioned earlier, resource places are created by the *ACTIVITY* pattern, and cannot be further refined. Each resource place is associated with exactly one activity, so it is the right place to hold all the needed requirements for activity.

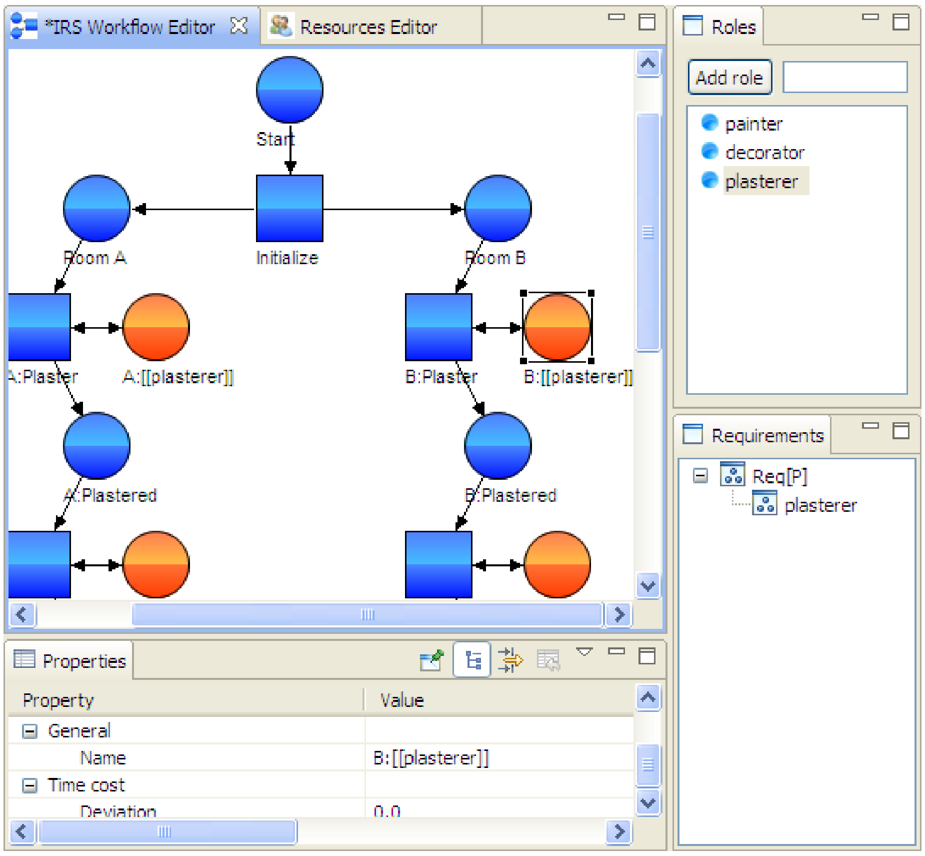
The whole net has been built using only the *SEQUENCE*, *CONCURRENCY* and *ACTIVITY* patterns. On the Fig. 5, all possible refinement operations are presented both for each standard place (not resource place) and each transition.

Every transition has got a special property, a measure, which describes its *desire to be executed.* We can modify this value to indicate transitions, which should have higher/lower probability of being executed, when in conflict with some other ones. This value is used only when at least two transitions are in conflict. By default this value is set to 1, but if we wish to make some transition to be executed more often, this value should be set accordingly. For instance if we have two active transitions, one with this value set to 3, and the second one to 1, then the first of them will be fired with 75% chance. This way we can declare, which actions are more probable or what kind of situations occur more often.

3.2 Dynamic Part

Statistics Generation Generator is a tool for preparing a list of complete runs with respect to guidelines defined in a workflow project and resources set. For this purpose, a special, extended copy of the designed workflow is created. Apart from copies of all the elements created by the designer, there are additional resource places for automatic resource availability management. It is guaranteed, that before each run, the whole net will be reset. Allocation decisions taken between two different runs are then mutually independent. It can happen that two identical runs could be generated by chance. Each run is executed and recorded using the following algorithm.

Initially, the *in* place of a workflow, is marked by a token, as well as there are tokens in the resource places. In a loop, a complete set of active transitions (activities) is constructed. If the set is empty, and the workflow is finished (a token is present on *out* place of the workflow), then the run is stored, the token from *out* place is moved to back *in* place. If there are at least two enabled transitions, one of them is randomly chosen (according to its *desire to be executed*). The selected enabled transition is fired and the loop is repeated. Note, that some



Property	Value
General	
Name	B:[[plasterer]]
Time cost	
Deviation	0.0

Fig. 4. Designer

of these transitions will indicate the begin or end of some activity. In such case an additional resource acquistion or return will be performed.

When the generator attempts to start an activity, all the requirements are being covered by skills owned by resources assigned to it. When we start one activity, some other can also start or end, so the tool can model various concurrent situations. When an activity ends, assigned resources are freed, the generator updates the workflow timer, the amount of time and cost counters for later analysis. A series of runs allows us to consider usefulness of certain choices in the context of further possible events.

The working time of the generator is dependent mostly on the number of allocations and releases of resources, which corresponds to one run. The computation power is also very important. In our case the generation of 10000 (ten thousand) runs on a standard computer takes no more than a few seconds.

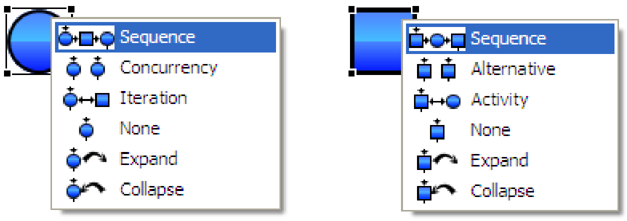
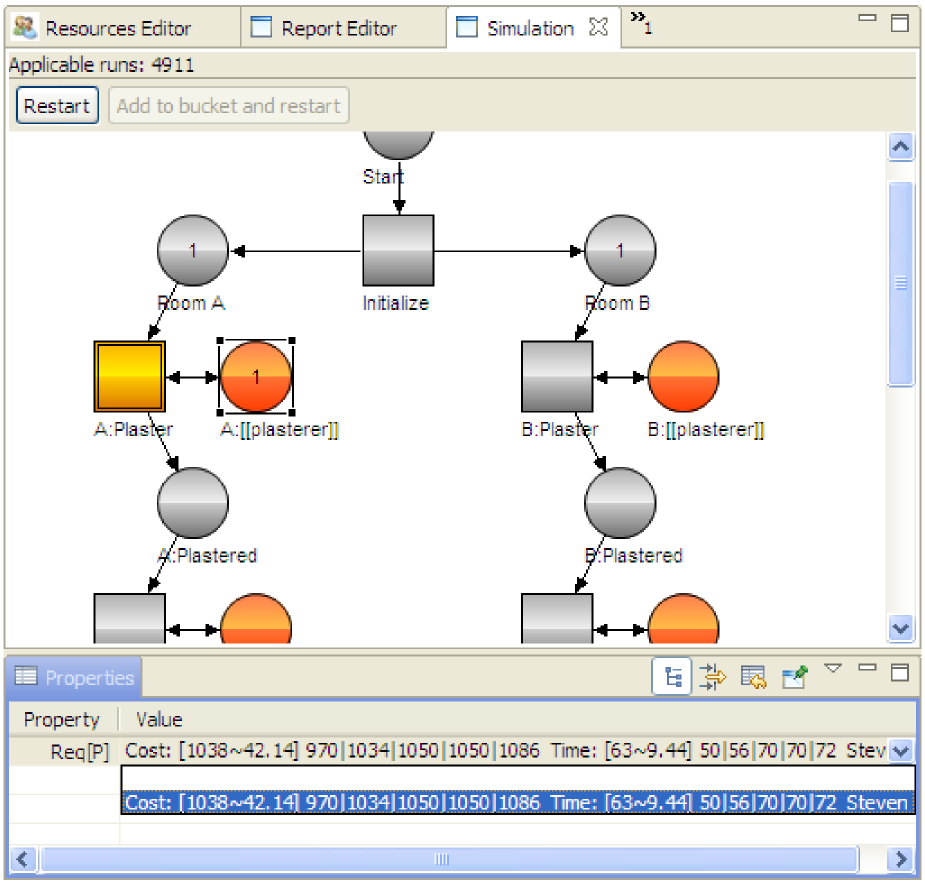


Fig. 5. Refinement patterns

Simulator Taking into account possible allocation decisions, we analyse time and cost of a workflow run. A few similar simulation tools have been reviewed in [BPM05]. On that basis some of the functionalities have been adapted to this tool, and a few flaws have been evaded. As a result, the application provides the simulator of the given net, and gives us a browser of performed runs.

In the report viewer (described later), we can browse all the runs created during generation. All these runs have been performed automatically, so using sorting capabilities of the report viever, we can easily find the optimal runs. Sometimes the differences in evaluation are very small, and the user may wish to examine the non-optimal allocation. For such purposes the simulator allows us to perform a step by step manual run of the designed net. An estimated time and cost of the run completion for every resource-to-requirement assignment at currently chosen state is presented. Thanks to this we know, which consequences are a result of our decision. The simulator remembers all the choices (resource assignments, order of activities start and end times) that a user makes during a simulation. It uses this knowledge to find all the statistical runs that are applicable to this situation, and then provides us with the estimates. Note, that in neither of Tibco, iGraphix nor Yasper, such manual interference in resource assignments is possible, because all resources in these simulators can have only one role. Moreover, up to the authors knowledge, there is no other tool, which supports simulation of resources with multiple roles, giving the user a chance to take part in a simulation at such informative level.

Let us suppose that a simulation has come up to a situation shown on the Fig. 6. The tokens presence on places *Room A* and *Room B* means that the *Initialization* phase has already ended. As we can see, enabled transitions have double-lined border. At the moment both *A:Plaster* and *B:Plaster* transitions require the same skills (in our case: *plasterer*), which have been defined in the corresponding places: *A:[[plasterer]]* and *B:[[plasterer]]*. Both places contain no tokens, which means, that no resources have been assigned to these activities yet. Currently selected place *A:[[plasterer]]* shows in the *Properties View* all the possible resource assignments for the *plasterer* skill. As it turns out, only *Steven* is suitable. Next to his name there is some information indicating the estimated cost and time of the run completion, with him taking part in this activity. In our example, $Steven's\ Cost$: $[1038 - 42.14]$ 970|1034|1050|1050|1084 $Time$: $[63 - 9.44]$ 50|56|70|70|72 means:



Applicable runs: 4911

Property	Value
Req[P]	Cost: [1038~42.14] 970|1034|1050|1050|1086 Time: [63~9.44] 50|56|70|70|72 Stev
	Cost: [1038~42.14] 970|1034|1050|1050|1086 Time: [63~9.44] 50|56|70|70|72 Steven

Fig. 6. Simulator

– average cost of completion is 1038 units with standard deviation of 42,14;
– average costs of completion in five successive quantiles are: 970, 1034, 1050, 1050, 1084;
– average time of completion is 63 units with standard deviation of 9.44;
– average times of completion in five successive quantiles are: 50, 56, 70, 70, 72.

When we select the *B:[[plasterer]]* place, the estimates for *Steven's* change to $Cost:$ [1004 – 44.44] 950|988|1000|1011|1075 $Time:$ [65 – 8.73] 60|60|60|69|80. So, starting work in *room A* will cost us more, but it will shorten the total execution time. Knowing this at such an early stage lets us undertake proper decisions from the very beginning. On that basis we may tend to choose *room A* first, if we want to save time. On the other hand, when cost is being considered crucial, we should choose beginning work from *room B*.

When all requirements are met, the corresponding activity becomes active. We can fire such enabled transition, and move to the next activity, where we will have more decisions to make. Note, that at every time, we make a decision or fire a transition, the estimates are being recalculated, to show the current situation in the net. This rule also applies to the *Report Viewer.*

Report Viewer Much more information about current situation in the simulated net can be seen on additional report view. The sample screenshot on Fig. 7 presents the state of report viewer, when *Steven* has been assigned to *A:Plaster* activity in the simulator. Thanks to synchronization between these two tabs, we can always take a look at all the computed details and expected values. Diagrams on the 7 present:

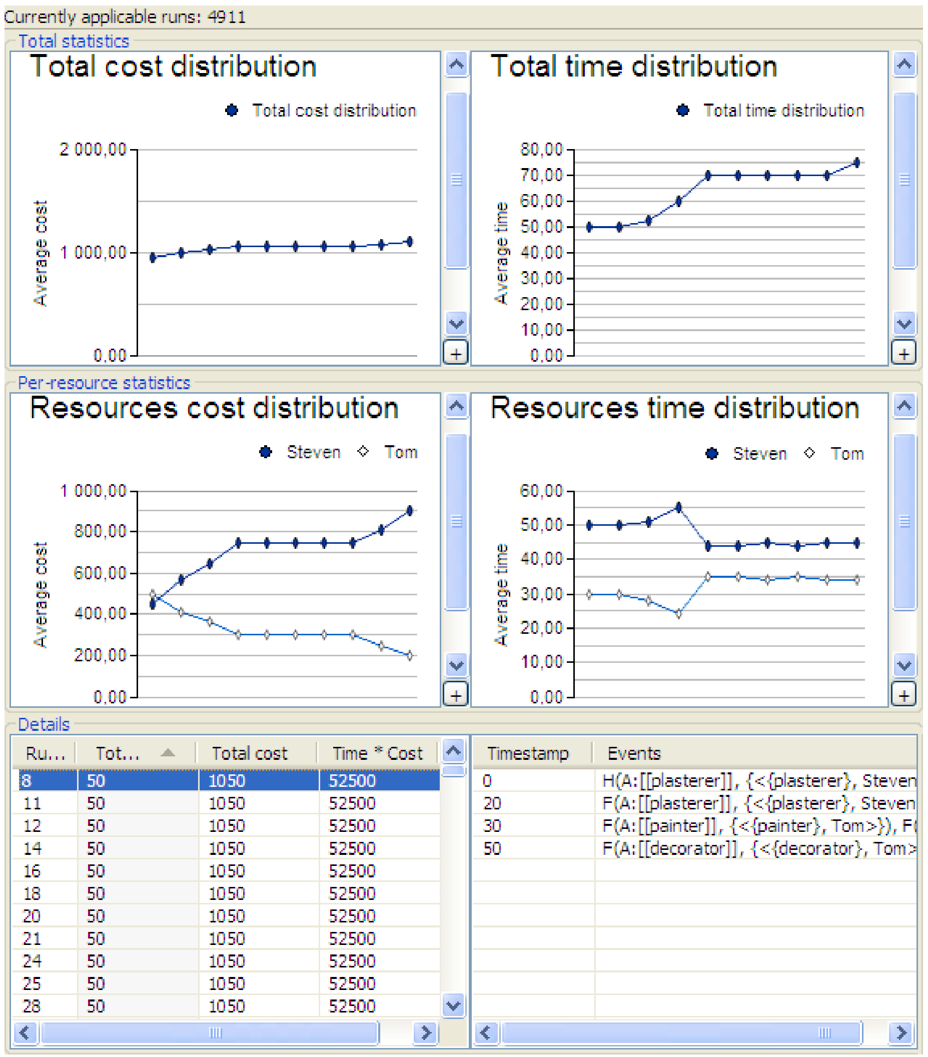
– **Total cost distribution** — average, cumulative cost of run performance in current simulator situation in 10 successive quantiles;
– **Resources cost distribution** — as above, but for each of the resource;
– **Total time distribution** — average time of run performance in current simulator situation in 10 successive quantiles;
– **Resources time distribution** — as above, but for each of the resource.

Diagrams of total cost and time provide us with information about differences between optimistic and pessimistic run executions. When viewing costs of resources, we can see, which of them have major influence on the growth in pessimistic cases. In the example shown above it is a fact that, when considering cost, in the optimistic case (on the left side of the diagram), *Steven* involvement is minimal, and the cost of *Steven's* work even goes below the cost of *Tom's* work. On that basis we can conclude that the main way to cut the cost will be to maximize *Tom's* involvement and minimize *Steven's.* It can be seen that bigger involvement of Steven means smaller involvement of Tom and vice versa. So if time does not matter we will prefer the cheaper resource.

The distribution of resource involvement time gives us somewhat different conclusions. It cannot be explicitly determined whether one of the resources should be favoured to improve the process duration. It may seem at a first glance, that in order to optimize the run we should always choose the fastest resource. But it is not the case, since it can slow down other parts of the workflow. The faster one can be the only one who can perform another action which should not be delayed.

Two additional tables at the bottom of this view present:

– list of runs, which are up to date with current situation in the simulator (including time and cost of them as well as pointing out the three most busy resources);
– list of successive concurrent events for the run selected in the first table.

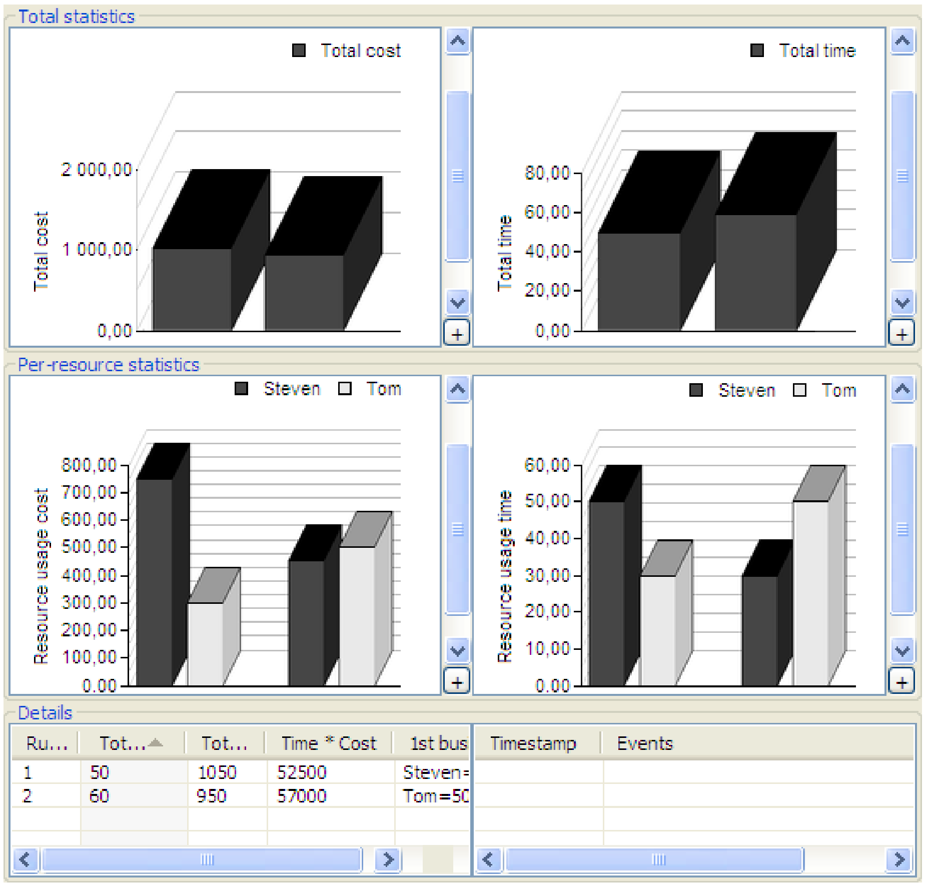


Ru...	Tot...	Total cost	Time * Cost
8	50	1050	52500
11	50	1050	52500
12	50	1050	52500
14	50	1050	52500
16	50	1050	52500
18	50	1050	52500
20	50	1050	52500
21	50	1050	52500
24	50	1050	52500
25	50	1050	52500
28	50	1050	52500

Timestamp	Events
0	H(A:[[plasterer]], {<{plasterer}, Steven
20	F(A:[[plasterer]], {<{plasterer}, Steven
30	F(A:[[painter]], {<{painter}, Tom>}), F
50	F(A:[[decorator]], {<{decorator}, Tom>

Fig. 7. Report viewer

Bucket Editor To ease the efforts, and allow the user to store the most interesting runs for further analysis, a simple *Bucket Editor* has been created. After the simulation comes to an end, the user can store the run he has just performed manually. This run will be available for later view. The bucket presents information similar to the report viewer, but because it contains information for several, manually chosen runs (not thousands like report viewer), those can be presented in a slightly different form.



Ru...	Tot...	Tot...	Time * Cost	1st bus	Timestamp	Events
1	50	1050	52500	Steven=		
2	60	950	57000	Tom=5C		

Fig. 8. Bucket viewer

Fig. 8 shows a screenshot of the bucket with two stored runs. As we can see it is organized similarly to the report viewer, and the visible diagrams present the same information as those from report viewer:

– **Total cost** — total cost of execution each of the stored runs separately;
– **Resources cost distribution** — as above, but for each of resources;

– **Total time distribution** — total time required to complete each of the stored runs separately;
– **Resources time distribution** — as above, but for each of resources.

The bottom tables represent information about stored runs, showing exactly the same attributes and properties, as the report viewer does.

First of them (displayed on the left hand side of every diagram) has been performed with the goal to minimize the time. Second, though, was made to run the process as cheap as possible, ignoring the time at all. As we can see, the cost reduction requires extra time in workflow execution. Of course, increasing the execution speed increases the cost. Major differences can also be seen in the resource usage distribution in both of these cases. Cost reduction has led to a significant decrease of *Steven's* involvement. In this case, *Tom* takes some of his responsibilities. When the time was the goal, the proportions has significantly changed. *Steven* became more desired, because *Tom* seemed cause the most delays.

4 Conclusions and future work

The provided example shows, how largely the work organization can differ, depending on goals that we want to achieve. In many cases minor changes in the resource allocation can have large influence on the overall result and, conversely, sometimes major allocation changes result in very similar outcomes. Very often humans do not take under consideration all the aspects, which the presented tool does. Its role is to simplify this whole process and make it easier, to find a suitable organization plan that will fit our needs and fulfill the criteria. While the simplification of resource allocation planning is one point, the other one is the possibility to point out uncommon executions, which can lead to increased effectiveness and so increase our profits. Combining user's knowledge and computer's computation power could therefore lead to major design corrections, which would be a basis for further business improvements.

The presented tool is still a prototype, and there are some practical and theoretical issues that need to be addressed. On the theoretical side additional extensions to resource and workflow definition language should be introduced. This includes further conformance to the form of resources described in [BPS09], because only a few resource attributes have already been implemented in this tool. There is also a big potential of the currently used workflow language, which can be further extended. In the context of statistical data tooling, some additional information could be presented (e.g. the Student's t-distribution, $3 - \sigma$). Finally, no measure of reliability have yet been introduced and, in the sense of realism of modelled cases, this is one of the major flaws of the presented tool.

On the more practical side, the integration with existing systems and methods should be made. First of all the statistical runs of the net could generate logs from the run to make them readable by other applications in the same way as the YAWL does [WS09]. The conformance with CPN seems attractive, because of its formal basis and constant development. Secondly, the integration with currently used common tools is to be made as well. The LDAP for instance is a main source for human resource information. The reporting tools like Microsoft Dynamics AX (Microsoft Business Solutions – Axapta) can be a source of information about resource experience and effectivness of each of such resource. Any other form of gathering of knowledge about past and predictable future resource usefulness can become important in such case. Therefore more integration of this tool will be examinated and developed in the future.

Acknowledgment

The authors would like to thank the reviewers of this work for the effort they put into improving the paper by the constructive reviews, which resulted in a thorough revision of the paper.

Tool description

The tool is written in Java and can be run on any machine with Java (JRE 6.0) installed. It can be downloaded from the http://duch.mimuw.edu.pl/˜pch/IRSMT/irsmt.zip. The zip file contains a full version of the tool and a README file, which explains how to install and use the application.

References

[PCh03] Piotr Chrząstowski-Wachtel, Boualem Benatallah, Rachid Hamadi, Milton O'Dell, Adi Susanto, *Top-down Petri Net Based Approach to Dynamic Workflow Modelling*, Lecture Note in Computer Science. v2678. 336-353., 2003.

[BPM05] M. Laugna, J. Marklund. *Business Process Modeling, Simulation, and Design.* Prentice Hall, Upper Saddle River, New Jersey, 2005.

[YA06] Kees van Hee, Olivia Oanea, Reinier Post, Lou Somers, Jan Martijn van der Werf, *Yasper: a tool for workflow modeling and analysis*, Application of Concurrency to System Design, International Conference on, pp. 279-282, Sixth International Conference on Application of Concurrency to System Design (ACSD'06), 2006.

[BS07] Bruce Silver, Bruce Silver Associates, *The BPMS Report: TIBCO iProcess Suite 10.6*, BPMS Watch www.brsilver.com/wordpress, 2007.

[BPS09] W.M.P. van der Aalst, J. Nakatumba, A. Rozinat, and N. Russell. *Business Process Simulation: How to get it right?* In J. vom Brocke and M. Rosemann, editors, International Handbook on Business Process Management, Springer-Verlag, Berlin, 2009.

[WS09] A. Rozinat, M. Wynn, W.M.P. van der Aalst, A.H.M. ter Hofstede, and C. Fidge., *Workflow Simulation for Operational Decision Support.*, Data and Knowledge Engineering, 68(9):834-850, 2009.

Deadlock Control Software for Tow Automated Guided Vehicles using Petri Nets

Carlos Rovetto[1], Elia Cano[1] and José-Manuel Colom[2]

[1] Dpt. of Computer Science and Systems Engineering (DIIS)
[2] Aragón Institute of Engineering Research (I3A)
University of Zaragoza, Spain
{carlos.rovetto,elia.cano}@utp.ac.pa, jm@unizar.es

Abstract. Factoring and warehouse distribution centers face numerous and interrelated challenges in their efforts to move products and materials through their facilities. New technologies in navigation and guidance allow true autonomy with more flexibility and resource efficiency. In this paper we investigate a complete design approach to obtain deadlock-free minimal adaptive routing algorithms for these systems. The approach is based in an abstract view of the system as a Resource Allocation System. The interconnection network and the routing algorithm elaborated by the designer, are the initial information used to obtain in an automatic way a Petri Net model. For this kind of routing algorithms, we prove that the obtained Petri Net belongs to the well-known class of S^4PR net systems, and therefore the rich set of analysis and synthesis results can be applied to enforce the liveness property of the routing algorithm.

Key words: *AGVs, Control Software, Resource Allocation Systems (RAS), Modular Models, Structural Analysis.*

1 Introduction

Nowadays, many factories and warehouse and distribution centers use *Automatic Guided Vehicles* (*AGVs*) for item transportation among workstations. The wheeled trailers are the most productive form of *AGV* for tugging and towing because they haul more *conveyor-loads* per trip than other *AGV* types. In this paper we consider a warehouse distribution center as a programmable system for conveyor-loads movement among workstations using tugger *AGV*. The problem to be investigated concerns the design of *Deadlock-Free* minimal adaptive routing algorithms for the guidance system of tuggers *AGVs*, travelling into an warehouse distribution center. We say that the routing algorithm is minimal because only routes of minimal length between two workstations are taken into account. Moreover, the routing algorithms we are considering are adaptive in the sense that the route of a conveyor-load is constructed segment by segment. The assignment of a segment to the route of a conveyor-load is done in a workstation when the first trailer try to leave the workstation towards its destination workstation.

From the methodological point of view, the design of deadlock-free minimal adaptive routing algorithms is a complex task, where the designer experience is required because deadlock states can appear. There exist several approaches to cope with this problem [1–5]. They consider more general routing algorithms than those considered in this paper (including, for example, non-minimal routes). Because this generality, very few powerful analysis and synthesis results are available.

Our approach gives a full design cycle for *minimal adaptive routing algorithms* using Petri Nets as formal model that allows structural analysis of the liveness property of the model. Afterwards, if it is necessary, the initial routing algorithm is changed. From the point of view of software engineering, in the context of the control software for $AGVs$ systems, this paper intends to make contributions in the following directions: (a) The formalization of an *abstraction* process of the system to retain only the relevant characteristics in the study of deadlock problems in the routing software of $AGVs$. This abstraction is constructed around a minimal set of concepts – processes and resources. (b) The demonstration that for AGVs with minimal adaptive routing algorithms, the proposed abstraction process gives rise to models belonging to a well known class of Petri Nets named S^4PR, and so, we have many available results to cope with these systems. (c) *A modular methodology* to construct the models based on the specification of processes with resources that form the modules. The modules are composed by the fusion of common shared resources (segments) by different modules. This paper is organized as follows. In Section 2 an illustrative example is presented. In section 3 the proposed methodology is presented in detail. Section 4 presents the first step of the methodology consisting of the abstraction of the warehouse distribution center and the routing algorithm to retain only those aspects related to the appearing of deadlocks. Section 5 is devoted to the Petri Net model representing the Resource Allocation view of the system. This section also proves that the Petri Nets obtained for these routing algorithms belong to the class of the S^4PR nets. Section 6 presents the analysis and synthesis phases of the methodology that profit the theoretical results known for the class of S^4PR. Finally, section 7 presents some conclusions.

2 An Example

In this section, a simple example of a warehouse distribution center, will be presented. The specification of this example illustrates the typical situation in the transportation system of items. We start with a layout of the shipping areas defined by a set of workstations WS and a set of segments SG interconnecting the workstations. The connection pattern among workstations will be called the $framework$ of the warehouse distribution center. The example that we are considering is an unidirectional ring in clockwise fashion as underlying framework. There are four workstations WS=$\{w_0, w_1, w_2, w_3\}$ and they are interconnected by a set of segments SG=$\{sa_0, sa_1, sa_2, sl_1, sl_2, sl_3\}$. This warehouse distribution center is depicted in schematic way in Fig. 1.a. Observe that if a workstation has two output segments, a train can follow any of them. This decision is taken by the local *minimal adaptive routing algorithm* of the workstation. The other defining element of the warehouse distribution center is the behavior of the conveyors because a train can tow single or multiple trailers hence the length of the conveyors is variable. As the conveyors flow in pipeline fashion through

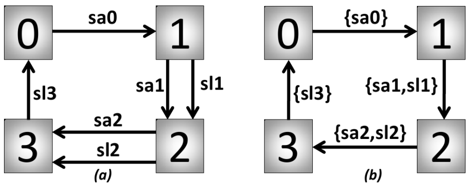


Fig. 1. a) Framework skeleton and its, b) Warehouse Graph.

the framework, these can have simultaneously allocated several segments of the framework. The first trailer of the AVG train is the **head** of the conveyors and reserve the segments to transit; the last trailer is the **tail** and release them. Traditionally, each segment supports only one conveyor at time to avoid collisions. In our example, each workstation executes, an instance of the following minimal adaptive routing algorithm parameterized by the identity of the workstation.

```
ALGORITHM 1 Minimal Adaptive Routing Algorithm for workstation i.
Input: The head trailer cl from the conveyor-load queue.
Local: $S_i \subseteq SG$, output segments for workstation i
       $F \subseteq S_i$, set of non-assigned output segments
Output: The next segment to be used for cl
begin
 if ($destination(cl) = i$) then store the conveyor-load cl in workstation i
 else
    if ( $sa_i \in F$) then use $sa_i$ ; $F{:=}F\backslash\{sa_i\}$
    else
       if (($destination(cl) < i$) $\wedge$ ($sl_i \in F$)) then
          use $sl_i$ ; $F{:=}F\backslash\{sl_i\}$
       else enqueue cl
       end if
    end if
 end if
end
```

That means the workstation knows its non-assigned output segments and the algorithm assigns, if it is possible, the output segment that the first trailer must follows in order to reach its destination. In other words, to reach a destination workstation, w_d, different to the current workstation w_i, the algorithm tries to assign the output segment sa_i if it is an output free segment of w_i. Otherwise, sl_i is assigned if this segment is an output free segment of w_i and the index d of the workstation w_d is less than the index i of w_i. This reservation is done by the head trailers. The intermediate trailers follow through the reserved segments and the tail trailer release the segments that they will be added to the set of free segments F. The design of minimal adaptive routing algorithms can lead to solutions where deadlock states can be reached. A deadlock state, in a warehouse distribution center, arise when a set of conveyor-loads are in transit to their respective destination workstations but all of them are stopped forever in intermediate workstations. They are waiting for the availability of output segments of these intermediate workstations that have been previously assigned to conveyor-loads belonging to this set. Therefore, none of the implied conveyor-loads will reach their destination workstations. The minimal adaptive routing algorithm of our example presents this anomaly that we illustrate by means of the following deadlock state. We have four conveyor-loads, $\{cl_1, cl_2, cl_3, cl_4\}$, each one composed by more than one trailer. It is easy to verify that the state described in table 1, for the four conveyor-loads in transit, is reachable, where H and T represent the current workstations of the head and tail trailers, respectively. The rest of the columns in the table 1 represent: *Allocated segments−* segments assigned to the conveyor-load; *Destination workstation−* represents the destination workstation of the conveyor-load; *Next segment−* segment to be assigned to the head trailer according to the minimal adaptive routing algorithm. Observe that

Conveyor	**Trailers**		**Allocated**	**Destination**	**Next**
-loads	H	T	**Segments**	**Workstation**	**segment**
cl_1	w_0	w_3	sl_3	w_1	sa_0
cl_2	w_1	w_0	sa_0	w_2	sa_1
cl_3	w_2	w_1	sa_1	w_3	sa_2
cl_4	w_3	w_2	sa_2	w_1	sl_3

Table 1. Deadlock state reached in the example concerning four conveyor-loads.

all conveyor-loads are in intermediate workstations and in order to advance in the warehouse distribution center, all conveyor-loads need segments (those given by the minimal adaptive routing algorithm) that they are allocated by other conveyor-loads in the same set (compare the two columns "Allocated Segments" and "Next segment" in the table 1). On the other hand, none of the tail trailers can release segments because if some tail trailer moves ahead, it will be in the same workstation that the head trailer and this is not possible. Therefore, we have reached a deadlock state where the four classical necessary conditions for the existence of a deadlock are fulfilled. Finally, you can observe that although we are in a deadlock state, there exist two segments, sl_1 and sl_2, that they are free, and the minimal adaptive routing algorithm cannot assign these segments to the four conveyor-loads of our scenario.

3 The proposed methodology

In this paper we advocate for a methodology where, after an analysis phase of the model obtained from the framework (the interconnection network) and the minimal adaptive routing algorithm, a synthesis procedure transform the original routing algorithm to make it deadlock-free. In order to implement this methodology we will make use of Petri Net models. Therefore, the first task will be the construction of the Petri Net model that retains only those aspects related to the appearing of the deadlock states. Deadlocks appear as a consequence of the allocation of the segments by the conveyor-loads in transit in the warehouse distribution center. Therefore, we will adopt a *Resource Allocation* perspective to abstract the system *(RAS view of the warehouse distribution center)* where segments will be considered as resources, that they are used in a conservative way *(they are not created nor destroyed)* by the user processes that they are the conveyor-load moving from a source workstation to a destination workstation. In next section, from the framework and the routing algorithm we will obtain a *Routing Graph* for each destination workstation. One of these Routing Graphs represents a transition graph where we present the reachable states of a conveyor-load, composed by more than one trailer, from a source workstation of the warehouse distribution center to the destination workstation corresponding to the Routing Graph. From these Routing Graph and the segments considered as resources, in section 5 we obtain a Petri Net that, in the case of minimal adaptive routing algorithms, belongs to the well known class of S^4PR nets. Now, using the known analysis results for this class of nets we can characterize the existence of deadlocks using a structural reasoning. The synthesis procedure is based on the methods for liveness enforcing developed by different authors [6] [7] [8]. The Fig. 2 presents in graphical form the methodology we propose for the design of deadlock-free minimal adaptive routing algorithms. In this methodology, the Petri Nets play a central role, because they are used to model the RAS view of the warehouse distribution center, and this is the reason of this paper: to present how to obtain these Petri Nets and to prove that they belong to a previously known class of Petri Nets (S^4PR), and so well studied.

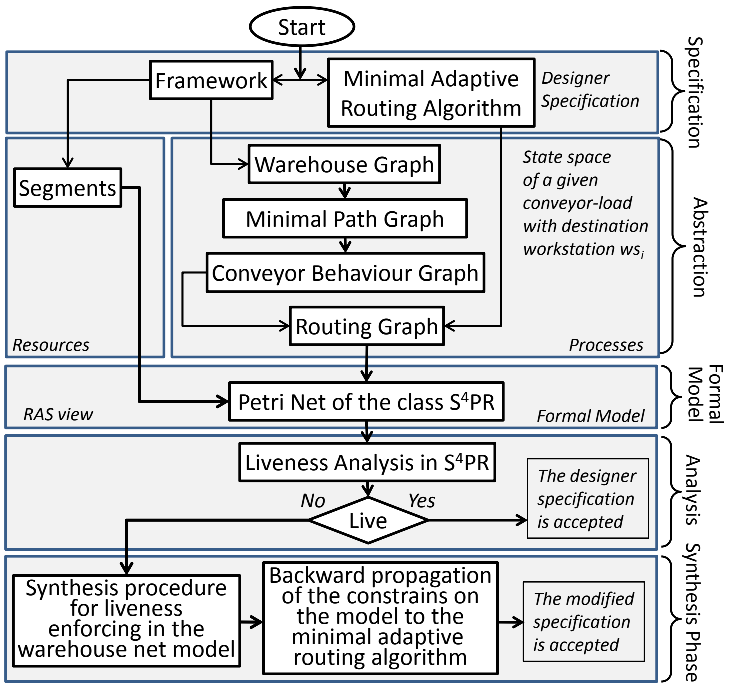

Start
Framework
Minimal Adaptive Routing Algorithm
Designer Specification
Specification
Segments
Warehouse Graph
Minimal Path Graph
Conveyor Behaviour Graph
Routing Graph
State space of a given conveyor-load with destination workstation ws_i
Resources
Processes
Abstraction
RAS view
Petri Net of the class S4PR
Formal Model
Formal Model
Liveness Analysis in S4PR
No
Yes
Live
The designer specification is accepted
Analysis
Synthesis procedure for liveness enforcing in the warehouse net model
Backward propagation of the constrains on the model to the minimal adaptive routing algorithm
The modified specification is accepted
Synthesis Phase

Fig. 2. Design Flow Methodology.

4 Construction of the Routing Graph

The goal of this section is to represent, step by step, the construction of the so called *Routing Graph* from the information about the framework of the warehouse distribution center and the minimal adaptive routing algorithm. This Routing Graph will represent the sequence of states that a conveyor-load must follow to reach a given destination workstation, w_i. The definition of the state concerns the set of segments that the conveyor-load is using each time. Therefore, *RG* give us the so called *Resource Allocation (RAS)* view of the warehouse distribution center. First, the framework is formalized through the *Warehouse Graph (WG)*. The WG is a labeled graph $WG=(W,S)$, where W is a set of vertices and S is a set of edges. The set W is equal to WS and $S \subseteq W \times 2^{SG} \times W$, where WS is the set of vertices and SG the set of segments. An edge $(w_1, s, w_2) \in S$ means that there exists a set of segments $s \subseteq SG$ from the workstation w_1 to the workstation w_2, as it is shown in the Fig. 1.b. We are considering the class of minimal adaptive routing algorithms. Therefore we will represent for each destination workstation w_i all the paths of minimal length from a workstation w_j to the destination workstation w_i. This information is captured into the *Minimal Path Graph (MPG_i)* with destination workstation w_i. Each one of these graphs is a subgraph of the $WG = (W,S)$ and it will be an acyclic directed labeled graph, $MPG_i=(V,E)$, where $V=W$, and $E \subseteq S$, verifying that:

1. All output arcs of w_i in WG do not belong to E.
2. The function $L_i{:}V \to \mathbb{N}$ is well defined: $L_i(w_i)=0$ and $\forall w_j \neq w_i$, $L_i(w_j)=k$, where k is the length of the minimal path from w_j to w_i in the WG.
3. All arcs $(w_1, s, w_2) \in S$ in WG, such that $L_i(w_i)+1 \neq L_i(w_2)$, do not belong to E.

The graphs MGP_i for the example of Fig. 1 are depicted in the Fig. 3. Observe that we will have four of these graphs, one for each possible destination workstation. Each MPG_i can be seen as the set of paths that can follow a conveyor-load

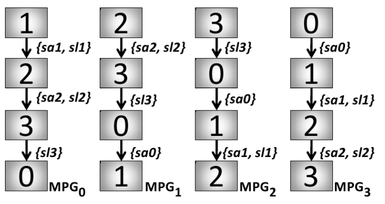


Fig. 3. Minimal Path Graph for all destination of our example.

originated in the workstation w_j with destination workstation w_i, an this path satisfy the minimality condition of the considered routing algorithm. Nevertheless, we are considering conveyor-loads with more than one trailer of length, because a conveyor-load with only one trailer cannot participate into a deadlock, since a deadlock must fulfill the *Hold and Wait* condition. Therefore in our model we must distinguish states according to the workstations where the head and tail trailers can be found. On the other hand, it is important to say that the advancement of the head trailer from a workstation to another can be done if and only if there exists at least a segment that can be allocated for this movement. Segments, therefore are resources in our RAS view of the warehouse distribution center. If the head trailer allocates the needed resources for the movement of the full conveyor-load, the tail trailer take charge of the release operation after the use of a segment. In order to represent the states of a conveyor-load with destination workstation w_i we will construct, from the MPG_i, the so called *Conveyor Behaviour Graph (CBG)* for the destination workstation w_i, $CBG_i{=}(Q, F)$, verifying that.

1. $Q{\subseteq}V{\times}V$, where $\forall w_h, w_t{\in}Q$, $w_h{=}w_t$ or $L(w_h){<}L(w_t)$. That is, the first component of the defined states corresponds to the workstation where the head trailer is, and the second to the workstation where the tail trailer can be found.
2. $F{\subseteq}Q{\times}\{A, R\}{\times}2^{SG}{\times}Q$, where F will contain the following edges:
   (a) Allocation edges $((w_{h1}, w_t), A, S, (w_{h2}, w_t))$, $\forall\ w_t{\in}V$,$\forall(w_{h1}, s, w_{h2}) \in E$.
   (b) Release edges $((w_h, w_{t1}), R, S, (w_h, w_{t2}))$, $\forall\ w_h{\in}V$,$\forall(w_{t1}, s, w_{t2}) \in E$.

Obviously, CBG_i is a directed acyclic graph because MPG_i is also a directed acyclic graph. The Fig. 4 shows the Conveyor Behaviour Graph for destination workstation 0, CBG_0, corresponding to the MGP_0 of Fig. 3. Finally, to

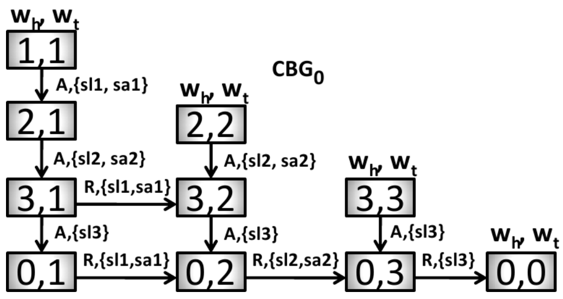


Fig. 4. Conveyor Behaviour Graph for destination workstation 0.

construct the announced Routing Graph of our warehouse distribution center we need to incorporate the information corresponding to the routing algorithm. The routing algorithm is a function $R{:}WS{\times}WS{\rightarrow}2^{SG}$, such that if wc is the current workstation of the head trailer and wd the destination workstation of the conveyor-load $R((wc, wd))$ determines the output segments of wc to be allocated in order to reach the destination workstation. The model that we will construct is a possibilistic model, in the sense that from a current workstation we can have several alternative transitions, each one corresponding to a different allocated segment. Therefore in order to represent this information of the routing algorithm, from each CBG_i we will construct the so called *Routing Graph (RG)* to the destination workstation w_i, $RG_i{=}(Q', F')$, where $Q \subseteq V \times V \times SG^*$ represents the set of states in which a conveyor-load can be found.

```
ALGORITHM 2 Construction of the $RG_i = (Q', F')$
Input: $CBG_i = (Q, F)$
Output: $RG_i = (Q', F')$
begin
  $next\text{-}level := \{(w, w, \varepsilon) | (w, w) \in Q\}$
  $Q' := next\text{-}level;\ F' := \emptyset;$
  while $next\text{-}level \neq \emptyset$ do
    $current\text{-}level := next\text{-}level;\ next\text{-}level := \emptyset;$
    for each $(w_1, w_2, r) \in current\text{-}level$ do
      for each $((w_1, w_2), X, S, (w_3, w_4)) \in F$ do
        for each $c \in S$ do
          if $(c \in R(w_1, w_i))$ and $(X = A)$
            then $next\text{-}level := next\text{-}level \cup \{(w_3, w_4, r\&c)\};$
                $Q' := Q' \cup \{(w_3, w_4, r\&c)\};$
                $F' := F' \cup \{((w_1, w_2, r), X, c, (w_3, w_4, r\&c))\};$
          endif
          if $(r = c\&t)$ and $(X = R)$
            then $next\text{-}level := next\text{-}level \cup \{(w_3, w_4, t)\};$
                $Q' := Q' \cup \{(w_3, w_4, t)\};$
                $F' := F' \cup \{((w_1, w_2, c\&t), X, c, (w_3, w_4, t))\};$
          endif
        endfor
      endfor
    endfor
  endwhile
end
```

The state is characterized by the workstations of the head trailer and the tail trailer, respectively, and the sequence of segments that, in this state, the conveyor-load maintains allocated; $F' \subseteq Q' \times \{A, R\} \times SG \times Q'$ is the set of arcs that represents the transition from a state to another by the movement of the head trailer or tail trailer. The movement of the head trailer allocates (A) the segment specified in the arc. Observe, that now in the RG_i a path from a state (w, w, ϵ), $w \neq w_i$, that corresponds to the birth of a conveyor-load in the workstation w, to the state (w_i, w_i, ϵ), represents the routing of a conveyor-load in the warehouse distribution center from the source workstation w to the destination workstation w_i. So, in order to obtain this $RG_i = (Q', F')$ from the corresponding $CBG_i(Q, F)$, we apply the algorithm 2 *(Note: with the symbol &, in the algorithm, we denote the concatenation operation of two strings)* In the Fig. 5 the RG_0, obtained from the CBG_0 of the Fig. 4. Applying the previous algorithm, we use the solid arcs to represent the segment allocation, and the dashed arcs to represent the segment release.

5 The Petri Net Model

In the previous section we have obtained the *RAS abstraction* of the warehouse distribution center plus the considered path selection algorithm. This abstraction

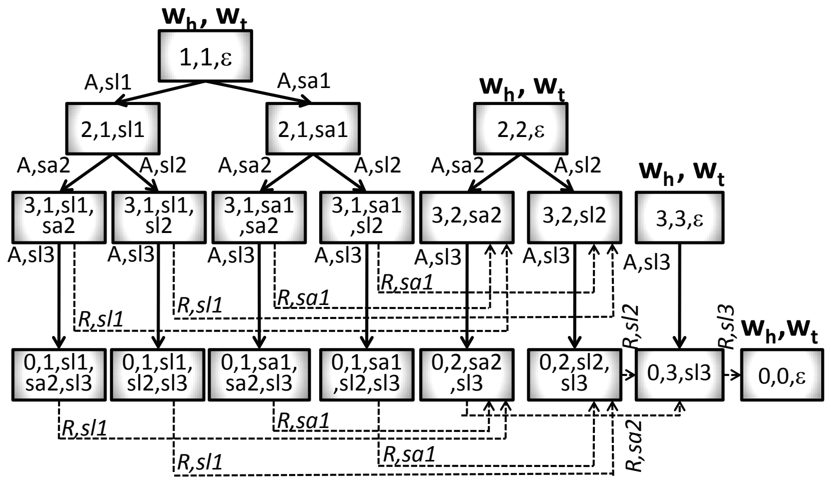


Fig. 5. Routing Graph for destination workstation 0.

is composed by the *resources*: The set SG of segments; and the set of *processes*: the set of routing processes to a destination workstation, each one represented by means of the corresponding Routing Graph. From these elements, in this section we proceed to the construction of a Petri Net integrating all processes and all resources.

First, from the $RG_i(Q', F')$, we construct the Petri Net $\mathcal{N}_i = \langle P_{0i} \cup Ps_i, T_i, F_i\rangle$ representing the state space of a conveyor-load born in the warehouse distribution center with destination workstation w_i. This construction proceeds according to the following rules.

1. Add a place to the set Ps_i for each vertex of the RG_i, $(w_1, w_2, s) \in Q'$ such that $w_1 \neq w_2$. The name of the place will be formed by the concatenation of identifiers of the workstations of the head and tail trailer, w_1 and w_2, respectively, and the sequence of segments that remain allocated for this conveyor-load and represented by s. All these places are unmarked at the initial marking M_0, because in the initial state there are not conveyor-loads in transit. We call these places *process places*.
2. Add a unique place po_i, $Po_i = \{po_i\}$, corresponding to the fusion of states of the form $(w, w, \epsilon) \in Q'$. The initial marking of this place will be equal to the maximum number of conveyor-loads that can be simultaneously in transit to the destination workstation w_i. If this number is not limited, or it its unknown, then we don't need to add a place po_i, i.e. the number of conveyor-loads with destination workstation w_i, in this network, is only limited by the available segments. These places will be called *idle places*.
3. Add a transition to the set T_i for each arc of the graph RG_i. For an arc $((w_1, w_2, s), X, c, (w_3, w_4, r)) \in F'$, the name of the transition will be $w_1\&w_2\&s\&w_3\&w_4\&r$. (The concatenation of this strings identifying the elements of the arcs).
4. For each arc $((w_1, w_2, s), X, c, (w_3, w_4, r)) \in F'$, $w_1 \neq w_2$, add an arc from the place $w_1\&w_2\&s$ to the transition $w_1\&w_2\&s\ w_3\&w_4\&r$, and an arc from this transition to the place $w_3\&w_4\&r$.

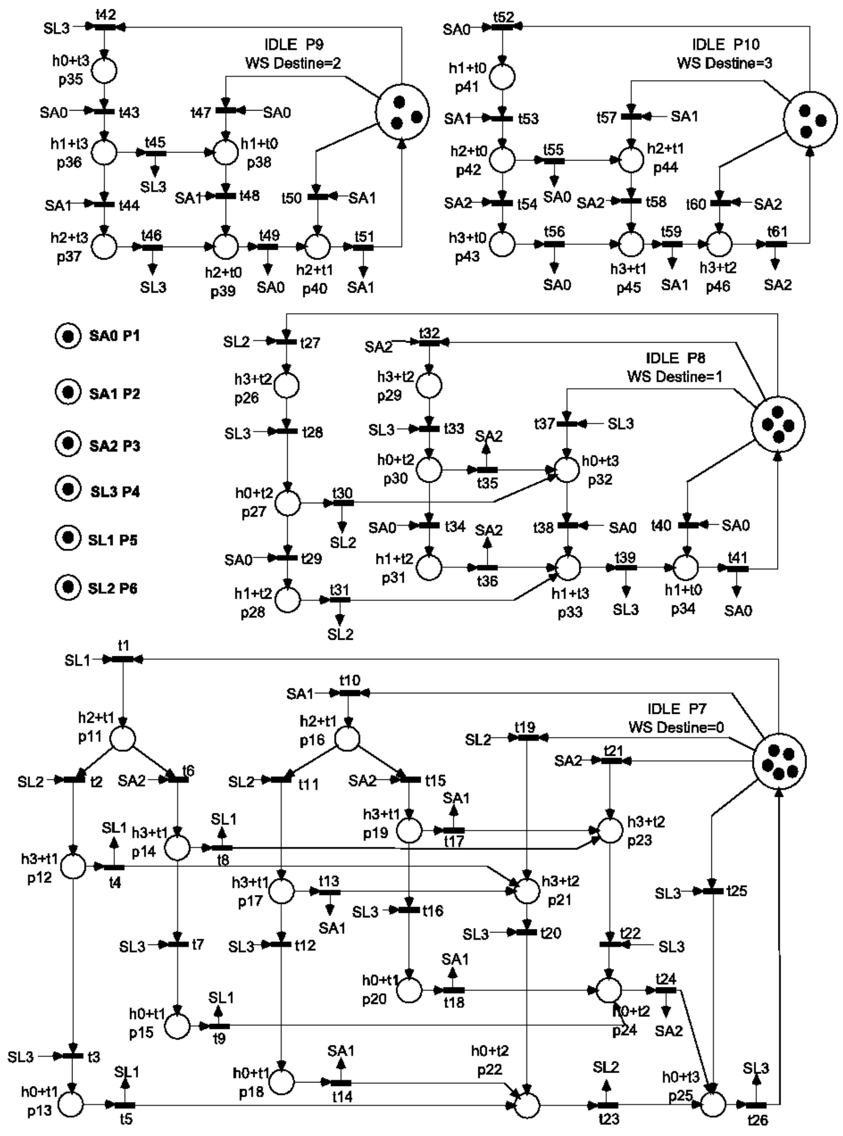


Fig. 6. Petri Net model for the example of the Fig. 1.

5. For each arc $((w, w, s), X, c, (w_3, w_4, r)) \in F'$ add an arc from the *idle* place $p0_i$ (if there exists) to the transition $w\&w\&s\&w_3\&w_4\&r$, and an arc from this transition to the place w_3, w_4, r.

Observe that the net $\mathcal{N}_i$, obtained following the rules of the preceding paragraphs, is a strongly connected state machine. In effect, by construction, each transition has only one input place and only one output place because a transition has been added for each arc in the graph RG_i, and the places correspond to the both ends of the directed arc. Moreover, it is a strongly connected state machine because all vertex in RG_i, (w_1, w_2, s), is reachable by a path from a source vertex (w, w, ϵ), since the construction of RG_i requires that we can construct the sequence of allocated segments s from a source vertex; and from a vertex (w_1, w_2, s) always exists a path to the destination vertex (w, w, ϵ). Taking into account that place P_{0_i} represent the fusion of all vertices (w, w, ϵ) of the graph RG_i, we can conclude that the net $\mathcal{N}_i$ is strongly connected. Additionally, we can see that all circuits of $\mathcal{N}_i$ contain the place p_0, because the original RG_i is acyclic. After all these transformations we obtain a set of strongly connected state machines $\mathcal{N}_i$, each one corresponding to a different destination workstation in the warehouse distribution center. The last step to obtain the RAS view of the warehouse distribution center is the addition of the resources, that, in this case, they are the segments connecting the workstations, and their integration with to the state machines. This can be done state machine by state machine and constructing the full model by fusion of the resource places with the same name. That is, we are constructing the model in modular way. The two steps to be applied are:

1. Add a place p_c to the set P_R for each segment $c \in SG$ of in the warehouse distribution center. The initial marking of this place will be equal to the maximum number of trailers that can be in transit simultaneously in the segment. *(Normally. it will be equal to one representing tha availability of the segment).*
2. For each arc of the graph RG_i of the form $((w_1, w_2, s), A, c, (w_3, w_4, r)) \in F'$, add an arc from the place p_c, (resource place representing the segment c) to the transition $w_1\&w_2\&s\&w_3\&w_4\&r$. This arc, in the Petri Net, will represent the allocation of the segment c. For each arc of the graph RG_i of the form $((w_1, w_2, s), R, c, (w_3, w_4, r)) \in F'$ add an arc from the transition $w_1\&w_2\&s\&w_3\&w_4\&r$ to the place c. This arc in the Petri Net represents the release of the segment c.

We denote this net by $\mathcal{N}_i^R$, representing the routing of the conveyor-loads to the destination workstation w_i, and the competition for the resource/segments. The full model is obtained from the different $\mathcal{N}_i^R$ by the fusion of the resource places (*segments places*) with the same name that appear in different $\mathcal{N}_i^R$. The Fig. 6 represents, in an schematic way, the full Petri Net corresponding to the example in Fig. 2. In this Fig. 2 the names of places and transitions have been simplified in order to maintain readable. In order to identify the states of the conveyor-loads with the places that represent them, we have used the simplified notation $h_i{+}t_j$; that it means that the head trailer is in workstation i and the tail trailer is in workstation j.

The final part of this section is devoted to prove that the obtained Petri Nets for warehouse distribution centers with minimal path selection algorithms belong to the subclass of Petri Nets named S^4PR [6][9]. In order to do that we recall the basic definitions of this class of nets.

Definition 1 (The class of S^4PR nets) *Let $I_{\mathcal{N}}=\{1,2,...,m\}$ be a finite set of indices. An S^4PR net is a connected generalised self–loop free Petri net $\mathcal{N}=\langle P,T,\mathbf{C}\rangle$ where:*

1. *$P = P_0 \cup P_S \cup P_R$ is a partition such that:*
   (a) $P_S = \bigcup_{i\in I_{\mathcal{N}}} P_{S_i}$, $P_{S_i} \neq \emptyset$ and $P_{S_i} \cap P_{S_j} = \emptyset$, for all $i \neq j$.
   (b) $P_0 = \bigcup_{i\in I_{\mathcal{N}}} \{p_{0_i}\}$.
   (c) $P_R = \{r_1, r_2, \ldots, r_n\}$, $n > 0$.
2. *$T = \bigcup_{i\in I_{\mathcal{N}}} T_i$, $T_i \neq \emptyset$, $T_i \cap T_j = \emptyset$, for all $i \neq j$*
3. *For all $i \in I_{\mathcal{N}}$, the subnet $\mathcal{N}_i$ generated by $P_{S_i} \cup \{p_{0_i}\} \cup T_i$ is a strongly connected state machine, such that every cycle contains p_{0_i}.*
4. *For each $r \in P_R$ there exists a minimal P–Semiflow, $\mathbf{y}_r \in \mathbb{N}^{|P|}$, such that $\{r\} = ||\mathbf{y}_r|| \cap P_R$, $\mathbf{y}_r[r] = 1$, $P_0 \cap ||\mathbf{y}_r|| = \emptyset$, and $P_S \cap ||\mathbf{y}_r|| \neq \emptyset$.*
5. *$P_S = \bigcup_{r\in P_R} (||\mathbf{y}_r|| \setminus \{r\})$.*

Each place p_{0_i} is called *idle place.* Places of P_R are called *resource places* being unique for the whole model. The Places of P_S are called *process places.* This definition must be completed with the definition of the *acceptable initial markings.* Initial markings represent no activity in the system, allowing the routing of each conveyor-load in isolation.

Definition 2 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$ be a S^4PR net. An initial marking $\mathbf{m_0}$ is acceptable for $\mathcal{N}$ if and only if: (1) $\forall i \in I_{\mathcal{N}}$, $\mathbf{m_0}[p_{0_i}] > 0$. (2) $\forall p \in P_S$, $\mathbf{m_0}[p] = 0$. (3) $\forall r \in P_R$, $\mathbf{m_0}[r] \geq \max_{p\in||\mathbf{y}_r||\setminus\{r\}} \mathbf{y}_r[p]$.*

From the previous definitions and the procedures described in the sections 4 and 5 to obtain the Petri Net model of an warehouse distribution center the following result can be easily verified.

Proposition 1 *Given an warehouse distribution center specified by means of a framework and a minimal adaptive routing algorithm, the Petri Net model obtained through the procedure described in sections 4 and 5, belongs to the class of S^4PR net systems.*

Proof (Sketch of the proof). In the section 5, after the rules to obtain the Petri Nets $\mathcal{N}_i$ from the corresponding Routing Graph RG_i, we have proven that each $\mathcal{N}_i$ is a strongly connected state machine, and for all $\mathcal{N}_i$,$\mathcal{N}_j$, $i \neq j$, they are disjoint net systems. We have also proven that every cycle of each strongly connected state machine $\mathcal{N}_i$ contains P_{0_i}. Therefore, to complete the proof we only need to prove the existence of a unique *p-semiflow* $\mathbf{y}_r \in \mathbb{N}^{|P|}$ for each resource r. But this is very easy to proof because from each transition where the resource place r inputs *(the resource is allocated)*, there exists a unique path, in the strongly connected state machine, to reach each transition where r outputs *(the resource is released).* Moreover, all transitions where r is an output place in the state machine $\mathcal{N}_i$ are connected by means of a minimal path from some transition where r is an input place. Therefore, the resource r plus all the process places defining the minimal paths connecting the output transitions of r and the input transitions of r form the *p-semiflow* that it is unique because we are dealing with the nets $\mathcal{N}_i$ that they are state machines.

Observe that the previous result is also true for non-regular frameworks because we are considering in an explicit way the paths to a destination workstation. Therefore, non regularity does not affect the final Petri Net. Nevertheless, non-minimality of the path selection algorithms can lead to more general class of nets than the S^4PR in the case of existence of cycles in the followed route by some conveyor-load. Once we have characterized the type of nets we can obtain, we can use the developed theory for S^4PR, trying to interpret these results from the point of view of the warehouse distribution centers, in the next section. In some cases we will see that we arrive to some negative results.

6 The Analysis and synthesis phase

The Petri Net model obtained in the previous section belong to the S^4PR class. Therefore, we can take advantage of this property and use the theoretical results about the liveness characterization in S^4PR. One of this results is presented in the following theorem.

Theorem 2 ([6]) *An S^4PR, $\langle\mathcal{N}, \mathbf{m_0}\rangle$, is non–live if and only if there exists a marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that the set of $\mathbf{m}$–process–enabled transitions is non–empty and each one of these transitions is $\mathbf{m}$–resource–disabled.*

This characterization is a state based characterization. The interpretation in terms of the warehouse distribution center is very easy. A token in a process place of the state machine $\mathcal{N}_i$ represent a conveyor-load in an intermediate workstation with destination workstation w_i. That is, is a conveyor-load in transit. The theorem 2 says that if all conveyor-loads in transit cannot advance because there is no an available segment to advance *(each one of these transitions is not enabled because an input resource place is empty)*, this situation characterizes a deadlock state: none of these conveyor-loads will arrive to its destination workstation because they are stopped forever in the current process places. In [6], verification procedures of the characterization stated in this theorem are presented. They are based in Integer Linear Programming Techniques.

An equivalent characterization to the previous one is based in the Petri Net concept of siphon. A siphon is a set of places that if they become a set of empty places, they remain empty forever *(these is a structural definition of siphon but we prefer to present the deep reason for the appearing of deadlocks in this class of nets)*. Therefore, all output transitions of the places of the empty siphon will be dead forever because at least an input place *(that belong to the siphon)* is empty forever. The presence of one of this siphons in the net is potentially bad because this siphon can become an empty siphon. The verification procedures search for a siphon and a reachable marking under which the siphon is empty. Empty siphons represent a generalization of the circular waits, because in a siphon we can find an intricate structure of superposed cycles of empty resources. For the Petri Net in Fig. 6, you can find the two following bad siphons D_i=$\{p_1$, p_2, p_3, p_4, p_{13}, p_{15}, $p_{17-to-20}$, p_{22}, p_{24}, p_{25}, p_{28}, p_{30}, p_{31}, p_{33}, p_{34}, p_{36}, p_{37}, p_{39}, p_{40}, p_{42}, p_{43}, p_{45}, $p_{46}\}$ and D_j=$\{p_1$, p_2, p_3, p_4, p_6, p_{13}, p_{15}, $p_{17-to-20}$, p_{22}, p_{24}, p_{25}, p_{27}, p_{28}, p_{30}, p_{31}, p_{33}, p_{34}, p_{36}, p_{37}, p_{39}, p_{40}, p_{42}, p_{43}, p_{45}, $p_{46}\}$. The deadlock state described in section 2 corresponds to the reachable marking written as a symbolic sum $m_r = p_5 + p_6 + 5 \cdot p_7 + 2 \cdot p_8 + \cdot p_9 + 2 \cdot p_{10} + p_{29} + p_{32} + p_{38} + p_{44}$. The reader can easily verify that the siphon D_i is insufficiently marked or he/she can verify the m_r satisfy the conditions of the theorem 2. Therefore, we conclude that the proposed path selection algorithm is not deadlock-free. After the previous analysis phase, the theory of S^4PR nets gives results and methods to enforce the liveness in the case of nets presenting deadlock states. These techniques transform the initial Petri Net model in such a way that deadlock states become not reachable. In some sense, they correspond to deadlock prevention techniques. We can incorporate this phase because we are using Petri Nets as formal model and they belong to the subclass of S^4PR. The known synthesis approaches enforcing liveness work on the bad siphons that can be found in the Petri Net model. These techniques can be classified into two groups.

1. **Centralized Approach:** [6][9] These techniques compute a place for each bad siphon preventing that the siphon becomes empty. This new place is of the same category that the resource places, and so it is said that the synthesis problem is solved by using virtual resources that they are implemented as a centralized monitors in the central software. In the case of the Petri Net of the Fig. 6 we need three places to make live the net. In fact, in some cases, to take the decision to allocate the virtual resource/segment in a local workstation we can need coordinate the local path selection algorithm with other local routing algorithms.
2. **Distributed Approach:**[10]. Previous limitations are solved developing a distributed control policy using the so called *swap virtual segments*.

All these methods are iterative, but the performed transformations maintain the transformed Petri Net inside the class of S^4PR nets.

7 Conclusions

The design of deadlock-free minimal adaptive routing algorithms for warehouse distribution centers is a complex and tedious task, for which the current methodologies, in many cases, only supply trial and error procedures. The assistance to the designer is very small in order to fix the problem in the proposed algorithm. In this paper we propose a methodology oriented to the design of deadlock-free minimal adaptive routing algorithms trying to cope with all phases of the design. The first step in this methodology consists of the *abstraction* of the system in order to retain only the elements of the system allowing the study of the appearing of deadlocks. These elements are the segments of the warehouse distribution center, that they are seen as the resources for which the routing processes compete to send conveyor-loads to destination workstations. The other elements are the routing processes itself that represent the routing sequence through the framework according to the routing algorithm. The result of this abstraction process is formalized by means of a Routing Graph for each possible destination workstation. From the Routing Graphs and the segments we have obtained Petri Nets that, for the class of routing algorithms that we are considering, belong to the class of S^4PR. Therefore, we profit that the class of S^4PR is a well studied subclass of Petri Nets and using the known results we can proceed with the analysis and synthesis phases of our methodology. So, the deadlock-free property in the warehouse distribution center correspond to the liveness-property in our Petri

Net model. The analysis of this liveness property can be done by two alternative characterizations that have a good interpretation at the level of warehouse distribution center. Algorithms and methods to verify the property can be found in [6]. In the case of non-liveness, there exist methods to enforce the liveness property based in the addition of places that can be interpreted in terms of Petri Net model as centralized software monitors.

References

1. Fanti, M.: A deadlock avoidance strategy for AGV systems modelled by coloured Petri nets. In: Discrete Event Systems, 2002. Proc. Sixth Int. Workshop on. (2002) 61 – 66
2. Wu, N., Zhou, M.: Resource-oriented Petri nets in deadlock avoidance of AGV systems. In: Robotics and Autom., 2001. Proc. 2001 ICRA. IEEE Int. Conf. on. Volume 1. (2001) 64 – 69 vol.1
3. Wu, N., Zhou, M.: Modeling and deadlock control of automated guided vehicle systems. Mechatronics, IEEE/ASME Transactions on **9**(1) (2004) 50 –57
4. Wu, N., Zhou, M.: Modeling and deadlock avoidance of automated manufacturing systems with multiple automated guided vehicles. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on **35**(6) (2005) 1193 –1202
5. Wu, N., Zhou, M.: Shortest routing of bidirectional automated guided vehicles avoiding deadlock and blocking. Mechatronics, IEEE/ASME Transactions on **12**(1) (2007) 63 –72
6. Tricas, F.: Analysis, Prevention and Avoidance of Deadlocks in Sequential Resource Allocation Systems. PhD thesis, Zaragoza. España, Departamento de Ingeniería Eléctrica e Informática, Universidad de Zaragoza (2003)
7. Tricas, F., Colóm, J., Ezpeleta, J.: A Solution to the Problem of Deadlocks in Concurrent Systems Using Petri Nets and Integer Linear Programming. In Horton, G., Moller, D., Rude, U., eds.: Proc. of the 11th European Simulation Symp., Erlangen, Germany, The society for Computer Simulation Int. (1999) 542–546
8. Park, J., Reveliotis, S.A.: Enhancing the Flexibility of Algebraic Deadlock Avoidance Policies Through Petri Net Structural Analysis. In: Proc. of the 2000 IEEE Int. Conf. on Robotics and Autom., San, Francisco, USA (2000) 3371–3376
9. Tricas, F., García-Vallés, F., Colóm, J.M., Ezpeleta, J.: A Petri Net Structure-Based Deadlock Prevention Solution for Sequential Resource Allocation Systems. In: Proc. of the 2005 IEEE Int. Conf. on Robotics and Autom., Barcelona, Spain (2005) 272–278
10. Rovetto, C., Cano, E., Colóm, J.: Liveness Enforcing Methods for Resource Allocation Distributed Systems using Petri Nets, Research Report RR-056-09. Depto de Informática e Ingeniería de Sistemas., University of Zaragoza (2009)

Acknowledgement

This work has been partially supported by the European Community's Seventh Framework Programme under project DISC (Grant Agreement n. INFSO-ICT-224498). This work has been also partially supported by the project CICYT-FEDER DPI2006-15390.

Nets-Within-Nets Paradigm and Grid Computing

Marco Mascheroni, Fabio Farina

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano Bicocca
Viale Sarca, 336, I-20126 Milano (Italy)**

Abstract. Grid is one of the most effective new paradigms in large scale distributed computing. Only recently Petri nets have been adopted as a formal modeling framework for describing the specific aspects of the Grid. In this paper we describe a Grid tool for High Energy Physics data analysis, and we show how modeling its architecture with nets-within-nets has led us to identify and solve a number of defects affecting the current implementation.

1 Introduction

In the last decade the Grid computing [10, 9] approach to parallel and distributed computing has defined a new path to enable high performance and throughput applications. Grid infrastructures expose computational and storage resources provided by different computing centers as uniform families of services that can be coordinated to create large scale e-Science workflows.

Grand-challenge experiments, like those related to High Energy Physics, life-science, and environmental science adopted the Grid as the tool for implementing their software. In this paper we will consider a Grid distributed data analysis tool developed to serve the community of the Compact Muon Solenoid (CMS) [19] experiment at the CERN Large Hadron Collider (LHC) [20]. A specific software tool has been developed to analyze physics data over the Grid, so that the users are protected from the architectural complexities of the distributed infrastructure itself. This application, called CMS Remote Analysis Builder (CRAB) [7] is released as open source software and has been adopted by the physics community since 2005. Even though the code quality is being continuously improved thanks to code analyzers (e.g., lint), the overall architecture has never been validated with formal tools like Petri nets.

The aim of this work is to validate some relevant parts of the CRAB tool using nets-within-nets [23]. In this paradigm the tokens of a Petri net can be Petri nets themselves. As we will see, the hierarchical structure of the system components is particularly suited for investigation with this formal framework. The RENEW tool [17] has been chosen as modeling platform, as it is the only nets-within-nets tool that is mature enough to describe a real system like CRAB.

** Partially supported by MIUR (Italian Ministry of Education, University and Scientific Research)

In particular, the features of Renew used to model the system are such that the obtained model is very similar to a hypernet [2].This is a class of high level Petri nets which implements the nets-within-nets paradigm using a dynamic hierarchy, and a bounded state space [3]. As detailed in Section 4, this approach allowed us to isolate some problems in the CRAB implementation. Our approach do not cover analysis yet: modeling and step-by-step simulation are the two means used to unveil these problems.

In the literature high level Petri nets have been applied to different contexts related to Grid computing technologies. Most of the works in this field focus on the usage of Petri nets as a tool for workflows specification and execution [1, 13, 11]. A different application of Petri nets to Grid is reported in [5]. Here the resources exposed by the distributed computing infrastructure are modeled directly with the aim of validating both properties like the soundness and the fairness of their sharing for a process mining workflow. As far as we know, high level Petri nets, and in particular hierarchical nets, have been applied neither to the Grid infrastructure, nor to the study of a classical Grid application pattern like the distributed data analysis.

The remainder of the paper is organized as follows: Section 2 introduces the basic notion of nets-within-nets we refer to, and the Renew tool. Section 3 describes the Grid architecture we are considering, while in Section 4 the modeling of the system and the bugs found thanks to the formal approach are presented. A discussion about the modeling choices used in our approach is made in Section 5. Finally, some conclusions are reported in Section 6.

2 The Nets-Within-Nets Paradigm and Renew

According to the nets-within-nets paradigm, the tokens of a Petri net can be structured as Petri nets themselves. This idea is due to Valk (see [21]), who defined and studied the class of *Elementary Object Nets (EOS)* in [22]. Later on, properties of EOS were studied in [15], and other classes of high level Petri nets which uses the nets-within-nets paradigm were defined, like for example [12, 2, 14, 24, 18].

In all these models a system is usually modeled as a collection of nets. One net is designated as the *system net*, the top level of the net hierarchy. All other nets are assigned to an *initial place*, a place in which they reside initially. This distribution of nets induces a hierarchy. The system evolves by moving tokens from place to place through the firing of autonomous transitions, or by synchronizing transitions between nets at different levels. The hierarchical structure of the model is usually static, but in some models there can be interactions between nets at different levels in the hierarchy which can dynamically change the hierarchy itself. For example, in hypernets a net N can be moved from a place belonging to a net A, to a place belonging to a distinct net B. The interaction between nets A and B is only possible if they are close in the hierarchy.

The development of the Renew software tool [17], a Java-based high-level Petri net simulator that provides a flexible modelling approach based on Reference nets [16], allows the use of this paradigm to model real systems. RENEW is not only a nets-within-nets editor and simulator: it allows the use of high level net concepts like arc inscriptions, transition guards, and coloured tokens. However, we only use a subset of the features of RENEW. In particular, we choose to model the system with a hypernet-like model [2] (we will discuss in section 5 why the system is not a proper hypernet). The system is modeled as a collection of *net instances*. Tokens are *references* to net instances. Therefore it is possible that a net has more than one reference (token) in the system which refer to it. Arc inscriptions contain single variables. When a transition is fired tokens are bound to these variables. Transition inscriptions may contain channel names, used by two or more nets when they need to synchronize. An *uplink* is used when a net wants to synchronize with the net above it in the hierarchy, a *downlink* is used when a net wants to synchronize with one of the reference tokens it contains.

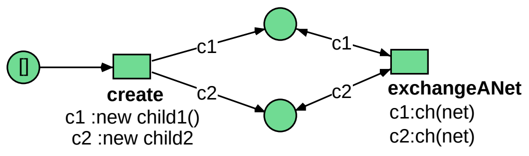
From a syntactical point of view the RENEW constructs we used in our model are the following:

– A net instance is created by a transition inscription of the form *var : new netname*, which means that the variable *var* will be assigned a new net instance of type *netname*.
– An *uplink* is specified as a transition inscription *:channelname (expr)*. It provides a name for the channel and a variable which is used for *vertical* communication between nets.
– A *downlink* has the form *netexpr :channelname (expr)* where *netexpr* is an expression that must evaluate to a net reference.

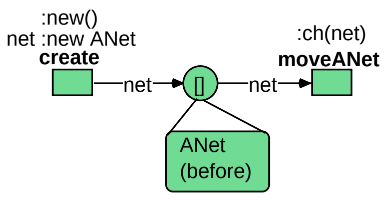
To fire a transition that has a downlink, there must be an input arc labelled with a proper variable name (*netexpr* for the previous downlink example), and this variable must evaluate to a net instance. The referenced net instance must provide an uplink with the same name,and it must be possible to bind the variables suitably so that the channel expressions evaluate to the same values on both sides. The parameter is bound to a variable present in one of the input arcs of the up(down)-link, and then it is bound to the parameter in the corresponding down(up)-link. Then the transitions can fire simultaneously.

The exchange of (structured) tokens between nets, typical of hypernets, is possible by means of parameters. Figure 1 shows an example. The only transition enabled at the beginning is *create* (Figure 1(a)), which creates an empty *child1* net, and a *child2* net (Figure 1(b), and Figure 1(c) respectively). The difference between using the parenthesis or not using the parenthesis in creating a new net is that, if you use them, then the transition that is being fired must synchronize on the channel *new()* in the child net. Therefore, transition *create* in the system net synchronizes with transition *create* in the child1 net, which creates the *ANet* net. Afterwards, transitions *exchangeNet, moveANet, receiveANet* can fire, moving *ANet* to *child2*.

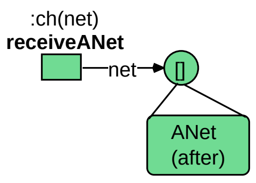
Let us notice that in our model the exchange of tokens between the two children nets, *child1* and *child2*, is made under the supervision of the system net. This means that the *system net* in some way observes the token exchange between its children.



(a) The system net



(b) The child1 net which creates a net of type *ANet* and sends it upward



(c) The child2 net which receives a net from above

Fig. 1. A simple example

3 The Application Context: Grid distributed analysis

The CMS experiment at CERN produces about 2 Petabytes of data to be stored every year, and a comparable amount of simulated data is generated. Data needs to be accessed for the whole lifetime of the experiment, for reprocessing and analysis, from a worldwide community: about 3000 collaborators from 183 institutes spread over 38 countries all around the world.

The CMS computing model uses the infrastructure provided by the Worldwide LHC Computing Grid (WLCG) Project [6] through the supporting projects EGEE, OSG and Nordugrid. Grid analysis in CMS is data driven. A prerequisite is that data is already distributed to some remote computing centers, and correspondingly published in the CMS data catalogue, so that users can discover available datasets. Parallelization is provided by splitting the analysis of large data samples into several jobs. The output data produced by the analyses are typically copied to the storage of a site and registered in the experiment specific catalogue. Small output data files are returned to the user. In the CMS experiment the CRAB tool set has been developed in order to enable physicists to perform distributed analysis over the Grid. The role of CRAB is to allow the user to run over distributed datasets the very same analysis she/he ran locally, and collect the results at the end. CRAB interacts with the distributed environment and the CMS services, hiding as much of the complexity of the system as possible. CMS community members use CRAB as a front-end which provides a thin client, and an Analysis Server which does most of the work in terms of automation, recovery, etc. with respect to the direct interactions with the Grid. The Analysis Server enables full workflow automation among different Grid middlewares and the CMS data and workload management systems. Indeed, the main reasons behind the development for the Analysis Server are:

– automating as much as possible the whole analysis workflow;
– reducing the unnecessary human load, moving all possible actions to server side, keeping a thin and light client as the user interface;
– automating as much as possible the interactions with the Grid, performing submission, resubmission, error handling, output retrieval, post-mortem operations;
– allowing better job distribution and management;
– implementing advanced use cases for important analysis workflows

The server architecture adopts a completely modular software approach. In particular, the Analysis Server is comprised of a set of independent components (purely reactive agents) implemented as daemons and communicating asynchronously through a shared messaging service supporting the "publish & subscribe" paradigm. Most of the components are themselves implemented as multi-threaded systems, to allow a multi-user scalable system, and to avoid bottlenecks. The task analyses are completely handled during their lifetime by the server through different families of components: there are components devoted to monitoring the Grid status of the single jobs in a task, other groups of agents coordinate to manage the output retrieval and the recovery of the failed jobs by scheduling their resubmission automatically. A relevant part of the agents is designed in order to handle the submission chain of user tasks to the Grid. As the Analysis Server internal architecture is a natural candidate for being analyzed with the nets-within-nets paradigm, as aforementioned, we decided to model and study the Grid submission chain. The aim of this study is to check that the involved agents behave correctly and efficiently with respect to the foreseen submission workflow. We decided to consider the system at the component-task-job level, as it represents a good compromise between the effects perceived by the tool final users and the large number of technical details that a complete representation of the Grid would require.

4 Modeling the submission use-case

In this Section we describe in detail the process of submitting jobs to the Grid through the CRAB Analysis Server. For each relevant component of the system its net representation is discussed. In addition, the bugs that have been discovered thanks to the net models are presented with the solutions that the actual code has adopted in order to solve the issues. The CRAB analysis suite was modeled using nets in a hierarchical fashion, as shown in Figure 2. A vertical line with multiplicity n, indicates the presence of n nets in the higher one (e.g.: the CRABClient net contains from 1 to N Task nets); a horizontal dashed line indicates that the linked nets are references to the same net. In our modeling we consider one client just for the purpose of simplicity. Of course, the discussed functionalities and use cases still hold when a larger number of clients

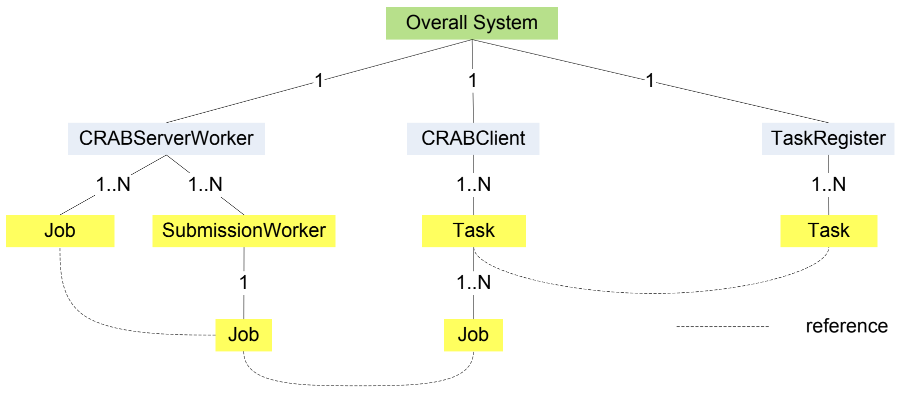


Fig. 2. The Nets hierarchy for the CRAB suite.

is considered, as the client server model assumes no direct interactions among the clients. In addition, for the use case that will be discussed, the server code separates properly the session of work for every task.

The OverallSystem net, which is the system net, contains three nets which respectively model the behavior of the client who is using the CRAB server (*CRABClient net*), the TaskRegister component which is a thread running on the CRAB server (*TaskRegister net*), and the CRABServerWorker which is also a thread running on the server (*CRABServerWorker net*). *Tasks* are the objects a client creates, and deals with. They are composed of *jobs*, the single units of work that need to be performed. The TaskRegister component is responsible for registering tasks, i.e. creating some data structures on server disks, checking if each task has all the inputs it needs to be executed, and checking if the Grid can access the proper security credentials to execute it. The CRABServerWorker component continuously receives jobs, schedules them for execution on the Grid infrastructure, and creates a SubmissionWorker thread which monitors the lifecycle of each job on the Grid. The clients interact with the server, and can initiate some operations like: submitting jobs, killing them if needed, and asking for the results.

4.1 CRABClient, Tasks, and Jobs

The first component we are going to discuss is the CRAB client, which is modeled with the net in Figure 3. This component is what enables all the action sequences that the users can do on their Grid analyses.

The first thing a client does is to create a new task on the client machine. The typical usage pairs a unique task with a CRAB analysis session. For this reason we assume that the *tasksPool* can contain a finite number of tokens. After the task has been locally created on the client machine, the client can perform a submit operation, which is of course the most important one as it starts the submission chain. The first time a task is submitted to the server, it is also regis-

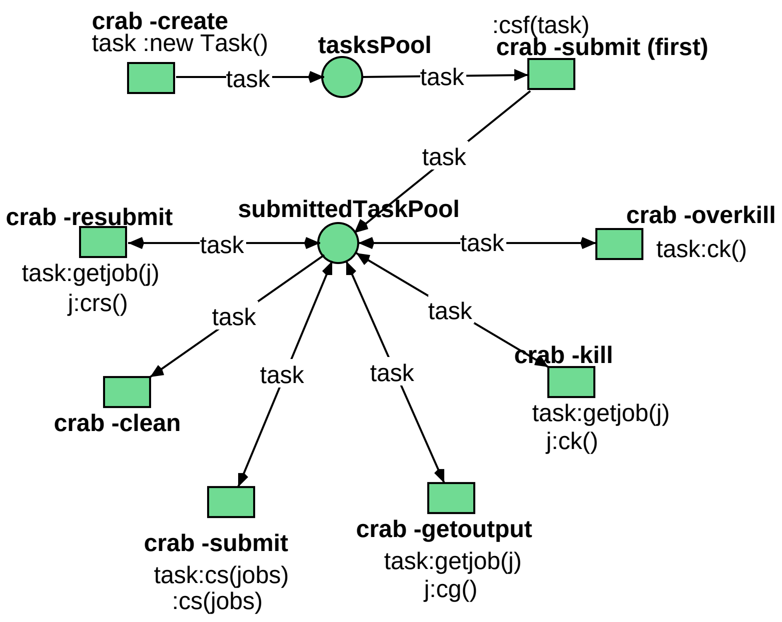


Fig. 3. The CRABClient net.

tered by the TaskRegister component. Subsequent submits are handled directly by the CRABServerWorker component. In our model the difference between the two types of submits is modeled as two different transitions. In particular *crab -submit(first)* transition has an uplink (*:csf(task)*), which means that it must be synchronized with the upper level. As a result the task reference is copied to the TaskRegister component by the Overall System net. After creation, the main operations a user can do are submit, resubmit, kill, getoutput, and clean. All these operations require an interaction with the server, but since we have focused on the submission use case, these interactions have not been explicitly modeled. For example the *getOutput* command is modeled as an interaction between the client and the job by means of two inscriptions. Handling all the possible interactions between the actors involved in the system would have resulted in a very big model, making it impossible to describe in this paper.

A task, see Figure 4, is a bag of jobs (the system allows to collect up to 4000 jobs into a singe task) and it is a representation that CRAB uses to perform collective actions on the Grid processes. Places *notRegistered, registering, registered* of the Task net contain information about the state of a task itself. These places control the enabledness of transitions *crab -submitFirst*, and *taskRegistered*, which are respectively called by the CRABClient when a job is submitted, and by the TaskRegister component when the task has been successfully registered after a *submit first* operation. The submit transition is called when a

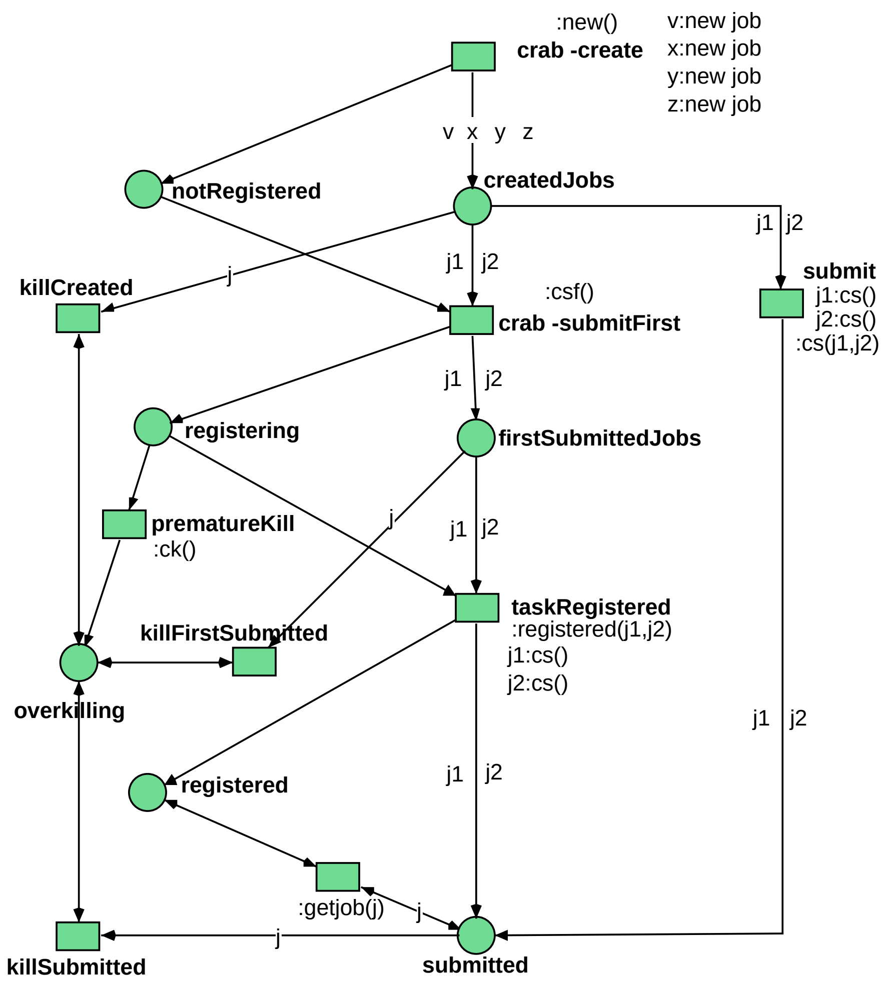


Fig. 4. The Task net. Only four jobs are considered in order to exemplify the relation with the job net.

CRABClient performs a *submit subsequent* action. In our model both *taskRegistered*, and *submit* transitions send upward two jobs through a synchronous channel, and make the job move to the submission request state.

The net representing the state of Grid jobs and their allowed actions is reported in Figure 5. This net has been modeled combining the finite state machine reported in the CRAB official documentation with the information extracted directly from the portion of code devoted to the Grid job state handling. Several transitions of this net contain uplinks, and therefore have to be synchronized with some other net. Transitions with a *:crs()* uplink (CRAB Resubmit) are

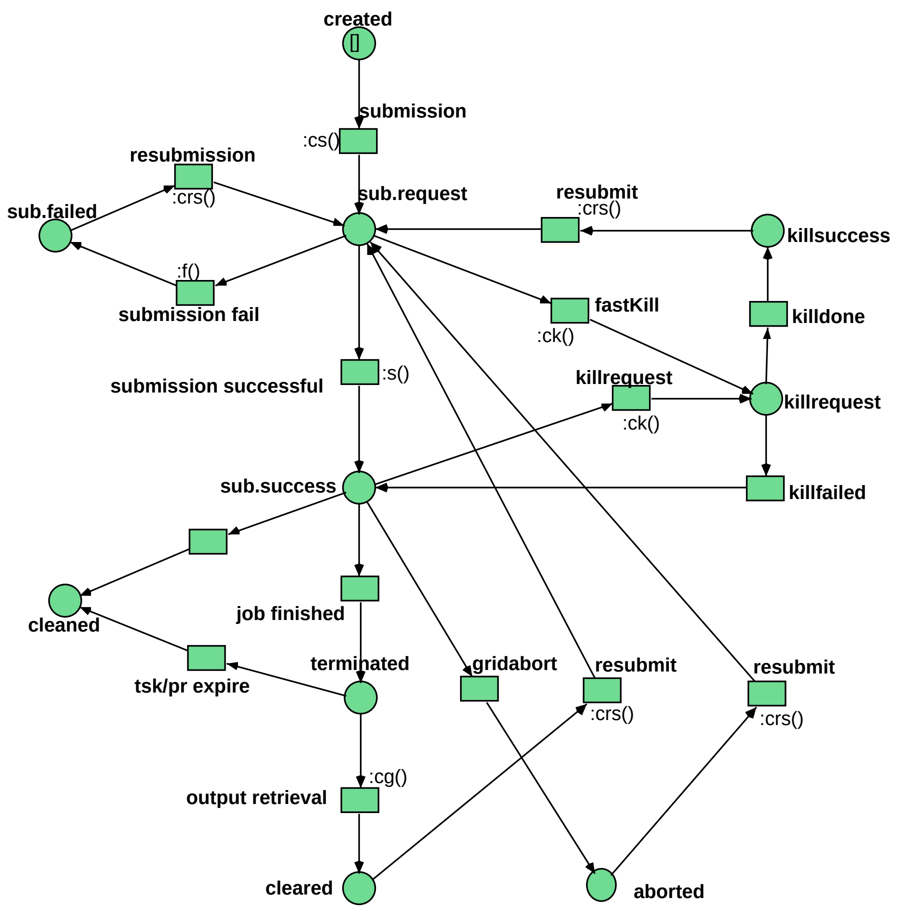


Fig. 5. The Job net.

transition enabled only if the job is in a state where a resubmit is possible, and are synchronized with the *crab -resubmit* transition of the CRABClient net, or the *resubmit* transition of the SubmissionWorker net. In the same way killings (channel *:ck()*), failures (channel *:f()*), submission (channel *:s()*), and output retrieving (channel *:cg()*), have to be synchronized with a correspondent transition in another net.

The integration of the documentation and the code with the formalism of the nets has allowed us to identify a bug in the way job states are modified. In particular, the net allows some transitions that are not actually activated by any event observed by the system (bug 1, b1). For example let us consider the unlabeled transition between the *sub.success* and the *cleaned* places in Figure 5: the latter denotes that a job has been abandoned because the user security credentials are expired and the Grid will not manage processes whose owner cannot be recognized. A malicious code interacting with the clients in place of the proper server could move jobs arbitrarily to this terminal state. The fix for this bug consisted in a review of the code managing the job state automata in accordance with what is stated by the presented Job net. Also, the pre-conditions that allow a client to perform a *kill* request over the jobs are not granted properly (b2): jobs can be killed when they are in states where the killing is dangerous. For example, a user could run into a condition where a failed job cannot be resubmitted as the system requires to kill it. That means the job is in a deadlock, as a failed job cannot be killed on the Grid.

4.2 TaskRegister

The TaskRegister component, shown on the left of Figure 6, duplicates the task and jobs structures that have been created at the client side and alters all the object attributes in order to localize them with respect to the running environment of the server, taking care also of security issues (like user credentials delegation) and files movement (check the existence of input). We modeled this cloning by means of the *reference semantics*: the TaskRegister component receives from the client a copy of the reference which points to the Task.

The component is able to handle more tasks simultaneously thanks to a pool of threads implementing the net of Figure 6. The first transition that is fired is *submission*, which is synchronized with the transition in the system net that receives the task reference from the CRABClient. Then four operations which can fail are executed on the task. These include local modification of the task with respect to the server environment, the user's credential retrieval (also known as delegation), the setting of the server behavior according to what the credentials allow to do and, finally, the checking that the needed input files are accessible from the Grid. If the registration fails the only possible operation available is *archiveTask* which deletes the reference to the task from the task register component. If the user has the privileges to execute the jobs in the task, and if the inputs needed by the task are available, then a range of jobs is selected from the task and passed to the CRABServerWorker by firing the *toCSW* transition (again under the supervision of the system net). The modeling and the simulation of the TaskRegister net has highlighted some relevant defects and bugs. In case of failure the TaskRegister component was not able to set properly the status of the jobs in a task to fail. This macroscopic lack in the system design implied different side effects. The server was not able to discriminate whether to retry automatically the registration process or to give up and notify the user about the impossibility to proceed (b3). In addition, the system could not tell if the registration has been attempted previously. This implies that the client transfers the input data every time a registration failure appears, with a waste of network resources (b4). Both the defects have been solved by introducing the proper synchronization between the *fail* transition in the component with *submission failed* in the job net. Mapping the synchronization into the server code has granted that the status of the jobs is set to the correct failure state and that

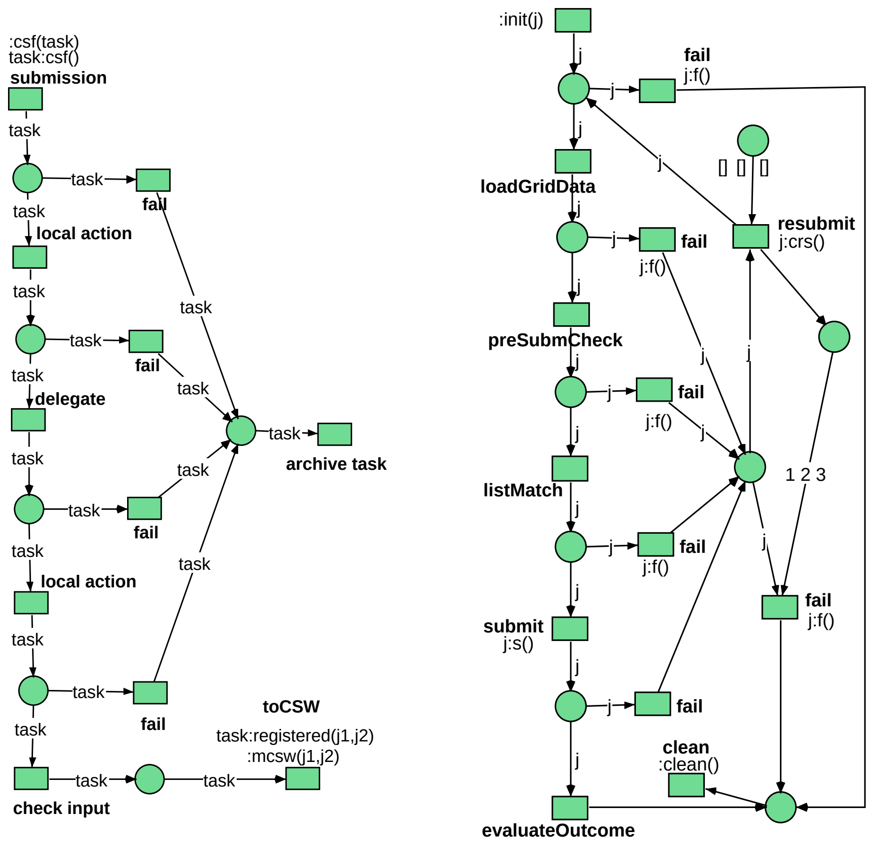


Fig. 6. TaskRegister and SubmissionWorker nets respectively

the submission counters are properly incremented (being implementative details the counter is not reported in the Job net). With this modification the server becomes aware that a first try has been executed and also network transfers are exploited more efficiently. A second bug has been identified thanks to the study of the synchronization among the transitions for the client, the jobs and the TaskRegister nets. In detail, the handling of the *kill* commands presents some issues. If a user requires to kill some jobs while the task is being registered, the system cannot distinguish properly which jobs have to be killed and therefore it applies an over-killing strategy by halting the whole task (b5). This happens because the code performs some sort of synchronization with the Task net instead of having rendezvous with the related transitions into the lists of killing jobs.

The killing of Grid jobs is a demanding action, both in terms of network communications and in terms of coordination among the different services involved in a Grid. Furthermore the killing of an analysis job is a permitted but infrequent action. For these reasons the CRAB developers have decided to suppress this early job termination feature in order to avoid the bug. Now users are allowed to kill jobs only once they have been actually submitted to the Grid.

4.3 CRABServerWorker, and SubmissionWorkers

In our model the result of a submit operation is that the CRABServerWorker component, shown in Figure 7, receives a structured token in the place *accepted*. If the submit was the first, transition *newTaskRegistered* is fired after the task has been registered by the TaskRegister component by means of transition *toCSW*, which is synchronized with transition *newTaskRegistered* through the overall system. If the submit is not the first, the task has been already registered, therefore transition *subsequentSubmission* is fired. After receiving the range of jobs, the CRABServerWorker component schedules these jobs for the execution on the Grid infrastructure. The practical effect of this component is to break the task into lists of jobs in order to improve the performance thanks to bulk interactions with the Grid middleware. The Submission Worker thread spawned by the component monitors the actual submission process of the jobs. We have modeled this fact by creating a Submission Worker net for each one of the jobs in the list. Indeed, transition *triggerSubmissionWorker* creates a new Submission Worker assigned to the variable *sw* and synchronizes it with a transition labeled *init*.

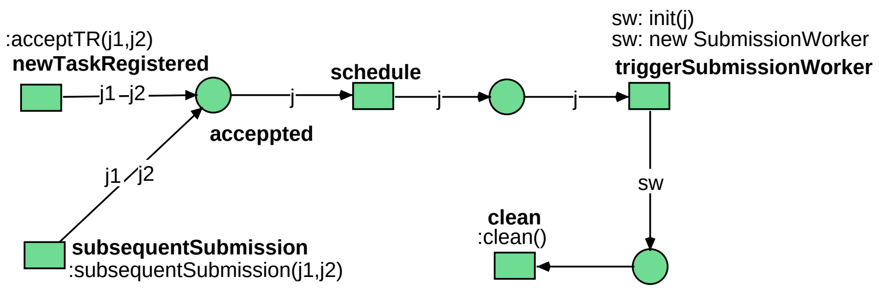


Fig. 7. The CRABServerWorker Net

The thread is responsible both for tracking the submission to the Grid infrastructure, and for resubmitting jobs when a failure occurs. Failures can occur for different reasons: network communication glitches, unavailable compatible resources, etc. Some types of failures are recoverable and in those cases the Submission Worker automatically tries to resubmit the job a three times. This value can be configured in the code, but in the model we only used the actually employed value of three. If the failure persists the job is permanently marked as failed. The net shown on the right in Figure 6 is our model of the submission worker component.

The study of the synchronization between the job and the Submission Worker nets allowed us to identify another bug in the code. The *submission success* transition in the job net (Figure 5) synchronizes with the *submit* Submission Worker's transition (right of Figure 6). This means that the CRAB Server marks the submission as successful just after the interactions with the Grid. Actually the network latencies could delay the propagation of the job failure message (b6) and, therefore, the correct rendezvous should be enacted between *submission success* and *evaluateOutcome*.

It is relevant to observe that the approach followed for the modeling of the CRAB Server submission chain is a particular case for a quite general class of Grid systems. All the Grid middlewares rely on jobs that are represented by finite state automata and that are concurrently managed by the different services involved in the Grid. In addition, the intermediate action of a broker like the CRAB Server is becoming a common pattern with the diffusion of scientific gateways: programmatic portals that abstract the user applications from the complexities of the distributed infrastructures acting as back end.

The adoption of the nets-within-nets paradigm has provided a natural and effective way to model subtle interactions among the different net levels. It would have required a significantly greater effort to discover the same problems with a flat net approach. In the following subsection details about the process of deriving the models from the documentation and the code are given.

4.4 Details on the model derivation process

The model was derived from the code by analyzing both the official documentation and the source code of the system. The Job net is directly built from the documentation. A finite state automata which describes the Job is reported explicitly. After that, simply by using pattern matching we analyzed the source code relevant for the submission use case by searching for interaction with jobs. Each source module is modeled as a net (e.g.: CRABClient, TaskRegister, CRABServerWorker etc), and the interactions with the Job nets are modeled using the RENEW uplink/downlink mechanism. A modification of the status of a job in the code is modeled as a pair of synchronized transitions in the model itself: one in the job net and one in the net that models the component changing the job status.

To ensure that the model is an accurate representation of the software, we made several task submissions with the CRAB tool and monitored the status of the jobs during the evolution. The request parameters were set up so that different behaviours of the system are tested. For example, jobs lacking of input files, job submitted by users with expired credentials, and jobs killed before the completion of task registration process are test cases that have been considered. After that, we simulated each submission on the model, taking care that the simulation of the status of the job net was consistent with the actual job status in the system.

5 Discussion

In the study we have just presented, a formal approach was used to validate a system that has already been implemented. Simulating the behavior of the system by means of a computer aided tool was what allowed us to find problems in the implementation of the CRAB server. However, another great advantage of modeling a system with formal methods is the possibility to apply *automatic* analysis techniques to extract information about the system, like invariant analysis, and model checking.

In order to apply some of these techniques, the formal model must respect specific prerequisites. For example, most algorithms for model checking a concurrent system require a bounded state space. Nets-within-nets models which satisfy this last requirement are hypernets [2] and their generalization [4], which can both be expanded to 1-safe Petri nets [3, 18]. This expansion guarantees the possibility of applying all the analysis techniques of this well known class of Petri nets to hypernets.

The first idea was to use such a class of nets to model the CRAB server, but because of the absence of modeling limitations and verification features in RENEW, and because of the high complexity of the system, we preferred to use a slighty more powerful version of hypernets. To come back to the class of hypernets, having therefore the certainty that the state space is limited, the following fixes are necessary:

- Transitions which create or delete tokens must be deleted in some way. For example, transition *crab -create* of the CrabClient net cannot create an unbounded number of tasks anymore, but an input place which contains as many tokens as the maximum number of allowed tasks must be added.
  This is not a big problem. As a matter of fact the computers disks space is limited, and consequently so are the number of tasks which can be created by a user.
- Hypernets use a *value semantics*, which means that a net cannot have two references to it. Nevertheless, in our model some transitions duplicate the references to a net. Duplication of references is somehow dangerous if the intention is to keep the state space bounded. Loosely speaking, the risk is of an uncontrolled grow of the references of a net without a corresponding deletion of these references. In our model the use of the value semantics can be achieved by deleting these duplications of references, and using simple tokens to communicate the intention to modify the referenced net.

Even though analysis of properties is not available with the current version of the model because of the issues just discussed, the more practicality of the reference semantics from a modeling point of view helped us finding several design defects in the implementation of the CRAB server. In the future we plan to restrict the model to a hypernet in order to be able to verify properties like invariants, or to do model checking [1]. In our opinion, as a first step it was

[1] Restricting the model to hypernets is not the only way to have a limited state space, but a formal proof is available using hypernets thank's to the 1-safe expansion

important to use a powerful formalism to avoid getting lost in the details of the model, even though that meant sacrificing the analysis capabilities.

6 Conclusions

In this paper, we discuss a large scale Grid application used to perform distributed data analysis in High Energy Physics experiments. Because of the complexity of the architecture, the software tool has been modeled using the nets-within-nets paradigm in order to validate the correctness of its behavior using simulation. In particular we considered the fundamental use case of the submission of user data analysis to the Grid. Every component of the CRABServer involved in this use case has been modeled in the hierarchy of the nets and compared to the behavior expected by its users.

From the simulation of the model a number of bugs and design defects emerged. This has led the developers to improve the overall quality of system implementation in the subsequent releases that the users now adopt. Two groups of bugs have been identified: bugs related to wrong coding of the expected behaviors and bugs where the specific adoption of nets-within-nets formalism has highlighted synchronization problems among the entities .

In addition, the approach followed to model the CRAB tool set has shown its generality in order to model most of the Grid applications in which an orchestration entity drives the nets representing both the finite state machines of the jobs running on the distributed infrastructure and the services exposing the resources themselves.

The class of nets used to model this system is a more powerful version of hypernets, using the reference semantics instead of the value semantics, and allowing creation/deletion of tokens. As discussed in Section 5, it is possible to restrict the model to a proper hypernet by sacrificing its readability (some places and transitions must be added). Then, by means of hypernets and their expansion to 1-safe nets, it will be possible to use all the techniques defined for the class of 1-safe nets for analyzing the system.

A plugin of RENEW that allows to draw and to analyze a hypernet is being developed. We plan to use this plugin to make automatic verification of properties of the system.

References

1. Martin Alt, Andreas Hoheisel, Hans Werner Pohl, and Sergei Gorlatch. A Grid Workflow Language Using High-Level Petri Nets. In *Procs of the 6th Int. Conf. on Parallel Processing and Applied Mathematics: PPAM05*, pages 715–722, 2005.
2. Marek A. Bednarczyk, Luca Bernardinello, Wiesław Pawłowski, and Lucia Pomello. Modelling mobility with Petri Hypernets. In *Recent Trends in Algebraic Development Techniques*, volume 3423/2005 of *Lecture Notes in Computer Science*, pages 28–44. Springer Berlin / Heidelberg, 2005.

3. Marek A. Bednarczyk, Luca Bernardinello, Wiesław Pawłowski, and Lucia Pomello. From Petri hypernets to 1-safe nets. In *Proceedings of the Fourth International Workshop on Modelling of Objects, Components and Agents, MOCA'06, Bericht 272, FBI-HH-B-272/06, 2006*, pages 23–43, June 2006.
4. Luca Bernardinello, Nicola Bonzanni, Marco Mascheroni, and Lucia Pomello. Modeling symport/antiport p systems with a class of hierarchical Petri nets. In *Membrane Computing*, volume Volume 4860/2007 of *Lecture Notes in Computer Science*, pages 124–137. Springer Berlin / Heidelberg, 2007.
5. Carmen Bratosin, Wil van der Aalst, and Natalia Sidorova. Modeling Grid workflows with Coloured Petri nets. In *Procs. of the 8th Workshop on Practical Use of Coloured Petri Nets and CPN Tools: CPN 2007*, pages 67–86, 2007.
6. CERN. Worldwide LHC Computing Grid. http://lcg.web.cern.ch/lcg/public/. Accessed May, 2010.
7. Giuseppe Codispoti, Mattia Cinquilli, Alessandra Fanfani, Federica Fanzago, Fabio Farina, Carlos Kavka, Stefano Lacaprara, Vincenzo Miccio, Daniele Spiga, and Eric Vaandering. CRAB: a CMS Application for Distributed Analysis. *IEEE Transactions on Nuclear Science*, 56(5):2850–2858, 2009.
8. Jordi Cortadella and Wolfgang Reisig, editors. *Applications and Theory of Petri Nets 2004, 25th International Conference, ICATPN 2004, Bologna, Italy, June 21-25, 2004, Proceedings*, volume 3099 of *Lecture Notes in Computer Science*. Springer, 2004.
9. Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
10. Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid services for distributed system integration. *Computer*, 35:37–46, 2002.
11. Zhijie Guan, Francisco Hernandez, Purushotham Bangalore, Jeff Gray, Anthony Skjellum, Vijay Velusamy, and Yin Liu. Grid-Flow: a Grid-enabled scientific workflow system with a Petri-net-based interface: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18:1115–1140.
12. Kathrin Hoffmann, Hartmut Ehrig, and Till Mossakowski. High-level nets with nets and rules as tokens. In Gianfranco Ciardo and Philippe Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 268–288. Springer, 2005.
13. Andreas Hoheisel and Uwe Der. Dynamic Workflows for Grid Applications. In *Procs. of the Cracow Grid Workshop 03*, page 8, 2003.
14. Michael Köhler and Berndt Farwer. Object nets for mobility. In Jetty Kleijn and Alexandre Yakovlev, editors, *ICATPN*, volume 4546 of *Lecture Notes in Computer Science*, pages 244–262. Springer, 2007.
15. Michael Köhler and Heiko Rölke. Properties of object Petri nets. In Cortadella and Reisig [8], pages 278–297.
16. Olaf Kummer. *Referenznetze*. Logos-Verlag, 2002.
17. Olaf Kummer, Frank Wienberg, Michael Duvigneau, Jörn Schumacher, Michael Köhler, Daniel Moldt, Heiko Rölke, and Rüdiger Valk. An extensible editor and simulation engine for Petri nets: Renew. In Cortadella and Reisig [8], pages 484–493.
18. Marco Mascheroni. Generalized hypernets and their semantics. In *Proceedings of the Fith International Workshop on Modelling of Objects, Components and Agents, MOCA'09, Bericht 290, 2009*, pages 87–106, September 2009.
19. The CMS Collaboration. The CMS Experiment at CERN LHC. *J. Inst.*, 3:S08004, 2008.

20. The TLS Group. The Large Hadron Collider Conceptual Design. Technical report, CERN, 1995. Preprint hep-ph/0601012.
21. Rüdiger Valk. Nets in computer organization. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume Volume 255/1987 of *Lecture Notes in Computer Science*, pages 218–233. Springer Berlin / Heidelberg, 1987.
22. Rüdiger Valk. Petri nets as token objects: An introduction to elementary object nets. In Jörg Desel and Manuel Silva, editors, *ICATPN*, volume 1420 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 1998.
23. Rüdiger Valk. Object Petri nets: Using the nets-within-nets paradigm. In *Lectures on Concurrency and Petri Nets*, volume 3098/2004 of *Lecture Notes in Computer Science*, pages 819–848. Springer Berlin / Heidelberg, 2004.
24. Kees M. van Hee, Irina A. Lomazova, Olivia Oanea, Alexander Serebrenik, Natalia Sidorova, and Marc Voorhoeve. Nested nets for adaptive systems. In *ICATPN*, pages 241–260, 2006.

Verifying Reference Nets By Means of Hypernets: a Plugin for Renew

Marco Mascheroni[1], Thomas Wagner[2], and Lars Wüstenberg[2]

[1] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano Bicocca
Viale Sarca, 336, I-20126 Milano (Italy)**
mascheroni@disco.unimib.it
[2] University of Hamburg,
Faculty of Mathematics, Informatics and Natural Sciences,
Department of Informatics
http://www.informatik.uni-hamburg.de/TGI/

Abstract. In this paper we examine ways to verify reference nets, a class of high level Petri nets supported by the Renew tool. We choose to restrict reference nets to hypernets, another nets-within-nets model more suitable for verification purposes thanks to an expansion toward 1-safe Petri nets. The contribution of the paper is the implementation of such analysis techniques by means of a Renew plugin. With this plugin it is now possible to draw, and to analyze a hypernet. The work is demonstrated by means of a simple example.

Keywords: Verification, High-level Petri nets, Reference nets, Hypernets

1 Introduction

The verification of properties of a software system has become an important part of software engineering. Especially specifications critical to the correct execution of a software system need to be verified in order to guarantee them after deployment. The problem of verification is its complexity and the effort required for it. Without proper methods the verification itself is difficult, costly and time-consuming.

In this paper we approach the general problem of verification with the help of Petri nets. The formalism is deeply rooted within established theoretical and formal methodologies, as well as being supported by a multitude of tools and analysers. Petri nets have been studied in detail and contain properties, for which established verification techniques exist. Using Petri nets the general approach is to map and translate specific software issues and properties to these Petri

** Partially supported by MIUR, and DAAD net properties. These properties can then be verified using the known Petri net techniques. Assertions made for these can then be translated back for the software behind it.

High level nets, Petri net models enriched with additional abstraction constructs, are well suited to represent complex systems due to their high abstraction constructs. One of their problems is that verification of their properties is difficult. Properties which are computable with low-level formalisms become undecidable, and thus cannot be verified anymore in some high-level models. However, high-level formalisms can be restricted in some way so that they can be translated into low-level formalisms, which in turn can be verified again. In particular, the interest of this paper is on high level nets which use the nets-within-nets paradigm, formalisms in which the tokens of a Petri nets can be structured themselves as Petri net. The two formalisms analyzed are *reference nets*, the formalism used as a basis for the Renew tool, and *hypernets*, another nets-within-nets formalism with particular restrictions that allow the expansion toward an equivalent 1-safe Petri nets. In this paper we will show how to translate a subset of the high-level reference net formalism into hypernets, which in turn can be easily translated into 1-safe nets. These can then be analysed by existing toolsets. The main result of our work is the implementation of a Renew hypernet plugin which incorporates features for computing S-invariants, and features for model checking a hypernet. As far as we know, this is the first time that analysis techniques typical of Petri nets has been implemented in a tool which support the nets-within-nets paradigm, and it is mature enough to be used in a real application context. In the rest of the paper when we will talk about invariants we are always referring to S-invariants.

The paper is structured into the following sections. Following this introduction the theoretical and technical background is shortly discussed in section 2. This section will focus on the reference net and hypernet formalisms. In section 3 we will show how to translate reference nets into hypernets and determine the prerequisites for analysis. Section 4 describes the Hypernet plugin created for Renew. Section 5 gives a short example how these different tools are incorporated and used to analyse a given net. The conclusion of the paper is found in section 6.

2 Background and related work

In this section we will introduce by means of examples the basic theoretical formalisms used in this paper, as well as motivate why we have chosen them as our means of verification and modelling. The interested reader can find them in the cited references. In general Petri nets offer a simple way of modelling concurrent behaviour of a system. Higher level nets often introduce abstractions from the simple net models, which offer structures and methods not available to or difficult to model in low-level Petri nets. One major such abstraction is the idea of nets within nets, introduced in [11] for Object Petri nets. This paradigm allows for arbitrary nets to be the tokens of other nets. In this way it is possible to model the behaviour and interaction of different entities within a complex system, all modelled with Petri nets. Using these formalisms to model and even implement software systems is quite natural. Of course high-level Petri nets and especially formalisms following the nets-within-nets idea are far more complex then the relatively simple low-level Petri nets. This increases the effort and complexity of verifying properties within these nets, which is the main motivation of this paper.

In the following subsections we will describe the reference and hypernet formalisms.

2.1 Reference Nets and Renew

The reference net formalism serves as the starting point of our examinations. It was described in [7]. It is a high level Petri net formalism based on the nets-within-nets paradigm. In this formalism it is possible for tokens within a net to be almost arbitrary objects and especially other Petri nets. Nets can then be used like tokens within their respective so-called system net, but it is also possible to let nets of different layers communicate with one another. The reference net formalism uses reference semantics. This means that tokens within a net do not exclusively correspond to their object/net (value semantics), but only reference their object/net. As a result of this multiple tokens can refer to the same object. This makes it possible to express complex systems in a natural way.

Communication between different net instances within the reference net formalism is handled via synchronous channels, based on the concepts proposed in [5]. Synchronous channels connect two transitions during firing. Transitions inscribed with a synchronous channel can only fire synchronously, meaning that both transitions involved have to be activated before firing can happen. During firing arbitrary objects can be transmitted bidirectionally over the channel. While the exchange of data is bidirectional there is a difference in the handling of the two transitions. The transition, or more accurately the inscription of the transition, initiating the firing is called the *downlink*. The downlink must know the name of the net in which the other transition, the so-called *uplink*, is located. The inscription of the downlink has the form *netname:channelname(parameters)*, in which the parameters are the objects being send and received during firing. If the downlink calls an uplink located in the same net the net name is simply replaced by the keyword *this*. The uplink's inscription is similar, but looses the net name, so that it has the form *:channelname(parameters)*. Uplinks are not exclusive to one downlink and can be called from multiple downlinks, so that this construct can be used in a flexible way. It is also possible to synchronise transitions over different abstraction levels. While during firing one downlink is always linked to just one uplink, it is possible to inscribe one transition with multiple downlinks, so that more than two transitions can fire simultaneously.

Figure 1 shows a simple example of a reference net system. The example was modelled using the Renew tool, which will be described later. It models a producer/consumer system, which holds an arbitrary number of producers and consumers. The system consists of three kinds of nets: the system net, the

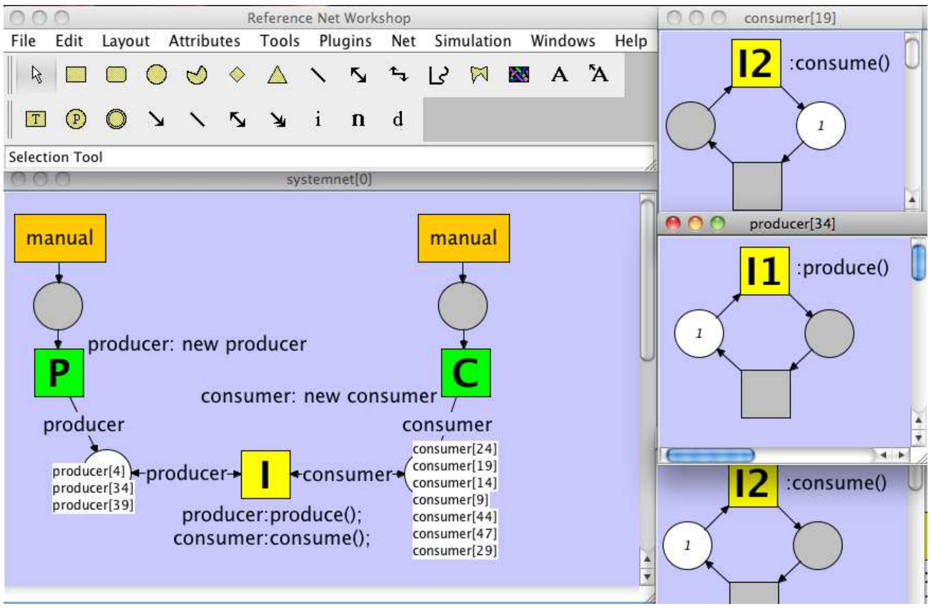


Fig. 1. Reference net example

producer nets and the consumer nets. The producer and consumer nets both possess the same basic structure, but use different channels. The system net serves as a kind of container for the other nets. The transitions labeled **manual** initiate the creation of new producers and consumers by creating new tokens when a user manually fires them during simulation[3]. The transitions labeled C and P actually create new producer or consumer nets when firing. These new nets are put onto the places below the transitions. The transition labeled I synchronises the firing of a transition in one consumer and one producer each (labeled I1 and I2 in the other nets). In this way it is possible to simulate the behaviour in such a way, that whenever a producer produces a product an arbitrary consumer consumes it. It is of course possible to enhance this model by, for example, adding an intermediary storage, which can store items from arbitrary producers until consumers need them. Another way of making the model more realistic is to explicitly model the products as nets as well. That way they would not just be simple tokens but actual objects being exchanged via the synchronous channels between the producers and consumers. In this case the parameters of the channels would be the nets, which would be transmitted from within the producer nets into the consumer nets.

The Petri net editor and simulator Renew (The **RE**ference **NE**t **W**orkshop) was developed alongside the reference net formalism, and is also described in [7] as well as in [8]. It features all the necessary tools needed to create, modify, simu-

[3] This is a special function of the Renew tool, which was used for this example.

late and examine Petri nets of different formalisms. It is predominantly used for reference nets, but can be enhanced and extended to support other formalisms. It is fully plugin based, meaning that all functionality is provided by a number plugins that can be chosen, depending on the specific needs. Plugins can encapsulate tools, like a file navigator or certain predefined net components, or extensions to the standard reference net formalism, like hypernets or workflow nets. RENEW is freely available online and is being further developed and maintained by the Theoretical Foundations Group of the Department for Informatics of the University of Hamburg. Since the tool supports the idea of nets within nets and is flexible enough to support multiple formalisms, it was chosen as the basic environment for the examinations of this paper.

2.2 Hypernets

As we will discuss later in section 3, we introduce hypernets in this paper because they have been used as a restriction of the reference nets formalism to allow property verification in RENEW. Hypernets are a nets-within-nets formalism introduced to model systems of mobile agents [2]. After their introduction several studies has been conducted on hypernets. In [3] it has been shown that it is possible to expand a hypernet in a 1-safe Petri net in such a way that the (hyper) reachability graph of the hypernet is equivalent to the reachability graph of the 1-safe net. In [1] a class of transition system, called agent aware transition systems, has been introduced to describe the behaviour of hypernets. In order to model a class of membrane systems, a generalisation of the hypernet formalism which relaxes some constraints of the basic formalism was introduced under the name of generalised hypernet in [4], and a theorem proving the existence of an expansion towards 1-safe nets for generalised hypernets was proved in [9].

Due to technical limitations in the RENEW tool only the basic version of the formalismi [3] has been implemented. Now we will informally discuss how hypernets work by means of an example. From a structural point of view a hypernet is a collection of (possibly empty) agents $\mathcal{N} = \{A_1, A_2, ..., A_n\}$, which are modelled as particular Petri nets. A state of a hypernet is obtained associating to each one of the A_i agents (nets), but one, a place p belonging to one of the other agents. That place will be the place which contains the agent A_i. This containment relation induces a hierarchical structure which by definition must be a tree. The root of the tree is the only agent which is not associated to any place (this agent is the system net).

The system evolves moving agents from place to place. A peculiar characteristic of hypernets is that the hierarchical structure is not static, but an agent can be moved from a place p belonging to an agent A_i, to a place q belonging to a distinct agent A_j. Another characteristic of hypernets is that agents cannot be created or destroyed. To ensure this "law of conservation of tokens" each net representing an agent is structured as a set of *modules* which have the structure of synchronised state machines, enriched with some *communication* places that allow the exchange of tokens between two agents close in the hierarchy. Agents and modules have a sort, and an agent can only travel along modules of the same sort.

In Figure 2, and Figure 3 the hypernet modelling a slightly modified version of the *one seater plane* case of study is drawn. This case of study has been introduced in [3], and models an airport in which planes can do basic things like landing, deplaning/boarding passenger, refuelling, and taking off. The changes we made in regards to the number of travellers in the example, the simplification of the safety refuel check and the part of the hypernet which makes sure a plain is empty when it is being refuelled.

To keep the example simple we considered a version with a plane which has only one seat. We choose to illustrate this example to show in an informal way how hypernets works. Moreover, in Section 5 we will show how it is possible to prove some properties of this simple example using the Renew plugin we developed.

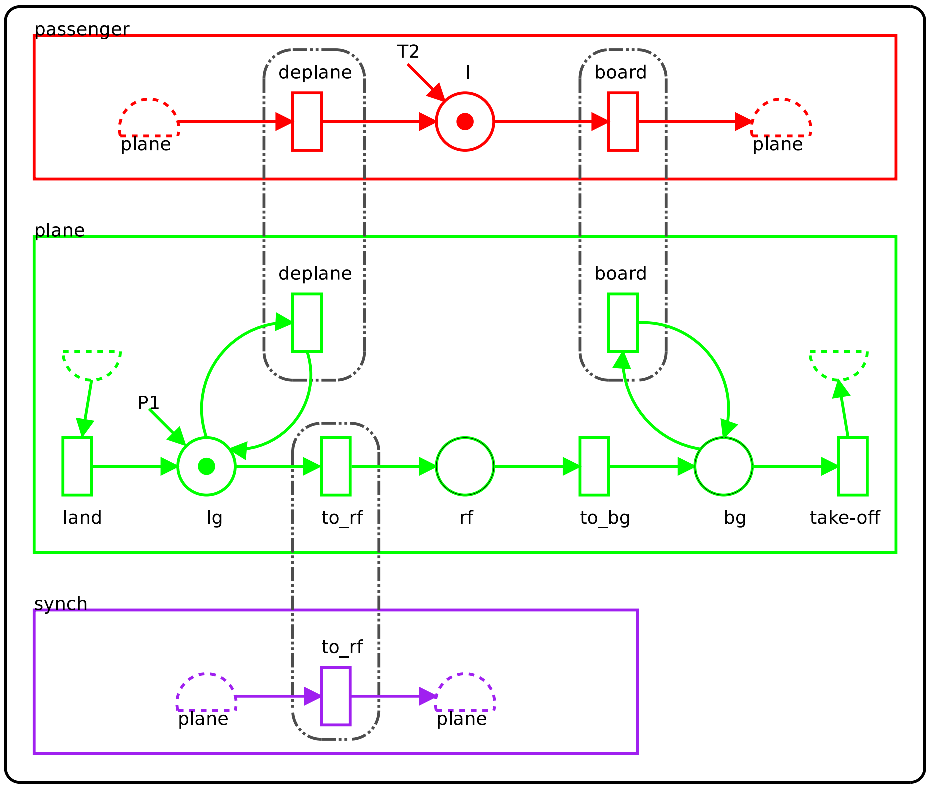


Fig. 2. Airport agent

The agent in Figure 2 models the behaviour of the airport. It has three modules, one for handling passengers, one for handling planes and one for synchronisation purposes. Transition *board* belongs to both module *passenger* and module *plane*, and can only be executed synchronously. The same applies for transitions *deplane* and *to_rf*. Communication places are the dashed half circles. They can either be *up-communication places*, used for communicating with the net at the level immediately above in the hierarchy (such as the two communicating places of the module plane in the airport agent), or *down-communication places*, used to communicate with an agent located in another module of the current net (such as the communication places in the synch, and passenger modules of the airport). In the latter case, a name of a module is provided. In this module there must be an agent ready to provide the *traveling* token which will be moved in the hierarchy, otherwise the transition is not enabled.

For example, transition *deplane* of the passenger module in Figure 2 has an input communication place which indicates that a token is expected. Since this communication place is marked with the *plane* annotation, the traveling token which is being moved to place l must be provided by a plane agent. This plane agent must be located in the input place of transition deplane in module plane of the airport, namely lg. In the example the only agent which can provide a token is $P1$. The traveling token, which must be a passenger, is then selected

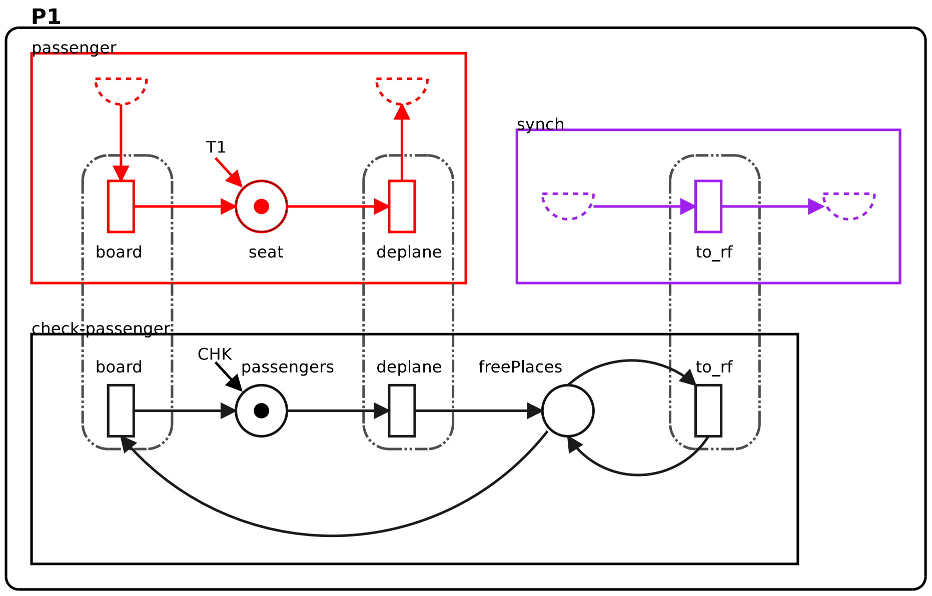


Fig. 3. The $P1$ plane agent shown in Figure 2

and taken from the *seat* place of the plane agent (Figure 3), and moved to l.

Transition *to_rf* is another example of use of communication places. From the airport perspective it is only required that an agent located in the *plane* module has a module *synch* containing with a transition *to_rf* preceeded and followed by two up-communication places. This requirement is fulfilled by agent $P1$, but from the $P1$ perspective it is also required the enabledness of the synchronized *to_rf* transition in the module check-passenger. Therefore this configuration *to_rf* is not enabled because *freePlaces* is not marked.

Hypernet being a high level net model means that the execution of a transition, like *deplane*, has several *firing-modes* [10]. Each firing-mode in a hypernet is a called *consortium*, and is obtained by selecting a transition, a set of agents that contain the transition, and a set of *passive* agents that will be moved as shown in the previous example when the consortium fires. For example, one enabled consortium is the one we just discussed which moves the agent $T1$ from place *seat* of the plane, to place *rf* of the airport agent that we just discussed. Another consortium is corresponding to agent $T2$, which in the configuration shown in the example is not enabled since $T2$ is not located in place *seat*.

One of the most important features of hypernets is that they have a straightforward expansion towards a behaviourally equivalent 1-safe nets. This expansion not only gives hypernets a precise semantics in terms of a well known Petri nets basic model, but also guarantees the possibility to reinterpret on hypernet all the analysis techniques developed for the basic model. The 1-safe net is built in the following way:

- For each agent A, and for each place p in the hypernet a place named $\langle A, p\rangle$ is added in the corresponding 1-safe net. A token in this place means that A is located in p,
- For each consortium Γ in the hypernet a transition named t_Γ is added in the 1-safe net,
- An arc is added from a place $\langle A, p\rangle$, to a transition t_Γ if A is a passive agent in Γ, and p is the input place from which the agent A comes.
- An arc is added from a transition t_Γ, to a place $\langle A, p\rangle$ if A is a passive agent in Γ, and p is the output place where the agent A is going to.

Finally, a place $\langle A, p\rangle$ of the 1-safe net is marked if in the initial configuration of the hypernet agent A is located in place p.

For example, the *one seater plane* case of study we just discussed is translated in the 1-safe net shown in Figure 4. Plane $P1$ can be in places lg, rf, bg in the hypernet, thus the 1-safe net contains places $\langle P1, lg\rangle$, $\langle P1, lg\rangle$,$\langle P1, lg\rangle$. The same must be done for traveler agents, and for the CHK check agent. Since transition *deplane* in the hypernet has two firing-modes, in the 1-safe net two transitions which models each of the firing modes of *deplane* are added (for simplicity both called *deplane*). The same has been done for transition *board*. The firing of a transition in the 1-safe net exactly models what happens when a consortium fires in the hypernet.

As already mentioned, it can be demonstrated that this net is 1-safe, and has a reachability graph isomorphic to the one of the corresponding hypernet. Details, formal definitions, and proofs discussed can be found here for hypernets in [3], and in [9] for the generalization version.

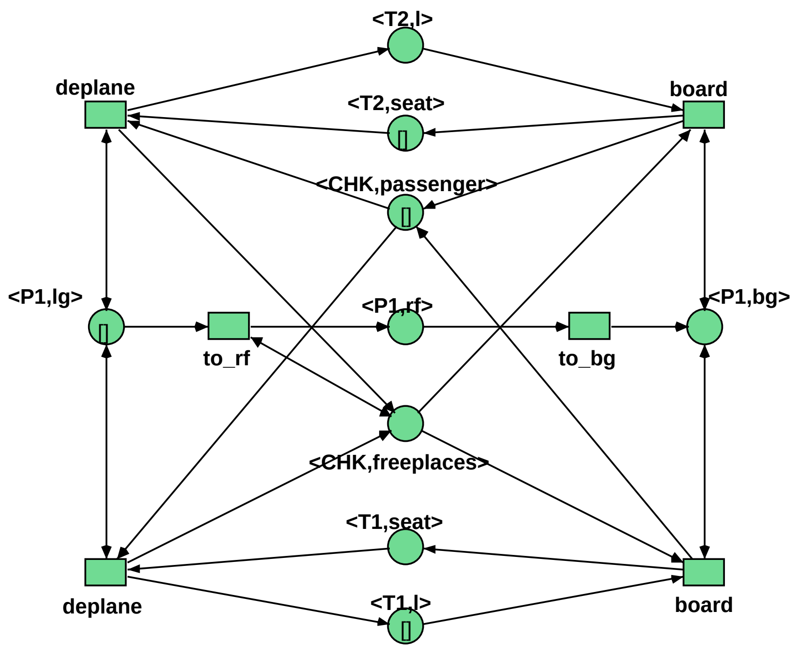


Fig. 4. The expansion toward 1-safe net of the hypernet in Figure 2 and Figure 3

3 Restricting Reference Nets to Hypernet

The main motivation for using high level nets is that, given a system, it is possible to obtain a model of the system with an high level net which is smaller compared to the model obtained using basic Petri nets. However, if you are not careful, the increase of the modelling power decreases the decision power of the model. For example, in [6] it was shown that, even considering a simple subclass of reference nets with one system net, and several references to an object net, the reachability problem becomes undecidable.

It is in this perspective that the implementation of the hypernet formalism as a plugin of the RENEW tool has been made. Restricting reference net is probably the most intuitive way to use verification techniques in RENEW. In particular, the use of a nets-within-nets formalism like hypernets as a restriction permits the use of the nets-within-nets paradigm, which is probably the most intresting feature in RENEW. The original contribute of the paper is to show how this plugin allows the use of verification techniques, like invariants and CTL model checking, to check properties of systems which are suitable to be modeled with the the nets-within-nets paradigm.

4 The Hypernet Plugin

From a technical point of view the implementation of a new formalism in Renew is done using a plugin mechanism. The most important method contained in the classes implementing the plugin is a *compile* method which takes as input a *shadow* net, a set of Java objects containing all the information about the net the user has drawn in the graphical editor of Renew, and transform it in a set of Java objects used by the simulator engine to simulate the net. This compile method is responsible for checking that the net drawn by the user is an actual hypernet in our case. In particular, in order to be able to use Renew as a hypernet simulator, the arc and transition inscriptions used in the modeling process must be restricted in such a way that the drawn net is a hypernet. Therefore the restrictions applied in the plugin are the following:

– Inscriptions (tokens) inside places can only be in the following forms: *identifier* or *identifier:netType*. In the first case the identifier represent the name of an empty net, and will be treated by the simulator engine as an black token; in the second case a new instance of the net *netType* will be created and placed inside the place.
– Inscriptions on arcs are restricted to single variables only. Each arc must contain exactly one variable inscription.
– The inscriptions of input (output) arcs must not be duplicated. In this way it is possible to preserve the identity of nets: duplication of tokens is forbidden.
– Balancing of transition has to be checked, i.e.: the set of variable names used to inscribe input arcs must coincide with the set of variable names used to inscribe output arcs.
– Communication places are deleted, and are simulated by means of synchronous channels. These channels are counted when checking transition balance.

For example, the airport agent shown in Figure 2 can be drawn as a hypernet in Renew using the net shown in Figure 5. The traveler empty tokens are place inscriptions $T1$ and $T2$, and the plane net instance is created by the $P1 : place$ inscription. Each transition is balanced. For example transition *deplane* in the airport has a bidirectional arc labelled *pl*, and an output arc labelled *pa* for which there is a correspondant downling, namely $pl : deplane(pa)$. Each communication place is deleted, and it is replaced with a synchronous channel. *Land* and *takeoff* transitions are equipped with two uplink because they were connected to two up-communication places. *Deplane* and *board* transitions contain two downlinks because they were connected to down-communicating places. The module name used to label communicating places is used to retrieve the variable name used in the downlink.

The $P1$ agent of Figure 3 is drawn in the hypernet plugin of Renew with the net in Figure 6. Again, up-communication places are replaced by channels, and transition *to_ rf* must synchronise with the corresponding transition in the airport agent.

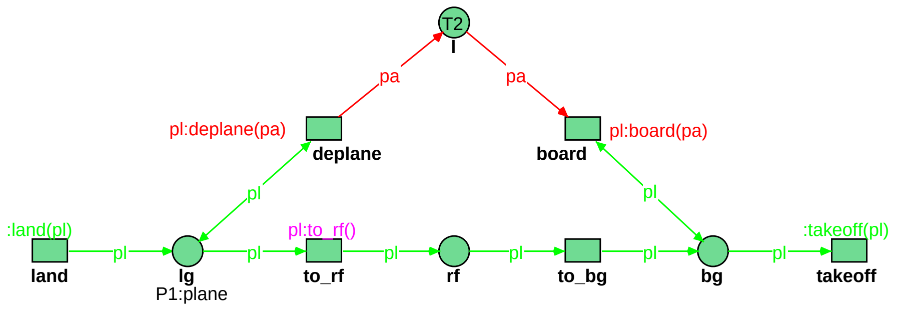


Fig. 5. The airport agent drawn with the hypernet plugin of Renew

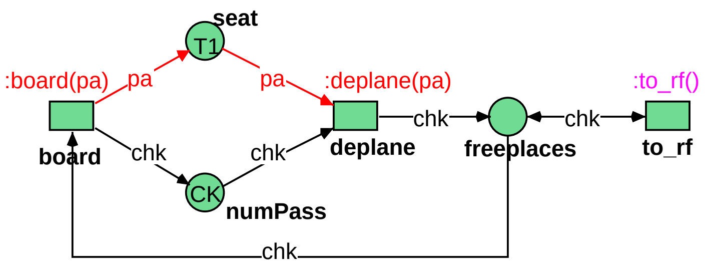


Fig. 6. The plane agent drawn with the hypernet plugin of Renew

As we already mentioned, thanks to the expansion to 1-safe nets it is possible to use verification techniques defined for this class of net to analyse system modelled with a hypernet. Two of the most useful techniques are invariants analysis, and model checking. We explored two possibilities of using them in the plugin we implemented: internal implementation in Renew, or exporting the 1-safe net in a format understandable by other tools. Since implementing these analysis techniques in an efficient way is a difficult task (some tools are very elaborated, and have been implemented over several years), and since very efficient open source tools are available for free, we decided to use external tools to implement invariant analysis, and model checking of a hypernet.

In the following sections we will show how the extensions and incorporation can be used in a practical example.

5 Example

The invariant analysis, and the model checking extensions we implemented in Renew can be used to prove properties of a system. We have chosen the external tools LoLA (see `http://www2.informatik.hu-berlin.de/top/lola/lola.html`) and INA (see `http://www2.informatik.hu-berlin.de/~starke/`

`ina.html`) for analysing purposes. Starting from the hypernet example of Section 2.2, we will prove using invariants that there is never more than one passenger on the plane, and we will prove using the model checker that a plane never refuels if there are a passenger on board.

By running the invariant analysis we get the following invariants:

T2@l	T2@seat	CHK@pass	P1@lg	P1@rf	P1@bg	CHK@freepl	T1@seat	T1@l
0	0	0	1	1	1	0	0	0
1	1	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0
0	0	0	0	0	0	0	1	1
1	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	1	0

The first four invariants are those which guarantee the truth of "law of conservation of agents", achieved thanks to the state machine decomposition in the formalism. For each agent there is a corresponding invariant indicating the places in which that agent can be located. Since the places of each invariant contains only one token in the initial marking, it is mathematically proved that each agent can be only in certain places: the places which are of the same sort of the agent itself. Moreover, these four invariants can also be used to prove that the net is 1-safe: they cover all places of the net, and contain only one token in the initial marking.

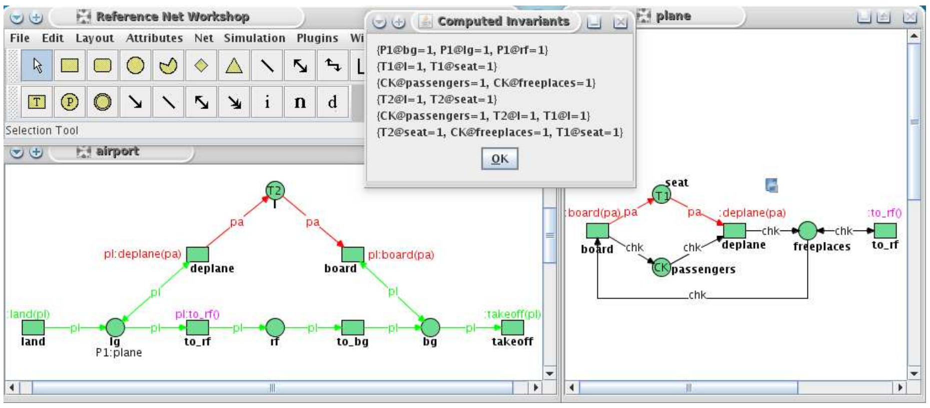
The fifth invariant is $\{\langle T2, l\rangle, \langle CHK, numPass\rangle, \langle T1, l\rangle\}$ and contains two tokens in the initial marking. Together with the second and the fourth invariants it can be used to prove that if the place $\langle CHK, numPass\rangle$ is marked then one of the two passenger is seated on the plane. The place is not marked only if both passenger are in the airport.

The sixth invariant is the counterpart of the fifth, and states that only one of the following places can be marked: $\{\langle T2, seat\rangle, \langle CHK, freeplaces\rangle, \langle T1, seat\rangle\}$. The information is clear: only one passenger can be in the *seat* place of the plane. If none of them is in the plane $\langle CHK, freeplaces\rangle$ is marked.

In Figure 7 a screenshot of Renew after the computation of invariants is shown.

While invariants analysis can be launched, and the computed invariants can be analysed to extract information about the system, in order to analyze the system using model checking a formula specified in a temporal logic is needed. Since we choose LoLA, which is a CTL model checker, we need to specify the property we want to verify using this logic. For example, checking the property "if the plane is located in the place representing the refueling station then no passenger is on board" can be done by entering as input of the Renew plugin we implemented the following CTL formula:

$$\begin{array}{l} ALLPATH\ ALWAYS \\ NOT\ ((T1.seat = 1\ AND\ P1.rf = 1)\ OR\ (T2.seat = 1\ AND\ P1.rf = 1)) \end{array}$$



```
{P1@bg=1, P1@lg=1, P1@rf=1}
{T1@l=1, T1@seat=1}
{CK@passengers=1, CK@freeplaces=1}
{T2@l=1, T2@seat=1}
{CK@passengers=1, T2@l=1, T1@l=1}
{T2@seat=1, CK@freeplaces=1, T1@seat=1}
```

Fig. 7. A screenshot of the invariants computed inside Renew

The formula checks that in every reachable state ($ALLPATH\ ALWAYS$) the situation in which both placed $\langle T1, seat\rangle$ and $\langle P1, rf\rangle$ are marked never occurs (and the same for places $\langle T2, seat\rangle$ and $\langle P1, rf\rangle$). The analysis performed confirms that the truth value of the formula is *true*, which is enough to guarantee that the property is true for the system.

As it can be seen in this simple example, the advantage of using model checking is that it is possible to express, and consequently to verify, more properties compared to invariant analysis. In our example, the information that a plane never refuels if a passenger is on board is not present in the computed invariants, but can be verified using the model checking. However, the drawback is that it is necessary to explore the whole state space of the system in order to verify a property. Invariants are computed on the static structure of the net, which is usually exponentially smaller compared to the state space of the system. In general, in real huge application both the techniques are useful: invariants give a quick overview of some properties of the system, model checking take more time and it can be used to verify specific properties of the system.

6 Conclusion

In this paper we discussed the verification of high-level Petri nets which use the nets within nets paradigm, with particular attention to the reference nets and the hypernets formalisms. We examined them, and we showed how to transform a subset of reference nets into hypernets, which in turn can be transformed into 1-safe nets. We then proceeded to describe the hypernet plugin created for Renew in the course of our work. With the help of this plugin and external tools we can analyse the transformed low-level nets, and in this way verify properties of the high-level net.

The contributions, and the results of this paper are the implementation of a plugin for Renew with which it is possible to draw of a hypernet, to compute its invariants, and to model check it. With this approach it is now possible to verify properties of systems modelled with net within nets oriented formalisms, such as reference nets and hypernets.

The results of this paper will make it possible to automatically analyse a hypernet, instead of first transforming it by hand, and then analysing the equivalent low-level nets. This will make the verification simpler and more user-friendly, which in turn will make it easier for software engineers to use these techniques in practical use cases. We plan to use these approaches to verify the model of an actually adopted Grid tool for High Energy Physics data analysis, and in the context of the HEROLD project. Future work will also focus on extending the possibilities of the verification, automating the process as far as possible and extending the toolset to other high-level Petri nets formalisms. The flexibility and adaptability of the Renew tool will be a large asset in this endeavour. Finally, the definitions of analysis techniques directly on the high level model, without the need of converting it to a low-level one, is a subject for future investigations, because it will avoid the conversion to low-level nets, which is an expensive operations in term of computational resources.

References

1. M Bednarczyk, L Bernardinello, W Pawłowski, and L Pomello. Modelling and analysing systems of agents by agent-aware transition systems. In F. Fogelman-Soulie, editor, *Mining Massive Data Sets for Security: Advances in Data Mining, Search, Social Networks and Text Mining, and their Applications to Security*, volume 19, pages 103–112. IOS Press, 2008.
2. Marek A. Bednarczyk, Luca Bernardinello, Wiesław Pawłowski, and Lucia Pomello. Modelling mobility with Petri Hypernets. In *Recent Trends in Algebraic Development Techniques*, volume 3423/2005 of *Lecture Notes in Computer Science*, pages 28–44. Springer Berlin / Heidelberg, 2005.
3. Marek A. Bednarczyk, Luca Bernardinello, Wiesław Pawłowski, and Lucia Pomello. From Petri hypernets to 1-safe nets. In *Proceedings of the Fourth International Workshop on Modelling of Objects, Components and Agents, MOCA'06, Bericht 272, FBI-HH-B-272/06, 2006*, pages 23–43, June 2006.
4. Luca Bernardinello, Nicola Bonzanni, Marco Mascheroni, and Lucia Pomello. Modeling symport/antiport p systems with a class of hierarchical Petri nets. In *Membrane Computing*, volume Volume 4860/2007 of *Lecture Notes in Computer Science*, pages 124–137. Springer Berlin / Heidelberg, 2007.
5. Soren Christensen and Niels Damgaard Hansen. Coloured petri nets extended with channels for synchronous communication. *Lecture Notes in Computer Science*, 815/1994:159–178, 1994. Application and Theory of Petri Nets 1994.
6. Michael Köhler and Heiko Rölke. Properties of object Petri nets. In Jordi Cortadella and Wolfgang Reisig, editors, *ICATPN*, volume 3099 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 2004.
7. Olaf Kummer. *Referenznetze.* Logos Verlag, Berlin, 2002.
8. Olaf Kummer, Frank Wienberg, Michael Duvigneau, Jörn Schumacher, Michael Köhler, Daniel Moldt, Heiko Rölke, and Rüdiger Valk. An extensible editor and simulation engine for Petri nets: Renew. In J. Cortadella and W. Reisig, editors,

International Conference on Application and Theory of Petri Nets 2004, volume 3099 of *Lecture Notes in Computer Science*, pages 484 – 493. Springer-Verlag, 2004.
9. Marco Mascheroni. Generalized hypernets and their semantics. In *Proceedings of the Fith International Workshop on Modelling of Objects, Components and Agents, MOCA'09, Bericht 290, 2009*, pages 87–106, September 2009.
10. Einar Smith. Principles of high-level net theory. In *Lectures on Petri Nets I: Basic Models*, volume Volume 1491/1998 of *Lecture Notes in Computer Science*, pages 174–210. Springer Berlin / Heidelberg, 1998.
11. Rüdiger Valk. Object Petri nets: Using the nets-within-nets paradigm. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advanced Course on Petri Nets 2003*, volume 3098 of *Lecture Notes in Computer Science*, pages 819–848. Springer-Verlag, 2003.

Improving a Workflow Management System with an Agent Flavour

Daniel Moldt, José Quenum, Christine Reese, and Thomas Wagner

University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences, Department of Informatics
`http://www.informatik.uni-hamburg.de/TGI/`

Abstract. This paper discusses an application of software agents to improve workflow management systems, with a practical emphasis on Petri net-based systems. The properties of agent technology will be used to gain advantages within the workflow management systems on both a conceptual and practical level. In this paper we discuss the theoretical background of our work, the conceptual idea and approach and one possible practical implementation. As a central practical means we use reference nets, a high-level Petri net formalism. These nets are used to model both agents and workflows, which results in a clean and natural integration of both technologies.

Keywords: High-level Petri nets, workflow management systems, multi-agent systems, software architecture.

1 Introduction

Workflows and *Workflow management systems* (WFMS) have been very attractive research topics in the last decades [13, 18, 16]. They provide means to further understand, unambiguously specify and analyse business processes within organisations. According to the state of the art, several heterogeneous WFMS can be combined to perform a specific complex task that cuts across various organisations. While such a combination is currently possible in an ad hoc manner, a more systematic approach demands greater care both from the modelling and implementation perspectives. In this research, we set out to address this limitation. This paper discusses the conceptual approach to achieve the overall goal.

Among the rising software paradigms, the concept of *agent* is a preeminent one. In the last decade, agents have been touted as a most appropriate paradigm to support the design and implementation of decentralised and distributed applications/systems, which yield intelligent behaviour and require a great deal of interoperability. A decentralised application implies an application which consists of autonomous entities. Clearly, these are among the properties one seeks while developing an approach for interorganizational workflows. Therefore, agents can contribute a great deal in our quest for a systematic approach for interorganizational workflows.

Using agents to design and implement distributed applications across a network is not new in itself. However, the autonomy of the various subparts is not well captured in the overall collaboration. As such, approaches of this kind cannot be generalised for the design and implementation of collaborative applications across various organisations. Concepts such as workflows, which render a clear view of business processes within organisations need to come into play.

Introducing agents in WFMS is not counterintuitive. By virtue of being autonomous, sociable and intelligent, human agents and artificial ones share many similarities. Moreover, human agents' operational mode can be viewed as a set of independent yet interoperable entities. From that angle, a human agent can be viewed as an agent system. Finally, since human agents have to coordinate the various tasks they are involved in at a certain point in time, they can be regarded as a WFMS. We take this human analogy, especially its duality, to show that both, agents and workflows, can be joined in a complex system.

As discussed in the foregoing, WFMS can benefit from agents in various regards. In this paper, we discuss seven points where agents help enhance the level of management in interorganizational workflows. These include distribution, autonomy, interoperability and intelligence. In order to make the resulting approach appealing to new technologies, we structure our assumptions and ideas into a reference architecture, which we believe lays the foundation for more specific and advanced architectures to support collaboration within and between organisations. We do not claim that our current implementation is as powerful as the existing commercial tools. However, thanks to the Petri nets formalism, our implementation holds the potential to properly handle concurrency, robustness and resilience in the future.

The contributions discussed in this paper are a clearer articulation of ideas and intuitions presented in [20–22]. More than all these papers, we elaborate on the underpinnings of the conceptual role agents can play in the new approach. Finally, we discuss the implementations.

The remainder of the paper comes as follows. Section 2 describes the technical and theoretical background of our work. Section 3 gives an overview of our overall architecture. Section 4 examines the conceptual view of our approach, while Section 5 discusses one implementation of our approach. Finally, Section 6 draws conclusions and directions for the future.

2 Frameworks & Formalisms

In this research, MASs are designed following MULAN's (**MUL**ti-**A**gent **N**ets) structure [11, 23]. MULAN has been extended with CAPA (**C**oncurrent **A**gent **P**latform **A**rchitecture) in order to comply with FIPA's (**F**oundation for **I**ntelligent and **P**hysical **A**gent (see http://fipa.org)) (communication) standards to support concurrent execution [4]. MULAN and CAPA describe the various components of a MAS using reference nets, which can be executed using the RE-

NEW (**RE**ference **NE**ts **W**orkshop (see http://www.renew.de)) tool. The reader should thus note that not only do we offer a formal ground to reason about the behaviours of agents, but we also provide an execution environment for MAS.

Inspired from the *nets within nets* concept introduced by Valk [25], MULAN's structure is a four-layer architecture used to describe a MAS. These layers respectively describe the overall MAS, the agent platforms, the agents and their behaviour, the protocols. Every layer within MULAN's reference architecture is modelled using reference nets, a high level Petri net formalism which will be discussed later.

In order to adhere to FIPA's standards, and especially the communication mechanisms, MULAN has been extended with CAPA. In so doing, CAPA agents can easily interact with any type of FIPA compliant agents.

Each of the key concepts in this paper, agent and MAS on the one hand and workflows on the other hand, is represented using a Petri net formalism. Agents and MAS are represented using reference nets, while workflows and WFMSs are represented using workflow nets implemented as a special kind of reference nets. Reference nets have been introduced in [15]. They follow the philosophy of nets within nets. Reference nets use *Java* as an inscription language, manipulate various types of data structures, and like many other types of high-level Petri net formalisms, offer several types of arcs. Finally they use synchronous channels to synchronise with other nets or Java objects. The treatment of Java objects and net instances is transparent, so that both kinds of artefacts can be exchanged arbitrarily. The workflow nets used in our systems are based on principles of workflow nets introduced in [26]. They are implemented as reference nets and make use of a special task transition introduced in [9]. Moreover, in the Petri net community, tool support is a general trend. Therefore, the RENEW tool has been developed to support quick prototyping of systems or parts of systems using the reference net formalism. RENEW provides an editor to specify and draw the nets as well as a simulation engine to test and validate them.

3 Overall architecture

The results we present in this paper are part of a larger ongoing effort. The systems described in the next sections can be classified within the overall architecture described in [19]. The goal of this architecture is to integrate agent and workflow technologies. The architecture consists of five tiers, built on top of each other. Each tier itself is a layered architecture, which combines various aspects of both technologies. Starting from either a pure workflow or agent system, the architecture gradually evolves into a novel integrated unit system, which equally benefits from both original concepts. The main motivation in building this architecture lies in the shortcomings of each individual concept as is discussed in detail in [19] and [28]. In short, agent technology struggles with offering a clear behavioural view of large distributed systems, but can describe the structure of such a system in a natural way. Workflow technology can easily describe the behavioural view of a complex system, but struggles with the structural view.

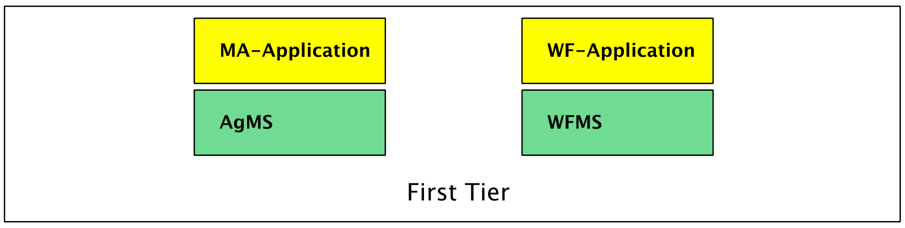


Fig. 1. Architecture of the first tier of the overall architecture, modified from [19]

In combining both technologies we can integrate the views offered by both into one new system which supports both a clean structural and behavioural view. It should be explained that the notion of tiers in this architecture denotes a kind of step-by-step refinement/enhancement on the way to the overall goal of integration. The tiers can be viewed as the layers of the overall *abstract* architecture but they do not correspond to layers within a *concrete* architecture. Each tier modifies the structure of its own layered architecture compared to the previous tier and in doing so enables new or improved aspects to be used. We will now shortly discuss the five tiers of the architecture.

First Tier The first tier is our starting solution to address the limitations of both technologies. It involves either a pure agent management system (AgMS) or a WFMS. Such systems exclusively use workflow or agent technology to provide their functionality. This means that there is almost no integration between the two. Because of this, the architectural view of this tier (see Figure 1) offers only two layers. The bottom depicts the adopted management system (either agent or workflow), on top of which an application lies. Examples of systems, which can be classified into this tier, are Mulan and Capa on the agent side and WFMS like ADEPT (see [3]) or WIFAi (see [24]) on the workflow side.

Second Tier In the second tier, one of the paradigms is used to realise the other. In other words, we use agents to design a WFMS, and vice versa. Because of this, there are two variants of the second tier (agents in the background or workflows in the background). The architectural view of this tier (see Figure 2) offers three layers. The topmost layer still represents an application but between the bottom management system and the application an intermediate layer has been added. This layer implements a management system for the alternative concept using the functionality of the bottom layer. For example, if the bottom layer is an AgMS, then the intermediate layer is a WFMS based on agents. The application layer of this tier uses *only* the concept provided by the intermediate layer.

Building on the first tier the second tier can be achieved by designing the application within the first tier to be the required management system. On top of that management system another application can then be built, turning the application layer of the first tier into the intermediate layer of the second tier.

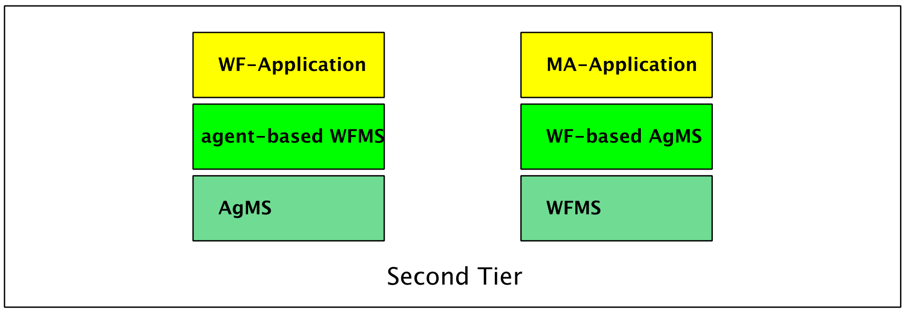


Fig. 2. Architecture of the second tier of the overall architecture, modified from [19]

Compared to the first tier, the advantage is that one can perceive an integration of the concepts. However, the available constructs only affect the background instead of being directly available in the application layer. For example, distribution, interoperability, etc. are facilitated in WFMS using agents.

In the remainder of this paper, the higher level tiers will only be based on the variation using agents in the background, i.e. the one including an AgMS, an agent-based WMFS (AgWFMS) and an application. Examples for these kind of systems are detailed in [6], [7] and [10].

Third Tier The third tier greatly enhances the application development by employing both agents and workflows. This results in an arbitrary degree of integration between agents and workflows. In addition to the interface between the application and intermediate layer, this tier allows direct access from the application layer to the bottom management system. In practice, the application can thus use both the interfaces offered by the core AgMS and the AgWFMS. Consequently, the key functionality of the AgMS is then combined with that of the AgWFMS. The architectural view of this tier (see Figure 3) only adds a direct connection between the application and the bottom layer, compared to the second tier. While this additional connection is a clear advantage over the previous tier by expanding potential and flexibility, it suffers from a major limitation. The resulting system is completely unstructured, i.e., the relation and integration between agent and workflow needs to be potentially re-invented for each application. Consequently it becomes very difficult to harness the power of this tier, especially the efficient design of complex systems. In order to reach a structured integration of both concepts within the architecture, we need to take one step back and limit the immense possibilities offered by this tier.

Fourth Tier The fourth tier adds an integration layer to the architecture, which is responsible for restricting the possibilities of the third tier in order to provide an explicit structure for the application developed on it (see Figure 4). Through this integration both technologies are used in the background and the relationship between agents and workflows is pre-defined. However, only *one* perspective is

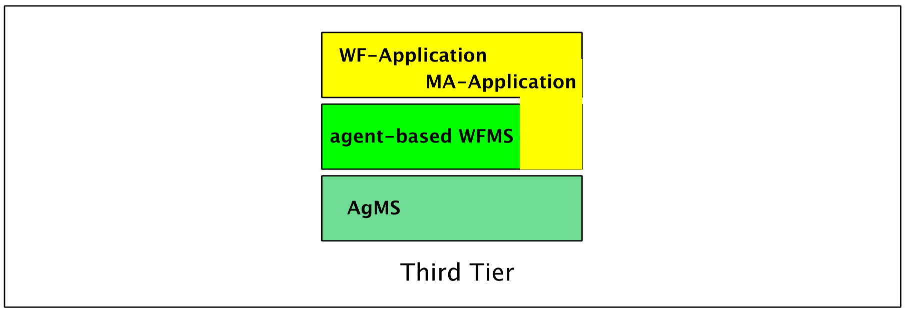


Fig. 3. Architecture of the third tier of the overall architecture, modified from [19]

supported when modelling on the application layer. In this way it provides an abstraction and thus a higher level of modelling.

In order to achieve the desired explicit structure, this tier basically reduces the functionality offered to the application layer compared to the third tier. It refocuses on either agents or workflows as the exclusive main abstraction for application development. There are once again two distinct variations of this tier, one offering agents to the application (called *workflowagents*; left hand side of Figure 4), the other one offering workflows (called *agentworkflows*; right hand side of Figure 4). The integration layer provides exclusively WFMS or AgMS functionality, but uses both technologies in the background. This means that an agent application possesses parts and aspects of workflows and vice versa (in the other variation). In this way the possibilities of this tier are, opposed to the third tier, restricted, because we refocus on just one technology. But by doing this, we gain a much more powerful means of supporting one of the two technologies. The integration of both variations will take place in the fifth tier. For now we need both variations in order to create the desired structure within the relation between agents and workflows in both directions separately.

It is worth noting that, even though we are looking at either workflows or agents at the top layer again, the main difference between this tier and the second tier is that we no longer have one concept realising the other. Rather, we obtain a successful combination of both, i.e., agents and workflows working side by side and benefiting from each other. The agentworkflow variation of this tier is the main focus of this paper and will be discussed in detail in the main sections.

Fifth Tier This tier introduces the concept of *unit*, an abstraction to any entity involved in the design of the system. Units offer both the facets of agents and workflows. In order to achieve this, the AgMS and WFMS have to be integrated and combined. This results in a novel type of management system, a *unit management system* (UMS). The architecture of this tier can be seen in Figure 5. The integration layer of the fourth tier has been split into two parts, of which UMS represents the upper part. Since one can no longer clearly differentiate between

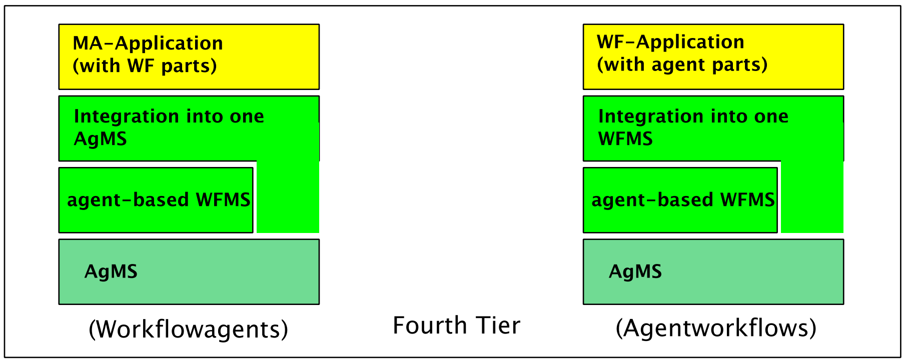

MA-Application (with WF parts)
Integration into one AgMS
agent-based WFMS
AgMS
(Workflowagents)
Fourth Tier
WF-Application (with agent parts)
Integration into one WFMS
agent-based WFMS
AgMS
(Agentworkflows)

Fig. 4. Architecture of the fourth tier of the overall architecture, modified from [19]

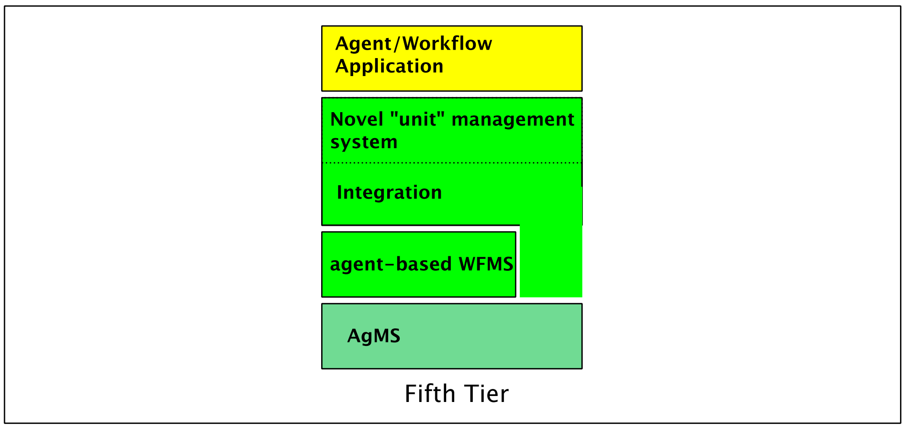


Fig. 5. Architecture of the fifth tier of the overall architecture, modified from [19]

agents and workflows, the application layer is simply called unit application, also referred to as agent/workflow applications in [19]. In merging both agent and workflow concepts into a single unit concept, both the structural view (from the MAS) and the behavioural one (from the workflows) are available. Note that both these views are available during runtime and design time.

4 Conceptual View

In this section we will discuss our conceptual approach to improving workflow management with agent technology. As stated before our goal is to use aspects of software agents to benefit the execution and management of workflow instances. Because we use agents to improve workflows we generally refer to this approach as *agentworkflows*. We reason that common properties of agents, like mobility, autonomy and proactivity, and especially the encapsulation of workflow instances through agents can greatly benefit workflow execution. Having agents in general handle aspects of workflow management can also help distribute the execution in order to spare less powerful resources.

As mentioned before this particular work is just part of a larger effort to integrate workflow and agent technologies in order to achieve a novel approach that combines the advantages of both technologies. The work presented in this paper focuses on the fourth tier of the overall architecture (see Section 3). We discuss the variation of the fourth tier, in which the integration layer offers a WFMS to the application layer. This WFMS strongly relies on the functionality of the AgMS in the background in order to provide its own functionality. This means that the workflows offered to the application possess some properties gained from the agents. How this can be achieved will be discussed in this section, as well as the advantages and disadvantages of this approach,.

One of the core ideas behind our approach is to encapsulate workflow instances through autonomous software agents. With this it is possible to transfer properties of that software agent directly to the workflow instance. In other words the clear separation between agent and workflow begins to diminish. This is a key concept of the overall architecture and is important for the fifth tier. Another important aspect of our approach is to also consider other entities of the workflow management system as agents. Having the general functionality of the WFMS be provided by agents is already part of an AgWFMS of the second tier. In our approach agents can be used to realise any of the elements (e.g., tasks, users, resources, etc.) of the workflow as well. In [17] for example, we used agents to realise activities. Thanks to the agent technology, the resulting WFMS does not only focus on the behaviour of the system, as was the case in the previous tiers of the overall architecture, it also emphasises the structure of the system. This paper focusses on the aspect of encapsulating workflow instances through agents. Other aspects are considered but will not be discussed in great detail.

We will now examine some of the principal and conceptual areas and aspects in which workflows can benefit from agents. The following points will directly cover some agent properties and their particular benefits but will also include some general observations.

Encapsulation In general the encapsulation of one or more workflow instances through agents can be seen as a prerequisite to opening up many of the possibilities offered by the agent-oriented paradigm. Without this concept it would be hard or impossible to transfer other agent properties over to the workflows. Nonetheless the encapsulation also benefits the workflows in more ways than that. For example the encapsulation provides the workflows with an even clearer identity within the overall system since they can now be identified in the same way as the other elements (agents) of the system. This makes it easier to monitor, observe and analyse the system, which in turn makes maintenance and improvement more efficient. A disadvantage of the encapsulation is that the number of agents active is possibly, drastically increased, depending on how the encapsulation is handled. This may pose problems on less powerful systems, which simply cannot handle this number of agents or the communication between them. However, since agent architectures are generally built to efficiently handle communication, this should not pose a real problem in practical use.

Mobility By allowing workflows to gain agent mobility they benefit in a variety of possibilities. In the context of software agents mobility describes the capability of a software agent to discontinue its execution within one execution environment (agent platform), migrate to another environment and continue the execution there starting off from its previous state. For agent-workflows this means that the execution of a workflow can be discontinued on one instance of an executing WFMS and continued on another WFMS. Practically this can be used if certain resources needed for the execution of a workflow are not available on every platform. This can include particular (groups of) users or certain, possibly critical data. Another use case for this property is to have a workflow instance migrate not because certain resources are needed, but because its home platform is beginning shutdown or because another platform carries less of a load then the home platform. This use of mobility can lead to improved flexibility, efficiency and fault tolerance.

Autonomy One of the key concepts of the agent paradigm is that agents are autonomous entities. This means that, to a certain degree, they are independent of their environment and can choose for themselves whether to execute an action or not. In the context of agentworkflows this property can be used in a number of ways. It can for example be used as a kind of access control to critical data for which an agent is responsible. This can be a workflow instance but also other entities like activities or the handling of users. Another use of this becomes relevant if combined with mobility. An agent migrating to another platform to access certain data or perform certain actions can do this relatively independent from the other agents and software constructs of that platform, if, of course, it has all the necessary permissions.

Intelligence Intelligence in software agents can be used to describe a multitude of aspects. One major aspect is the ability of certain agents to proactively decide by themselves which actions to take. In the context of workflow management this can be used to predetermine which users should be offered certain tasks, taking variables like workload into consideration. At this point reactivity of agents also comes into play. Software agents can react to events in their environment and adapt according to the situation. For example if there is an error during the execution of a task the agent could observe this and retry the action with changed parameters. Another very interesting aspect where intelligence, proactivity, reactivity and adaptiveness can be used is the adaptivity of workflow instances. Changing workflow instances and even entire workflow definitions according to changed circumstances (current or permanent) improves the flexibility and versatility of a WFMS and can be handled in a natural way using agent intelligence.

Distribution The agent oriented paradigm naturally supports the design of distributed software systems. The main reasons for this are the asynchronous message communication and the autonomy of the individual agents. By relying on agents as the main building blocks of a WFMS it is easy to use these predispositions for the distribution of the system. The communication of different parts of the system can be handled through asynchronous messages, which are flexible and versatile. Extending on this idea opens up even more possibilities of using distribution to the advantage of workflows. Interorganizational workflows can benefit from a distributed WFMS, so that their critical information is not stored in some centralised location.

Interoperability The Fipa communication standards are accepted by many widely-used agent frameworks. Adhering to these standards guarantees interoperability between the different involved software systems, independent of agent architecture or framework. This can be translated into workflow management based on agents as well. In this case different WFMS of different providers can work together, as long as they can process the data structures that are exchanged. This aspect is especially important in the context of interorganizational workflows since it allows some freedom for the choice of the different WFMS in the different companies. But also in general use cases interoperability can be used to an advantage. A Fipa-compliant WFMS can request data from any other Fipa-compliant system, which improves the possibilities of the WFMS. Another aspect which is related to this and distribution, is the openness of the system. Through interoperability and distribution it is possible to create flexible and dynamic open systems to which different WFMS can connect to, complete some tasks, and then disconnect again. Open systems can provide users with functionality that is otherwise difficult to obtain without specialised software solutions.

Structure This point is related to the motivation behind our overall architecture. As described, we reason that workflow systems have trouble adequately describing the structure of the system they are modelling, while focussing on the behaviour. Agent systems on the other hand possess a strong focus on this structure. By joining the two in the ways described in this paper we begin to combine this structural view given by the agents with the behavioural view of the workflows. This is mostly related to the encapsulation aspect discussed above, but contains a more abstract view. By relying on agents one can easily describe the current state of an entity including its current location (in regards to distribution), knowledge and behaviour. By adapting this for workflow instances it can already help provide the structural view needed within a distributed system. If the agentworkflow idea is taken even further and every aspect of the system modelled through agents the structural view becomes even more useful. The location (in regards to distribution) of every resource, user and workflow can be determined and displayed in a way that helps monitoring and maintaining the system.

It should be noted that all these properties only unfold their full potential if used in combination. Every one of these properties and aspects possesses some benefits but together with the others new and improved possibilities can be achieved. For example using mobile agents in a distributed environment of many interoperable agent platforms is more advantageous then forcing the same agent system onto all involved partners. Equally an autonomous, intelligent agent can decide for itself if a migration is reasonable or not and initiate the action accordingly. Using these properties together also strongly improves interorganizational workflow management and execution. While interoperability and distribution already favour this field, the other properties are also useful, especially in collaboration. For example mobility allows for the transmission of data in a natural way, while encapsulation allows for the clear separation of critical data.

The main disadvantage of our approach is that the realisation and handling of these improved workflows are more complex than handling regular workflows. The reason for this is mainly that the new and improved possibilities will be difficult to harness. It can, if used in the wrong way, affect execution in a negative way or even, in the worst case, prohibit correct execution at all. However, if used correctly and efficiently, they offer clear, distinct advantages to workflow execution in general. They offer novel ways of modelling many parts of workflows and can increase efficiency in use.

After discussing the conceptual view in this section we have shown that our approach offers many advantages, but is difficult to realise and handle. For the realisation part we have chosen technologies based on Petri nets. One problem of the conceptual approach is that different kinds of entities (agents and workflows) have to be combined. By choosing Petri nets as a common basis we can partially circumvent this problem, since it is easier to combine the two kinds of entities when they possess the same basis at the lowest level. On the other hand this choice has the problem of not being widely spread and available. However, certain aspects, like concurrency and displaying behaviour, are very easy to model using Petri nets. In the next section we will discuss one prototypical implementation of our conceptual approach using Petri nets, which already covers some of the properties described in this section.

5 Implementation

In this section we will discuss a prototypical implementation of our agentworkflow approach. As mentioned before we use Petri net based technologies to achieve a common basis for the integration and combination of workflow and agent technologies. In particular we use MULAN and CAPA for our agents and workflow nets for our workflow functionality (see Section 2). The starting point of the practical work is an AgWFMS of the second tier of the overall architecture. This AgWFMS has been described in detail in [27]. It relies solely on agents to provide the functionality but does not mix the agent and workflow concepts enough to be considered an agentworkflow system.

Before going into the details of the implementation we will shortly discuss how the different properties observed in the conceptual section can be mapped onto Petri nets. Since discussing these aspects in a reasonable extent would go beyond the scope of this paper and since extensive work on this has already been performed and published we will limit this to referring to other contributions. The mobility aspect has been extensively studied, especially in the context of

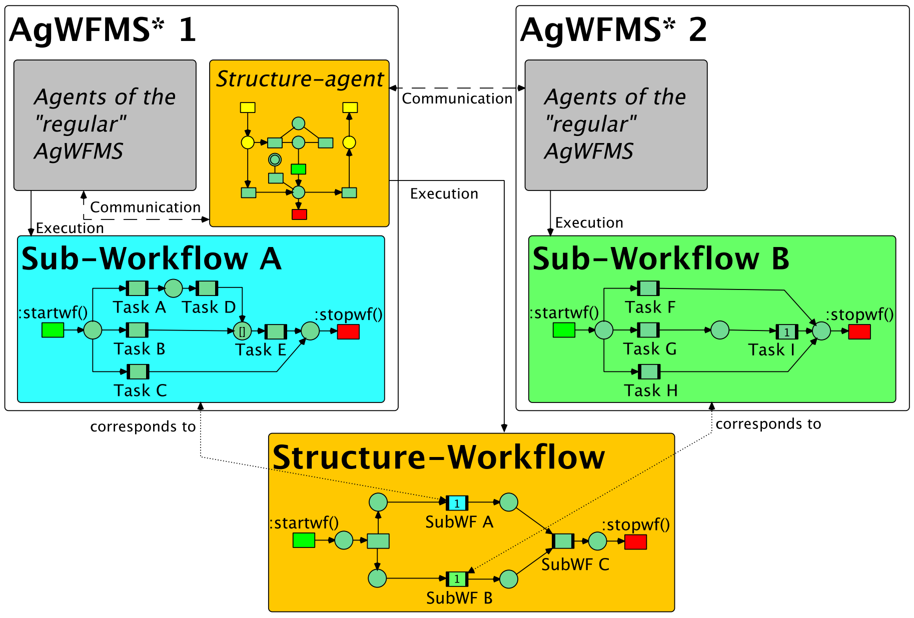


Fig. 6. Principle approach of the S-AgWf

nets within nets, for example in [2] and [12]. The autonomy and intelligence of Petri net agents have been discussed in the context of MULAN in [23]. The encapsulation aspect has been examined for object-oriented nets in [1]. Interoperability and openness have been explored in [14]. The final aspect, the structural view of combining agents and workflows, was discussed in [19].

The practical approach, called structure-agentworkflow (S-AgWf), extends the regular AgWFMS (now called AgWFMS*) to allow for the definition and execution of distributed workflow instances. More precisely, the workflow instances are now hierarchical workflows with nested subprocesses as defined by the WfMC (see [8]).As a consequence the entire system consists of a number of CAPA platforms, which all execute instances of the extended AgWFMS* and are working together. The different AgWFMS* instances are known to one another and messages can be exchanged between them. A more detailed description of the S-AgWf approach can be found in [28].

The basic principle of the S-AgWf can be seen in Figure 6. One agent encapsulates one workflow instance. This agent, called structure-agent, possesses an internal workflow, the structure-workflow. When the structure-agent for a new workflow instance is started, it receives the definition of the structure-workflow from the database agents of the AgWFMS* and instantiates the workflow net. When this initialisation is finished, the execution of the structure-workflow automatically begins. The tasks of the structure-workflow correspond to sub-workflows. Sub-workflows can only be executed on certain AgWFMS* instances within the overall system. The information about which AgWFMS* instance is suitable is stored within the data of the task and can be extracted by the structure-agent. Whenever a task becomes active, the structure-agent assigns itself as the executor of that task. Once this is done the structure-agent queries a special agent of his own platform for a list of all the known AgWFMS* instances currently active. It then compares this list to the information extracted from the task and chooses a suitable AgWFMS* instance. The interface-agent of the chosen instance is then contacted by the structure-agent. The structure-agent asks the interface-agent to instantiate the subworkflow locally and transmits all relevant parameters, like input data etc. The subworkflow is then executed like any other workflow in the regular AgWFMS. Once it has reached the end of its execution the responsible structure-agent is informed and any (optional) results are sent back. The results are transmitted into the structure-workflow net and the structure-agent completes the task, so that the execution can continue. The execution of tasks and subsequent instantiation and execution of sub-workflows is continued, until the end of the structure-workflow is reached. The initiator of the overall workflow is then informed and the structure-agent can terminate.

In the example in Figure 6 two sub-workflows are currently executed on the two different AgWFMS* platforms. The structure-agent responsible for the structure-workflow is communicating with the agents of the two AgWFMS* platforms in order to initiate the execution and receive results. When both sub-workflows are finished the structure-agent will start a final sub-workflow (*SubWF C*), before it can conclude the execution of the structure-workflow.

This realisation of the agentworkflow concept offers distinct and practical advantages, but also still suffers from some limitations. The possibility to distribute the execution of workflows is a huge advantage for the otherwise centralised AgWFMS. The support of nested subprocesses allows for interorganizational workflows to be defined and executed. Since the details of the local workflows are not needed globally, the sub-workflows and any critical data they may contain are only known to the local parties. This satisfies the need of interorganizational workflows to secure and conceal confidential and valuable information.

The main limitation of this particular, specialised implementation of the agentworkflow concept is its still centralised nature. If the platform of the structure-agent is disconnected or fails, the entire workflow fails. This could partially be rectified by adding mobility to the structure-agent. It can then easily migrate to another platform, if it discovers any changes in its home platform that might hinder its execution.

The pre-defined relationship between agents and workflows within this system combines the structural aspect of agents with the behavioural aspect of workflows as is the goal in the fourth tier of the overall architecture. The two concepts agent and workflow begin to merge together, since in this system a workflow is an agent and partly vice versa. The practical advantages this particular system gains from this merge mostly consist in a groundwork for further enhancements. Agent autonomy may for example be used to give the structure-agent more control over the workflow instance (e.g. choice over where sub-workflows are executed), which could result in added flexibility. However the instances within the S-AgWF system already possess certain degrees of distribution (structure-agents communicate with other AgWFMS* platforms in order to execute their structure-workflow), interoperability (the structure-agents exchange FIPA-compliant messages so it is possible to exchange the AgWFMS* platforms with other WFMS if they adhere to the interface) and encapsulation (the structure-workflow is clearly encapsulated by the structure-agent).

6 Conclusion

In this paper, we made a strong case for a systematic introduction of the agent concept to enhance the management of workflows. We pointed out key aspects in which agents improve workflows and their management.

In our quest to develop a systematic approach to support workflow management, we proposed a reference architecture, which builds on an integration of agents and WFMS. The architecture consists of five distinct tiers. These different tiers gradually show how the agent concept first integrates into WFMS and then enhances them on the various aspects we discussed above. This effort culminates in the fifth tier, where both concepts exist alongside each other. As stated in the foregoing, our overall goal in this research is to achieve a seamless integration of agents and WFMS. In this paper, we presented an approach which builds on the agent technology to address WFMS. However, the other variant of the fourth tier, the workflowagents, needs to be considered as well. In it, the main abstraction of the application layer are agents, which strongly rely on the functionality of the WFMS in the background to address the inherent limitations to an agent-based system. Clearly, following that perspective, a WMFS could for example bring its systematic and proof-driven approach to complex task execution. In the future, we wish to explore that perspective as well.

From the lessons learned from both approaches, we expect to collect the amount of information that enables us to design a full-fledged conceptual approach which offers the best of both agent and workflow technologies in one single system, i.e. the fifth tier. Such an approach will balance out the weaknesses of each technology. With the support of high-level Petri nets as a foundational formalism, we are guaranteed of combining structure and behaviour in one representation.

References

1. Ulrich Becker and Daniel Moldt. Objekt-orientierte Konzepte für gefärbte Petrinetze. In Gert Scheschonk and Wolfgang Reisig, editors, *Petri-Netze im Einsatz für Entwurf und Entwicklung von Informationssystemen*, Informatik Aktuell, pages 140–151, Berlin Heidelberg New York, 1993. Gesellschaft für Informatik, Springer-Verlag.
2. Lawrence Cabac, Daniel Moldt, Matthias Wester-Ebbinghaus, and Eva Müller. Visual Representation of Mobile Agents – Modeling Mobility within the Prototype MAPA. In Duvigneau and Moldt [5], pages 7–28.

3. Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, and Martin Jurisch. Von ADEPT zur AristaFlow BPM Suite - Eine Vision wird Realität: "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen. *EMISA Forum*, 29(1):9–28, 2009.
4. Michael Duvigneau. Bereitstellung einer Agentenplattform für petrinetzbasierte Agenten. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg, December 2002.
5. Michael Duvigneau and Daniel Moldt, editors. *Proceedings of the Fifth International Workshop on Modeling of Objects, Components and Agents, MOCA'09, Hamburg*, number FBI-HH-B-290/09 in Bericht. University of Hamburg, September 2009.
6. Lars Ehrler, Martin Fleurke, Maryam Purvis, and Bastin Tony Roy Savarimuthu. Agent-based workflow management systems (WfMSs) - JBees: a distributed and adaptive WfMS with monitoring and controlling capabilities. *Information Systems and E-Business Management*, 4, Number 1 / January, 2006:5–23, 2005.
7. Andrea Freßmann, Rainer Maximini, and Thomas Sauer. Towards Collaborative Agent-Based Knowledge Support for Time-Critical and Business-Critical Processes. In *Professional Knowledge Management*, volume 3782, pages 420–430, Berlin Heidelberg New York, 2005. Springer-Verlag.
8. David Hollingsworth. *The Workflow Reference Model*. Workflow Management Coalition. Verfügbar auf http://www.wfmc.org/reference-model.html.
9. Thomas Jacob. Implementierung einer sicheren und rollenbasierten Workflowmanagement-Komponente für ein Petrinetzwerkzeug. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg, 2002.
10. N. R. Jennings, T.J. Norman, and P. Faratin. ADEPT: An Agent-Based Approach to Business Process Management. *ACM SIGMOD Record*, 27:32–39, 1998.
11. Michael Köhler, Daniel Moldt, and Heiko Rölke. Modelling the Structure and Behaviour of Petri Net Agents. In J.M. Colom and M. Koutny, editors, *Proceedings of the 22nd Conference on Application and Theory of Petri Nets 2001*, volume 2075 of *Lecture Notes in Computer Science*, pages 224–241. Springer-Verlag, 2001.
12. Michael Köhler, Daniel Moldt, and Heiko Rölke. Modelling mobility and mobile agents using nets within nets. In Wil van der Aalst and Eike Best, editors, *Proceedings of the 24th International Conference on Application and Theory of Petri Nets 2003 (ICATPN 2003)*, volume 2679 of *Lecture Notes in Computer Science*, pages 121–139. Springer-Verlag, 2003.
13. Michael Köhler-Bußmeier. Hornets: Nets within Nets combined with Net Algebra. In Karsten Wolf and Giuliana Franceschinis, editors, *International Conference on Application and Theory of Petri Nets (ICATPN'2009)*, volume 5606 of *Lecture Notes in Computer Science*, pages 243–262. Springer-Verlag, 2009.
14. Michael Köhler-Bußmeier. SONAR: Eine sozialtheoretisch fundierte Multiagentensystemarchitektur. In Rolf v. Lüde, Daniel Moldt, and Rüdiger Valk, editors, *Selbstorganisation und Governance in künstlichen und sozialen Systemen*, volume 5 of *Reihe: Wirtschaft – Arbeit – Technik*, chapter 8–12. Lit-Verlag, Münster - Hamburg - London, 2009.
15. Olaf Kummer. *Referenznetze*. Logos Verlag, Berlin, 2002.
16. Kolja Markwardt, Daniel Moldt, and Christine Reese. Support of Distributed Software Development by an Agent-based Process Infrastructure. In *MSVVEIS 2008*, 2008.

17. Kolja Markwardt, Daniel Moldt, and Thomas Wagner. Net Agents for Activity Handling in a WFMS. In Thomas Freytag and Andreas Eckleder, editors, *16th German Workshop on Algorithms and Tools for Petri Nets, AWPN 2009, Karlsruhe, Germany, Proceedings*, CEUR Workshop Proceedings, 2009.
18. Daniel Moldt. *Höhere Petrinetze als Grundlage für Systemspezifikationen.* Dissertation, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg, August 1996.
19. Christine Reese. *Prozess-Infrastruktur für Agentenanwendungen.* Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg, 2009. Pdf: http://www.sub.uni-hamburg.de/opus/volltexte/2010/4497/.
20. Christine Reese, Jan Ortmann, Daniel Moldt, Sven Offermann, Kolja Lehmann, and Timo Carl. Architecture for Distributed Agent-Based Workflows. In Brian Henderson-Sellers and Michael Winikoff, editors, *Proceedings of the Seventh International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2005), Utrecht, Niederlande, as part of AAMAS 2005 (Autonomous Agents and Multi Agent Systems), July 2005*, pages 42–49, 2005.
21. Christine Reese, Matthias Wester-Ebbinghaus, Till Dörges, Lawrence Cabac, and Daniel Moldt. A Process Infrastructure for Agent Systems. In Mehdi Dastani, Amal El Fallah, Joao Leite, and Paolo Torroni, editors, *MALLOW'007 Proceedings. Workshop LADS'007 Languages, Methodologies and Development Tools for Multi-Agent Systems (LADS)*, pages 97–111, 2007.
22. Christine Reese, Matthias Wester-Ebbinghaus, Till Dörges, Lawrence Cabac, and Daniel Moldt. Introducing a Process Infrastructure for Agent Systems. In Mehdi Dastani, Amal El Fallah, João Leite, and Paolo Torroni, editors, *LADS'007 Languages, Methodologies and Development Tools for Multi-Agent Systems*, volume 5118 of *Lecture Notes in Artificial Intelligence*, pages 225–242, 2008. Revised Selected and Invited Papers.
23. Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2004.
24. Michael Tarullo, Daniela Rosca, Jiacun Wang, and William Tepfenhart. WIFAi - A Tool Suite for the Modeling and Enactment of Inter-organizational Workflows. In *SOLI '09. IEEE/INFORMS International Conference on Service Operations, Logistics and Informatics 2009*, pages 764–769. IEEE, 2009.
25. Rüdiger Valk. Concurrency in Communicating Object Petri Nets. In *Advances in Petri Nets: Concurrent Object-Oriented Programming and Petri Nets*, volume 2001 of *Lecture Notes in Computer Science*, pages 164–195. Springer-Verlag, Berlin Heidelberg New York, 2001.
26. Wil M.P. van der Aalst. Verification of Workflow Nets. In *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, volume 1248, pages 407–426, Berlin Heidelberg New York, 1997. Springer-Verlag.
27. Thomas Wagner. A Centralized Petri Net- and Agent-based Workflow Management System. In Duvigneau and Moldt [5], pages 29–44.
28. Thomas Wagner. Prototypische Realisierung einer Integration von Agenten und Workflows. Diploma thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg, 2009.

Automata and Petri Net Models for Visualizing and Analyzing Complex Questionnaires – A Case Study –

Heiko Rölke

Deutsches Institut für Internationale Pädagogische Forschung
(German Institute for International Educational Research)
Solmsstraße 75, 60486 Frankfurt, Germany
roelke@dipf.de

Abstract. Questionnaires for complex studies can grow to considerable sizes. Several hundred questions are not uncommon. In addition, routings are used to distinguish between question paths for different respondents. This leads to the question of how to ensure validity and other important properties.
We examine this question for a case with even more demanding side conditions: An important part of the OECD study ”PIAAC” (Programme for the International Assessment of Adult Competencies) is a background questionnaire (BQ) containing more than 400 questions. This BQ has to be adapted by all participating countries. Nevertheless, integrity of the overall system has to be ensured.

Keywords: automata, Petri nets, questionnaires, analysis

1 Motivation and Overview

Large scale studies in psychology, sociology, and for many other purposes try to find out characteristics of complete populations or at least of big parts of a population. International studies often aim at comparing the complete population of one country to that of another country. The well-known PISA study of the OECD, as an example, aims at comparing *all* students of age 15 worldwide. This is done by examining representative samples in each country that participates in PISA.

To be able to compare one first has to find out some background information about the people that are compared. This is mainly done by asking those people questions. Larger chunks of questions grouped together in order to find about the *background* of the surveyed people are called *background questionnaires*, or BQ in short.

The author of this paper was involved in the definition, implementation, national adaptation, and deployment of the BQ for the OECD PIAAC study.[1] The OECD is the "Organization for Economic Co-Operation and Development", see [5] for details. Among many other activities, the OECD is well-known for organizing world-wide comparability studies, like the PISA study. PISA [6] is the abbreviation of "Programme for International Student Assessment". The PIAAC study, "Programme for the International Assessment of Adult Competencies" can be seen as an extension of the PISA study for adults. It aims at finding out about skills needed by adults in order to be successful in everyday work life. See [7] for details about the PIAAC study. PIAAC is carried out by 24 countries all over the world (participating countries are located in North and South America, Europe, Asia and Oceania).

1.1 Background Questionnaire Properties

There is no exact definition of what a BQ is. It is therefore not possible to exactly determine properties that have to be valid for each and every BQ. Naively, it is just a bunch of questions that an interviewer has to present to an interviewee. In practice, in discussions with psychologists, sociologists, or other questionnaire practitioners, certain universally agreed principles and best practices become clear. From this starting point, desirable properties can be derived. Nevertheless, it is not possible in the moment to definitely define and answer all related questions. We strive for more general validity, though.

A BQ usually has one single entry or starting point, the first *item* or *question*. In practice, this is often a hidden item, where predefined data is imported. An example: One often knows the name of the interviewee in advance. Within the BQ, there may be many different paths through the question pool, often depending on previously entered data or chosen randomly. An item that is intended to be the last question of a BQ is called an *end item*. Again, this may be a visible item (a question) or a hidden item not visible to the interviewer. While there often only is one end item, for example thanking the interviewee for time and patience or, more technically, exporting the assembled data, this is not a standard requirement of a BQ.

Due to practical considerations, there often is the possibility to pause an interview or to break it off. While pausing has no implications for the structure, a break-off means that any item can be an end item or has a connection to an end item.

The normal flow through a BQ should not result in a dead end. A dead end is an item that was not considered to be an end item. Other requirements are more on the semantic side. Each possible question sequence has to make sense semantically. On the other hand, each desired or planned sequence has to be possible, e.g. by entering appropriate answers.

[1] The work was done in the international consortium responsible for implementing, deploying, and analyzing the study, led by ETS [3] in Princeton, USA. Most of the implementation work on the BQ was done by the CRP Henry Tudor [2] in Luxembourg.

1.2 Overview

The rest of the paper is structured as follows: In Section 2 we give an overview on the BQ of the PIAAC study. We also give examples of the format in that the BQ is defined. Following up on that we develop first simple models for the PIAAC BQ in Section 3. The models are put into practice in Section 4. They are used to gain quite some insight and find errors, but are not sufficiently powerful to represent all important aspects of our application. So we carry on in Section 5 with more powerful Petri net models that allow for more sophisticated analysis. We conclude in Section 6 with an outlook on further work and possible generalizations.

2 The PIAAC Background Questionnaire

The PIAAC study mainly consists of two important parts: a background questionnaire (BQ) and cognitive tests (cognitive items, CI). Both parts are embedded into an overall workflow that controls all parts of the survey. This workflow is implemented in the same way as the BQ.

The PIAAC BQ starts with general questions about the interviewee to find out whether he is suited to take part in the survey or not. Afterwards questions in different categories are asked, grouped together in blocks. Examples for such blocks are questions about the educational background, skills needed in everyday work, and questions about private life related to work skills. In order to shorten the overall interview time, parts of the blocks are arranged in a rotated design so that not all interviewees are asked the same questions. Another example of inter-block routing is that certain blocks are not administered if the requirements for asking these questions are not fulfilled, e.g. questions about current work in case of an unemployed interviewee.

In addition to the inter-block routing, complex routing is used within blocks to administer the right questions. A good example for such a routing are questions about the education of an interviewee: If an interviewee has never been to an university it is useless to ask questions about academic degrees. Respective questions should be skipped. Another typical situation includes loops: One might be interested in the degree of skills related to foreign languages. To accommodate speakers fluent in multiple languages, some kind of cycle or loop is needed.

The code example in Figure 1 shows an example of a questionnaire item with a free text entry. The XML syntax is not important here.[2] The item group that is defined in the code snipped defines a single item, i.e. one question is administered. The item has a unique identifier (ID), instruction text and answering possibilities. In this case a free text entry of length 12.

The second code example in Figure 2 shows a routing with two possible targets. This is a hidden item, i.e. an item that is not displayed but used internally.

[2] The XML syntax of the PIAAC BQ has been specially designed for this purpose. At the time of writing of this paper only limited support like editors or visualizers is available.

```
<itemGroup id="CI_PERSID" responseCondition="ALL" layout="list">
  <item id="CI_PERSID">
    <instruction>Please enter the sampled person ID</instruction>
    <responses layout="radioButton">
      <response code="00" freeTextEntry="true"
         freeTextEntrySize="12" > Sampled Person ID:[FTE]</response>
    </responses>
  </item>
</itemGroup>
```

Fig. 1. BQ code example: free text entry

```
<itemGroup id="CI_skip-C-200Rule" layout="list"
   responseCondition="ALL" hidden="true">
<item id="CI_skip-C-200Rule"/>
<routing>
  <condition>
    <operator type="equal">
      <variable name="CI200Rule"/>
      <constant>NI</constant>
    </operator>
  </condition>
  <then>
    <goto itemGroup="CI200Rule"/>
  </then>
  <else>
    <goto itemGroup="CI_start"/>
  </else>
</routing>
</itemGroup>
```

Fig. 2. BQ code example - conditional routing

The routing is conditional. It is based on the value of the variable `CI200Rule`. Based on this variable, the BQ jumps to item `CI200Rule` or `CI_start`.

Each variable can only be written once. There is a one-to-one relationship between an item and a variable. The variable has the same name as the item where it is initialized and written. Afterwards the variable can only be read, not changed or deleted. There is a notable exception to this rule: It is possible to go back in the questionnaire, for example in case of an error noticed later on. If this is done, the variables connected to the items eventually asked again can also be written again.

The PIAAC BQ together with the overall survey workflow contains more than 600 items. It has one single start item and one single end item. It is possible to break-off the interview at many items, but not all. Break-off leads to a special item that asks for the reason for the break-off.

3 BQ Modeling

A basic modeling strategy for background questionnaires is relatively straightforward. Items (and/or item groups) can be modeled for example as states of a finite automata. Going from one question to the other is a matter of transition from one state the the next. Routing can be modeled as conflicting state transitions. We will now have a closer look at this idea and discuss whether it is sufficient below.

3.1 Automata models

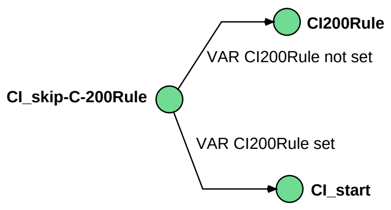


Fig. 3. Finite automata for code in Fig. 2

Figure 3 illustrates the idea of an automata model for the BQ. The automata implements the BQ code example of Figure 2. State `CI_skip-C-200Rule` is connected to the states `CI200Rule` and `CI_start`. The actual transition depends on the variable `CI200Rule` as described above. This data dependency causes problems with this basic model. We will come back to this problem later on.

The benefit even of such a simple formal model is twofold: Once a questionnaire is transformed to a finite automaton, automatic as well as manual inspection is possible. Automatic inspection can check important properties like connectedness and reachability of the final state(s). Manual inspection is enabled by using a graphical tool that displays the complete BQ. This allows for a much more convenient way of getting an overview of the BQ. It is nearly impossible to follow all routings in the sequential XML format, even if this is supported by an appropriate style sheet (XSLT, see [19]) using links and an overview frame. See Figure 8 to get an idea of the complexity of the BQ. Note that this figure only shows a small part of the overall questionnaire. The model shown in the figure is implemented as a Petri net, not an automaton.

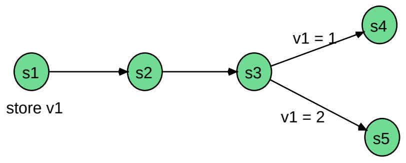


Fig. 4. Data dependent routing

Figure 4 illustrates a general problem with the simple modeling approach. In state `s1` the variable `v1` is written. Afterwards another state (`s2`) is reached and then `s3`. Only in this state the value of `v1` is read again to determine which state (`s4` or `s5`) should be reached next. This means that the variable is used non-locally. While such a situation is common in questionnaires, it is not possible to model a non-local usage of a variable in an ordinary finite automata.[3]

3.2 Petri net models

To overcome the problem of non-local variable usage illustrated above, we remodel the very same BQ part as a Petri net.[4] This can be seen in Figure 5.

As we can see in the figure, the problem can easily be overcome. Items, previously modeled as states in the automaton, are now modeled as places of the Petri net. Transitions have been introduced between the states/places. The places can be seen as the static part, e.g. question or instruction. The transitions are the dynamic part, e.g. the answer given to the respective question and/or the stored variable. Depending on the values stored and retrieved in the variables, the resulting net can be a Place-/Transition net or a colored Petri net.

Place-/Transition nets are possible for variables with restricted (=finite) domains. Luckily, this type is most commonly used in BQs. The vast majority of

[3] This is not completely true, because for variables with finite domains it would be possible to enumerate all reachable states for all values of all variables. Nevertheless, such a model would be hard to read and not very useful.

[4] Petri nets are not an arbitrary choice. They offer various advantages: graphical representation, formal analysis, tool support.

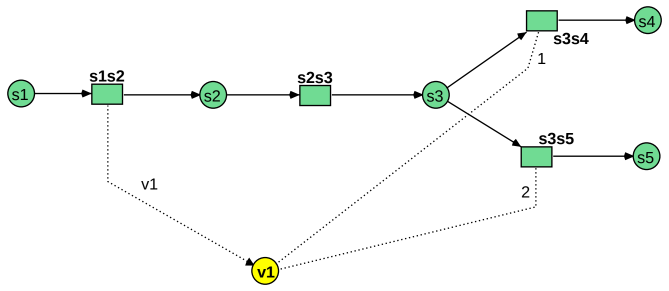


Fig. 5. Data dependent routing of Fig. 4 as PN model

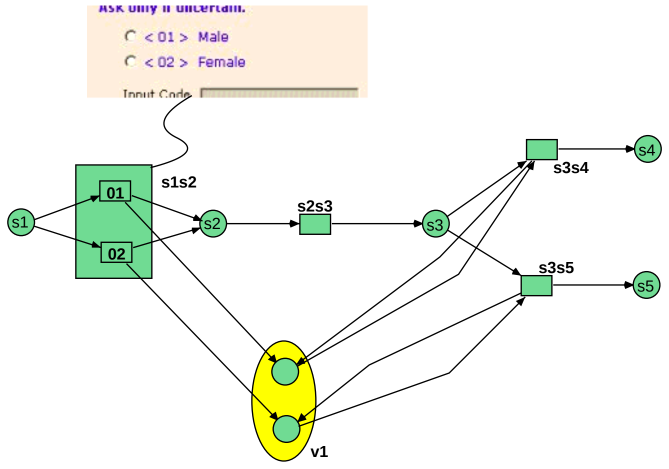


Fig. 6. Refinement of Fig. 4 as PT net

questions is of a single or multiple-choice type. Such a question is illustrated in Figure 6. The first question in this example is about the gender of the interviewee. Only two answers are possible. This can be modeled by a refinement of the net of Figure 5, as shown above. The resulting net is a P/T-net and can be analyzed using the respective tools.

For free text variables or numbers such a modeling would not be possible. Instead, we can use colored nets that support high-level data structures for places and variables directly. While such models are more difficult to analyze they offer other advantages. We will stick to P/T-nets for the moment and come back to the advanced net models later on in Section 5.

4 Modeling and Analysis for PIAAC

The definition, implementation, national adaptation, and deployment of the PIAAC BQ was driven by a high time pressure. Pre-existing questionnaire parts had to be combined and extended. A compromise had to be found that was (a) not too long, (b) implementable world-wide - both a cultural and a political challenge, and (c) able to gather enough data to give answers to the grounding questions of PIAAC. Therefore the work on the BQ started using a semi-structured approach (printable and human-readable Excel sheets), to be able to quickly disseminate all intermediate versions and get feedback. Only late in the process, this was transformed to a well-defined XML format. Therefore also the work on the formal analysis of the BQ started late and is not completely done yet.

The first attempt to get some insight into the BQ structure handled the BQ as a graph. Only an internal model was built, without any graphical representation. Variables were neglected, only the control flow was mapped. From this simple model some important insights were possible: We found dangling routings (jumps to undefined items, e.g. due to spelling mistakes), duplicate item names and isolated nodes - items that could never be reached. On the other hand, it turned out to be quite tedious to verify and analyze the error reports of the first analyzer because of the lack of a graphical representation.

Therefore we implemented another approach targeting on Petri nets. This formalism was chosen to be able to benefit from the advanced set of tools available, allowing to visually inspect a net and formally analyse it at the same time. Our work greatly benefited from the existence and widespread support of the PNML standard - see [8] for an overview or the web site [16] for more information. The usage of PNML allowed to implement the modeling process – modeling a Petri net that represents a specific BQ – as an XSLT transformation.

The first attempt to do so replicated the graph analyzer mentioned above. Variables were neglected, all routing possibilities were handled equally without interpreting the routing conditions. This resulted in a PNML net definition file only containing places, transitions, and arcs. An example can be found in Figure 7. Note that `[...]` means the omitting of plenty of PNML code.

```
<?xml version="1.0" encoding="UTF-8"?>
<pnml>
  <net id="piaac-BQ-DE-001" type="piaac-analyse">
[...]
   <place id="B_C02b1DE2b">
      <name>
        <text>B_C02b1DE2b</text>
      </name>
    </place>
[...]
    <transition id="t_B_C02b1DE2b_B_Q02b2DE2_32">
      <name>
        <text>t_B_C02b1DE2b_B_Q02b2DE2_32</text>
      </name>
    </transition>
[...]
    <arc id="B_C02b1DE2b_t_B_C02b1DE2b" source="B_C02b1DE2b"
       target="t_B_C02b1DE2b">
      <inscription>
        <text>1</text>
      </inscription>
    </arc>
[...]
  </net>
</pnml>
```

Fig. 7. Example PNML Code

Our modeling approach does not generate any graphical information. Therefore a tool had to be found that is able to import PNML files, construct a graphical representation automatically, and analyse the P/T-net. We chose the ProM tool for this purpose. For more information on ProM, see [18] and [17]. ProM especially well supports the handling of large nets and arranges the net elements in a way that is very well readable for the human eye.

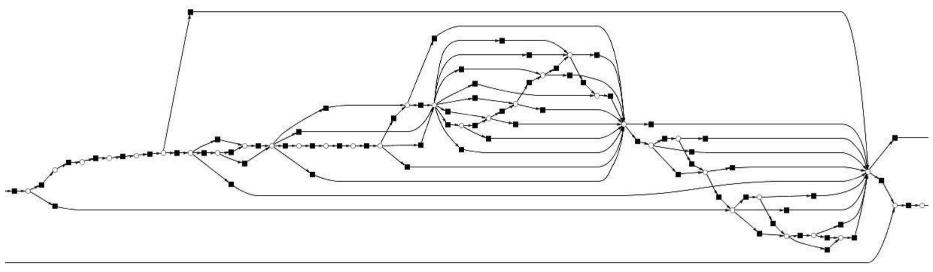


Fig. 8. BQ part as a P/T-net

In Figure 8 a small part of the complete BQ net is shown. As said before this is a simple model in the sense that the variables have been omitted. The figure is presented here just for illustration purposes. It serves to get an impression of the complexity of the overall BQ model. The complete model is way too big to be presented here. The layout of the example net has been done automatically by ProM.

Once available as a P/T-net in ProM, the build-in analysis means can be used. The PIAAC BQ has a single start item and a single end item. All items should be reachable and there may not be a dead end. The resulting BQ net therefore has to be a net with workflow properties. This is easily analyzable in ProM and gives good insight into the BQ definition. Doing so, we were able to find all the error types mentioned above with the big advantage of directly *seeing* the problems in the net graph. It now is way simpler to find fixes for the errors.

5 Advanced Net Models

In this section, we discuss experimental models that have not been used so far for the complete BQ. Nevertheless, as this is ongoing work, this will change soon.

To get deeper insight into the formal properties of a BQ, factoring in the variables and (routing) conditions is necessary. However, this may lead to way more complicated models, as we can see from a simple example. For this, we extend the example of Figure 6 to four possible answer categories on item `s1`, two answer categories on item `s2` and three conditional routings after `s3` relying on the variables `v1` and `v2`:

– `s4, if v1 = 1 and v2 = 1`
– `s5, if v1 = 2`
– `s6, if v1 > 2 or v2 = 2`

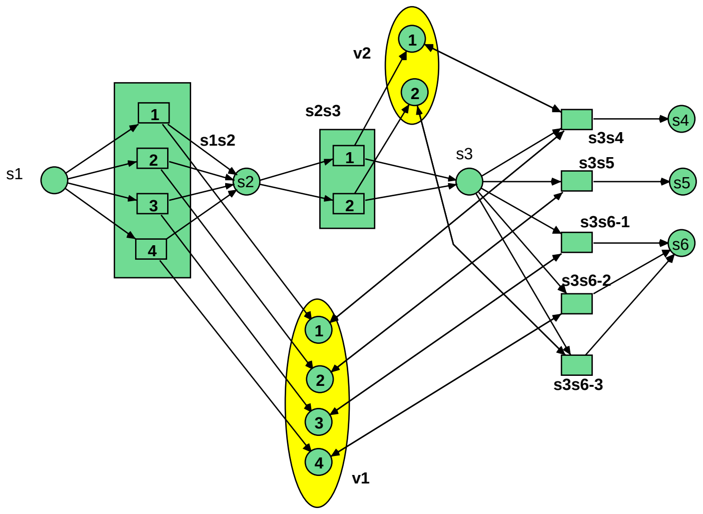


Fig. 9. Additional net elements for variables and conditions

Even this mild extension leads to a more complicated net model. Especially the last routing condition (leading to `s6`) is interesting, as it has to be unfolded to three transitions: two for the ”greater-than” and and extra for the ”or”-part. In real settings, this can easily grow to huge amounts of transition. As an example, in the German BQ there are routing conditions combining 5 variables, each enclosing up to 16 possible values, to route to more than 10 targets.

Another possibility is to model colored Petri nets instead of P/T-nets. Colored Petri nets directly support high-level data structures and variables. Nevertheless, they still allow for analysis, the necessary unfolding process is done inside the tool, for example the CPN Tools [4,10]. This option is under investigation.

In the moment, the PIAAC BQ is defined by means of writing XML code. This is tedious and error-prone work. The direct syntax can be checked relatively easily, but syntactical errors like missing routing targets are harder to detect. Semantic errors like dead ends even harder. Parts of these problems could be overcome by using a high-level Petri net formalism like Workflow Nets [9,15] as a means for rapid prototyping and/or adding small changes and corrections. Workflow Nets are directly executable, so that changes can be tried out easily. The graphical modeling permits typical errors mentioned above and gives a good overview on what one is doing. To support large BQ models, means for abstraction and rapid modeling are necessary. Workflow Nets offer such means.

Abstraction is possible in form of object tokens of the underlying reference net formalism [12,13,14] and by dynamic transition refinement [11]. Rapid modeling is facilitated by workflow patterns, a special form of net components - see [1] for an overview.

6 Conclusions and Outlook

We found a way of making use of well-known and well-understood formalisms and tools for a new domain. While some good results have already been achieved, several ways of extending the work are possible:

– The BQ analysis should be integrated into the normal BQ definition and release process. In the moment, it still requires manual work. It has been done completely only for the German version of the BQ.
– While the single steps of the analysis approach are rather straightforward, the combination still requires some manual work. This should be simplified to allow non-expert users to do the analysis on their own.
– The advanced models of Section 5 can be used directly for analysis of the BQ. This is still in an experimental state. We try to partition the BQ net into independent sub-nets to circumvent the net size explosion.
– In the moment, the BQ definition and especially the national adaptation process has long turn-around times. Countries request changes without the possibility to try them out beforehand. Using the rapid prototyping idea of Section 5 they could first try out the changes themselves and only request approval afterwards, once the changes are stable and working on the national level.

The examples and the analysis shown in this paper could partly be modeled using a sequential modeling formalism. However, Petri nets offer big advantages when it comes to non-local dependencies. For example, in the PIAAC BQ, some of the questions should only be asked a limited number of times in a country. This is straightforward to model in PN but maybe more difficult in other modeling formalisms.

The analysis of background questionnaires could benefit a lot from a sound formalization of what a BQ is. As mentioned early in the paper, no such definition exists so far. This question needs further research. Especially the similarity of BQs and workflows should be analyzed more deeply.

References

1. Lawrence Cabac. Net components: Concepts, tool, praxis. In Daniel Moldt, editor, *Petri Nets and Software Engineering, International Workshop, PNSE'09. Proceedings*, Technical Reports Université Paris 13, pages 17–33, 99, avenue Jean-Baptiste Clément, 93 430 Villetaneuse, June 2009. Université Paris 13.
2. Centre Research Public Henri Tudor (CRP-HT). http://www.tudor.lu. WWW.
3. Educational Testing Service (ETS). http://www.ets.org. WWW.

4. Computer Tool for Coloured Petri Nets (CPN Tools). http://wiki.daimi.au.dk/cpntools/cpntools.wiki. WWW.
5. Organization for Economic Co-Operation and Development (OECD). http://www.oecd.org. WWW.
6. Programme for International Student Assessment (PISA). http://www.pisa.oecd.org. WWW.
7. Programme for the International Assessment of Adult Competencies (PIAAC). www.oecd.org/els/employment/piaac. WWW.
8. L.M. Hillah, E. Kindler, F. Kordon, L. Petrucci, and N. Trèves. A primer on the petri net markup language and iso/iec 15909-2. *Petri Net Newsletter*, 2010.
9. Thomas Jacob, Olaf Kummer, Daniel Moldt, and Ulrich Ultes-Nitsche. Implementation of workflow systems using reference nets – security and operability aspects. In Kurt Jensen, editor, *Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark, August 2002. University of Aarhus, Department of Computer Science. DAIMI PB: Aarhus, Denmark, August 28–30, number 560.
10. Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer (STTT)*, Volume 9(3):213–254, 2007.
11. Michael Köhler and Heiko Rölke. Dynamic transition refinement. *Electronic Notes in Theoretical Computer Science*, 175:119–134, June 2007.
12. Olaf Kummer. *Referenznetze*. Logos Verlag, Berlin, 2002.
13. Olaf Kummer, Frank Wienberg, Michael Duvigneau, and Lawrence Cabac. Renew – the Reference Net Workshop. Available at: http://www.renew.de/, August 2009. Release 2.2.
14. Olaf Kummer, Frank Wienberg, Michael Duvigneau, and Lawrence Cabac. *Renew – User Guide*. University of Hamburg, Faculty of Informatics, Theoretical Foundations Group, Hamburg, release 2.2 edition, August 2009. Available at: http://www.renew.de/.
15. Daniel Moldt and Heiko Rölke. Pattern based workflow design using reference nets. In Wil van der Aalst, Arthur ter Hofstede, and Mathias Weske, editors, *Proceedings of International Conference on Business Process Management, Eindhoven, NL*, volume 2678, pages 246–260, 2003.
16. Petri Net Markup Language (PNML). http://www.pnml.org/. WWW.
17. Process Mining Toolkit (ProM). http://prom.win.tue.nl/tools/prom/. WWW.
18. W.M.P. van der Aalst, B.F. van Dongen, C. Günther, A. Rozinat, H. M. W. Verbeek, and A. J. M. M. Weijters. Prom: The process mining toolkit. In *Proceedings of the BPM 2009 Demonstration Track, Volume 489 of CEUR-WS.org, Ulm, Germany*, 2009.
19. XSL Transformations (XSLT). http://www.w3.org/tr/xslt. WWW.

Detecting and Repairing Unintentional Change in In-use Data in Concurrent Workflow Management System

Phan Thi Thanh Huyen and Koichiro Ochimizu

School of Information Science, Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa, 923-1292, Japan
{huyenttp, ochimizu}@jaist.ac.jp

Abstract. Workflow verification has attracted a lot of attention, especially control flow aspect. However, little research has been carried out on data verification in workflow literature although data is one of the most important aspects of workflow. This paper proposes an approach for detecting and repairing **Unintentional Change in In-use Data** (*UCID*) in a Concurrent Workflow Management System at build time. We define UCID as a situation in which some data values are lost or some data elements are assigned values different from the intentions of workflow designers due to non-deterministic access to shared data by different activities. Differently from previous studies, we consider UCID in two different ways: between concurrent activities in a single workflow (*intra-UCID*) and between activities in different concurrent workflows (*inter-UCID*). In this paper, we first investigate UCID situations in a workflow management system, and then we define a Time Data Workflow, an extension of the WF-Nets with time and data factors, with many attributes supporting UCID detection and correction. Based on these definitions, we develop an algorithm which helps to detect potential intra/inter-UCID at build time, along with algorithm evaluation and UCID resolution methods. Finally, we introduce a concrete project on building a change support environment for cooperative software development using UCID theory.

Keywords: Unintentional Change in In-use Data, Time Data Workflow, concurrent workflows, algorithm, Workflow Nets

1 Introduction

Correctness of a workflow model is very important, because any errors in workflow can lead to execution failure of the corresponding process. Therefore, workflow should be verified carefully before execution to reduce risks to the target process. Workflow verification has received a lot of attention since the birth of the workflow concept. However, researchers have only focused on structure verification, temporal verification and resource verification [2] [4] [7] [9]. Most verification techniques ignore data aspect and there is little support for data flow verification. Previous works on the data flow aspect have concentrated on detecting common data flow errors such as missing data, redundant data, inconsistent data, garbage data, etc. Among them, Unintentional Change in In-use Data (UCID) is perhaps one of the most dangerous and common problems. We define UCID as a situation in which some data values are lost or some data elements are assigned values different from the intentions of workflow designers due to non-deterministic access to shared data by different activities. Assuming that workflow is free of control errors, and activities in workflow can be scheduled within temporal constraint, we aim to support data verification in the workflow model by concentrating on UCID detection and correction.

Existing approaches have addressed this problem by detecting potential UCID patterns, limited to concurrent activities of a single workflow. Unfortunately, this error can cross a single workflow boundary. In a Workflow Management System (WFMS), in fact, there exist many workflows executing at the same time, which we call *Concurrent Workflows*, and they may be correlated if two activities from different workflows use shared data. Even if the data flow of each workflow is correct, we cannot ensure correctness of the whole system because of the mutual interactions among workflows. The problem is how to detect non-deterministic access to shared data of activities belonging to not only the same workflow but also different workflows and how to repair this kind of data abnormality.

Reference [19] is our first efforts in handling the UCID problem. Potential UCID situations, Time Data Workflow (TDW) concepts, along with two algorithms for detecting intra-UCID and inter-UCID have been introduced in [19]. This paper is a refined and extended version of the [19]. In this paper, we redefine TDW as an extension of Workflow Nets (WF-Nets) [8] instead of Petri Nets as before. Based on these definitions and two algorithms for detecting intra/inter-UCID in [19], a revised version of UCID detection algorithm is built. Compared with the previous ones, this revised algorithm is more accurate and useful. Furthermore, some heuristics for making the algorithm more flexible and effective are discussed. UCID resolution methods are also proposed in this paper. Then, we illustrate this theory in practice by using it in designing workflows which represent change activities in a software change process.

Our approach in UCID detection is to observe behaviors of concurrent activities having data relation. In the case of activities in the same workflow, their total orders can be decided based on control flow. However, control flow does not help in the case of activities in different workflows. Therefore, we must use activities' execution time attribute to identify their total orders. Regarding UCID resolution, we take advantage of composition features of the Petri Nets to create new workflows with UCIDs removed.

The rest of this paper is organized as follows. Section 2 discusses the motivation of our research. Section 3 defines the Time Data Workflow (TDW), an extension of the Workflow Net with time and data factors. Section 4 introduces UCID situations caused by concurrent activities in the same workflow (*intra-UCID*) or activities in different concurrent workflows (*inter-UCID*) [19]. An algorithm for detecting potential UCID in both cases of intra/inter-UCID at build time, along with algorithm evaluation, is given in Section 5. Section 6 presents UCID resolution methods. Section 7 introduces our project on building a change support environment for cooperative software development. Theory about UCID problem is employed in this project to detect and repair data abnormalities among concurrent Change Support Workflows. Section 8 reports on related work and finally, Section 9 concludes the paper and discusses points to future work.

2 Motivation

Let's take an example. We have two workflows W_1 and W_2, which are being executed independently. Workflow activities are modeled by rectangles, and data modified by an activity are written inside the corresponding rectangle. A small arrow is attached to a rectangle to denote an activity which is being executed. Data of the system are stored in a central repository. W_1 has five activities which modify A, X, B, C and D respectively. B and D are modified based on the value of X created by A_{12}. W_2 also has five activities which modify E, X, F, G and H respectively. Both A_{12} and A_{22} will modify X, but designers of W_1 and W_2, who don't have a comprehensive view of the whole system, may not recognize this problem. This is a common problem, especially in a big system with many workflows.

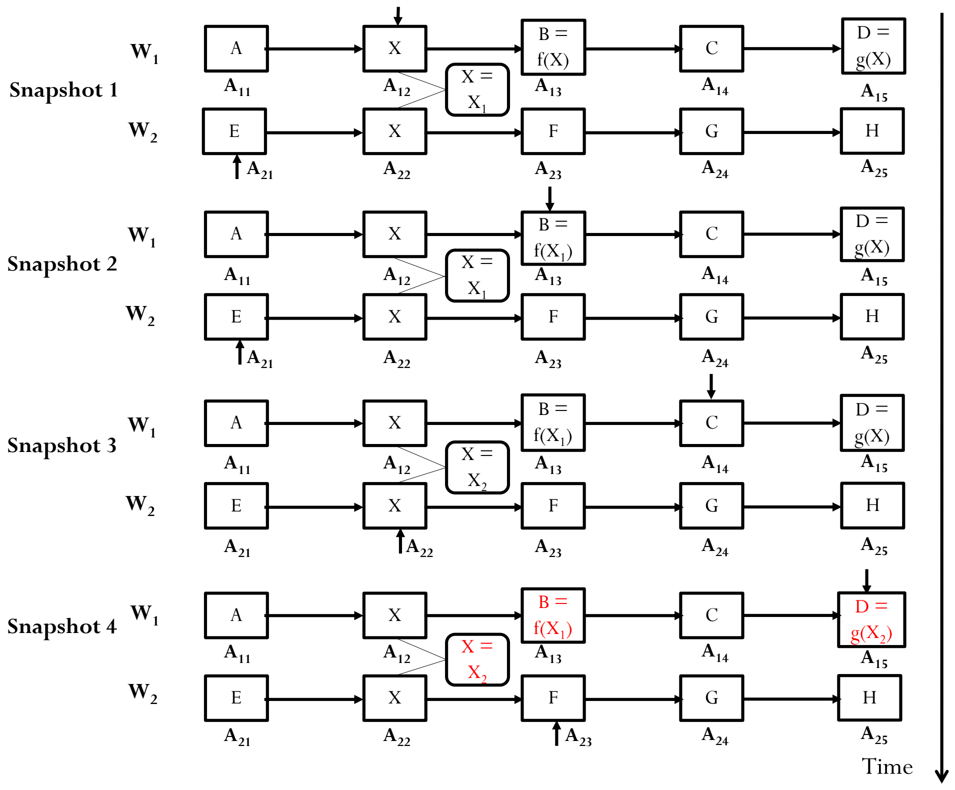


Fig. 1. Motivating example

Figure 1 describes some snapshots of the system at different time. For simplicity, we concentrate on describing the change in value of data elements relating to shared data X. In snapshot 1, A_{12} changes value of X to X_1. In snapshot 2, A_{13} changes value of B based on the value of X, X_1. In the next snapshot, A_{22} changes value of X from X_1 to X_2. In the last snapshot, A_{15} changes value of D based on the current value of X which is X_2. If X_1 is different from X_2, there are two problems in this scenario: X_1 is lost and D is assigned an unexpected value because D is modified based on the value X_2 instead of the value created by activity A_{12}, X_1. This is different from the intentions of the designers of the workflow W_1 and may cause an inconsistency between B and D. Regarding our definition of UCID, these errors are categorized into inter-UCID errors.

The first problem is similar to the lost update problem in database theory. Lost update problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect [20]. In this case, version control systems (VCSs) can be used if data of the system are individual artifacts like documents, source codes, etc. Version control is the management of changes to documents, programs, and other information stored as computer files. Changes are usually identified by a number or letter code, termed the "revision number". Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.

Unfortunately, VCS cannot help to avoid the second problem. In this situation, if data of the system are stored in a central database, the database management system (DBMS) can provide some concurrency control techniques, which are used to ensure the noninterference or isolation property of concurrently executing transactions such as locking techniques, timestamp ordering based techniques, etc. A database transaction is a transaction which satisfies the ACID (atomicity, consistency, isolation and durability) properties. These properties should be enforced by the concurrency control and recovery methods of the DBMS [20]. However, in this method, we must specify the boundary of each transaction. This requirement is difficult to implement because there are many people involved in a workflow and people in a workflow may know nothing about other workflows. If the whole workflow is considered as a unique database transaction, it is impractical because a workflow may use many data elements and may happen for a long time.

If this type of errors is discovered at runtime, a recovery mechanism must be performed to ensure the correctness of the whole system. However, recovery is a rather expensive work, especially in a cooperative environment with many concurrently executing workflows. Therefore, detecting these errors as soon as possible is necessary to reduce risk to the target process.

This paper examines UCID situations in a general basic system without concerning which type of workflow data is stored in the central repository of the system and the implementation of the central repository as well.

Regarding inter-UCID, our problem domain is workflows whose data and estimated execution time can be decided at the design phase, for example workflows in the software evolution process. In these cases, an early UCID detection will help workflow designers to have a more comprehensive view of the system, and make timely adjustments to the original workflows to avoid error at runtime. We assume that the following steps are conducted before workflow execution: identifying workflow activities and their orders, assigning activity properties (data, time…), and checking error using UCID detection and correction theory. If some potential UCID errors are detected, the first and second steps should be re-executed, based on suggested solutions given by UCID detection system.

With reference to workflows in which estimated execution time is not available at design time, UCID patterns and detection method will be used to detect UCID errors from workflow execution histories. However, this is out of the scope of this paper.

3 Time Data Workflow (TDW)

There are many ways to model a workflow, such as directed graphs, UML activity diagram, PERT, etc. In this paper, we chose the WF-Nets based approach to model workflow process, because it has many useful features needed in the area of business process modeling besides the mathematical nature of the underlying Petri Nets formalism [17].

WF-Nets is a subclass of Petri Nets dedicated for process/workflow modeling and analysis. Petri Nets is a popular graphical and mathematical modeling language in describing and analyzing systems which are characterized as concurrent, asynchronous, distributed, parallel, nondeterministic and/or stochastic [17]. Formally, Petri Nets is a tuple PN = (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 (flow relation) [8]. A Petri Nets PN = <P, T, F> is a WF-Nets if and only if there is one source place $i \in P$, one sink place $o \in P$ such that $\bullet i = \emptyset$, $o\bullet = \emptyset$, and every node $x \in P \cup T$ is on a path from i to o [8].

Our Time Data Workflow (TDW) is an extension of WF-Nets with time and data factors. Time and data are represented as attributes of transitions in a TDW. In this paper, we consider two types of relationships between an activity and a data element. First, an activity may *read* a particular data element as its input data. Second, an activity may *write* a particular data element as its output data. This means that this data element is assigned a new value. Inside an activity, *read* always happens before *write*. Assuming that durations of activities can be estimated at build time, we augment each activity A with two time values *min(A), max(A)* which describe the minimum and maximum execution durations of A respectively. The time unit is selected depending on specific workflow applications. Based on reference point P, which is the start time of its corresponding workflow, we can infer the *Earliest Start Time, EST(A)*, and the *Latest Finish Time, LFT(A)*, of A at run time. If *S(A), F(A)* are the *Start Time* and *Finish Time* of this activity at run time respectively, we can conclude that the *Active Interval* of A, *[S(A), F(A)]*, is within its *Estimated Active Interval, [EST(A), LFT(A)]*, that is, $[S(A), F(A)] \subseteq [EST(A), LFT(A)]$ [19].

In a TDW, activities are modeled by transitions, and causal dependencies are modeled by places and arcs, as shown in Figure 2 [19]. Building blocks such as the AND-split, AND-join, OR-split, OR-join are used to model sequential, conditional, parallel and iterative control structures of workflows. AND-split and OR-split transition correspond to transitions with two or more output places, while AND-join and OR-join transition correspond to transitions with multiple incoming arcs. Different symbols are attached to original rectangles to distinguish normal transitions from transitions containing branching conditions. Figure 2a illustrates a typical transition in a TDW, with execution duration ranging from d1 to d2; data elements a, b are inputs and c, d, e are outputs. The other parts of Figure 2 show how basic constructions of a workflow are represented by TDW's notations [19]. For the sake of simplicity, each activity is represented by a transition. Therefore, the terms 'activity' and 'transition' are interchangeably used in this paper.

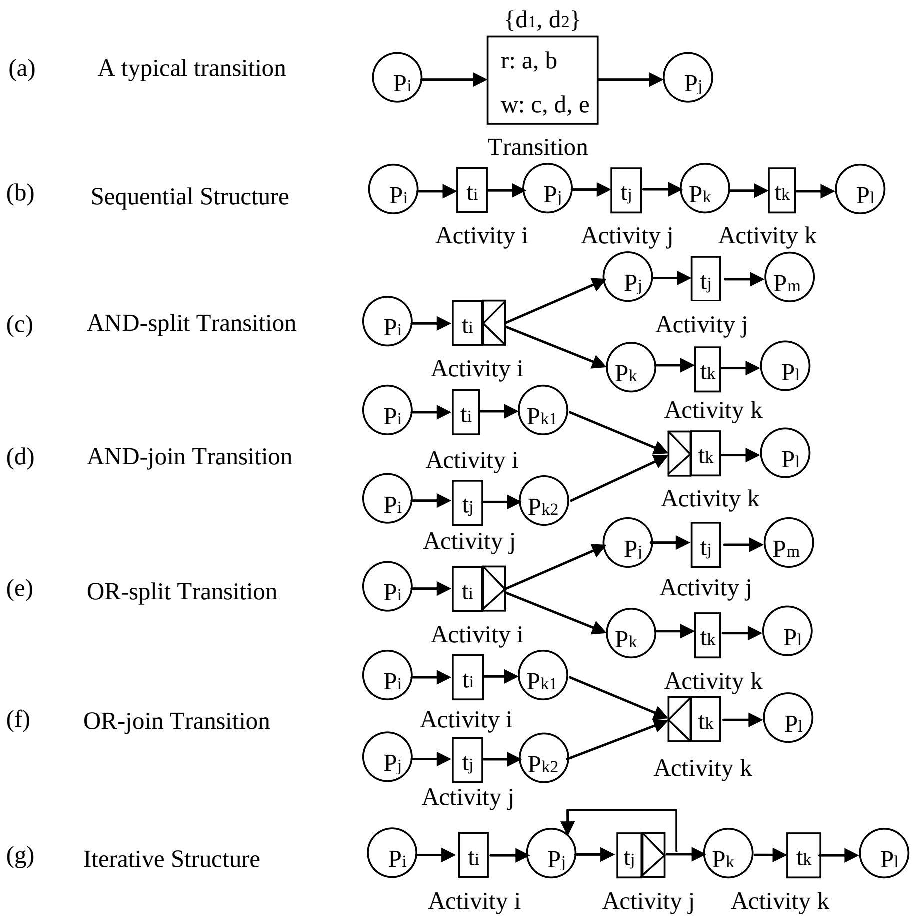


Fig. 2. Workflow primitives specified by TDW

As an extension of WF-Nets, TDW specifies the time and data properties of a single case in isolation, assuming that different cases are completely independent from each other. Therefore, UCIDs are caused by activities in a single TDW instance or activities belonging to workflow instances of different TDWs. Without the loss of generality, we assume that each TDW has one instance only.

Definition 1 (Time Data Workflow – TDW) A TDW, w, is a tuple <P, T, F, id, D, R, DE, TI > where:

- <P, T, F> is a WF-Nets with places P, transitions T and arcs F
- id is the workflow identifier.
- D is a set of data elements.
- R = {r, w, u} is a set of possible access rights to data elements (r: ***read***, w: ***write***, u: ***use*** (either ***read*** or ***write***)).
- DE: T x R$\rightarrow 2^D$ is a function that returns a set of data elements associated with a transition and an access right.
- TI: T $\rightarrow R^+$ x (R^+ x ∞) is a time interval function that returns minimum and maximum execution durations of a transition.

Definition 2 (Concurrent Time Data Workflow Model) A Concurrent TDW Model cwm = (W, T_{wm}) is a collection of TDWs which have overlapping execution times (concurrent TDWs):

- W = {w_1, w_2… w_n} is a set of concurrent TDWs, where w_i = < P, T, F, id, D, R, DE, TI >.
- T_{cwm} = T(w_1) $\cup$ T(w_2) $\cup$ … $\cup$ T(w_n) is the set of all transitions (activities) in cwm.

Given a TDW w as in Definition 1, we have the following definitions [19]:

Definition 3 (Path) A Path is a sequence of consecutive arcs.
A sequence p = (x_o, x_1, …, x_k) is a Path iff $\forall$ i, 0 < i < k – 1: (x_i, x_{i+1}) $\in$ F

Definition 4 (Transition Path) A sequence p = (t_0, p_1, t_1,… , t_k) is a Transition Path iff it is a path and t_0, t_k $\in$ T.

Definition 5 (Transition Reachability) Transition t_i is reachable from t_j if there exists a transition path (t_i,... , t_j) on wm.
Reachable (t_i, t_j) = true iff $\exists$ transition path p = (t_i,... , t_j)

Definition 6 (Transition Distance) Given two transitions t_i, t_j where Reachable (t_i, t_j) = true or where Reachable (t_j, t_i) = true, the Transition Distance between t_i, t_j is the length of the shortest path between them.

Definition 7 (Nearest Common Transition) Given two transitions t_i, t_j where Reachable (t_i, t_j) = false and where Reachable (t_j, t_i) = false, their Nearest Common Transition is the common transition which has the shortest distances to both of them, denoted as t_{nct}.

Definition 8 (Closest Data Relation Transition) Given two transitions t_i, t_j, where their nearest common transition is not an OR-split transition, t_j is called the Closest Data Relation Transition of t_i on data element d if t_j just precedes t_i in terms of time, and both t_j and t_i *use* (*read*/*write*) d, denoted as t_{cdrt}.

4 UCIDs in a Concurrent TDW Management System

A Concurrent TDW Management System is a workflow management system which is responsible for TDW construction and management. A module of UCID detection and correction is also integrated into this system.

Data flow can be implemented explicitly as a part of the workflow model by using a separate channel to pass data from one activity to another. Otherwise, it can also be implemented implicitly through a control flow or process data store [3]. The process data store is basically a central repository where all workflows' activities can access or update their data. We choose implicit data flow through the process data store as a basis for our approach. In this implementation model, UCID may occur, particularly in cases involving concurrent execution paths.

Given a Concurrent TDW Model cwm as in Definition 2, we have the following definitions:

Definition 9 (Data Relation) Two activities a_i, a_j ($i \neq j$) have data relation if $DE(a_i, u) \cap DE(a_j, u) \neq \emptyset$ [19].

Definition 10 (Concurrent activities) Two activities are called concurrent activities iff they belong to two parallel branches of a TDW or they are in different TDWs and have overlapping Active Intervals.

Definition 11 (Unintentional Change in In-use Data) A situation in which some data values are lost or some data elements are assigned values different from the intentions of workflow designers due to non-deterministic access to shared data by different activities [19].

Here we distinguish two kinds of UCID: intra-UCID and inter-UCID. The former considers UCID situations concerning concurrent activities in the same workflow, while the latter is related to concurrent activities in different workflows. Definition 12, 13 are based on definitions of read-write conflict and write-write conflict in [1].

Definition 12 (RW Intra-UCID) A situation in which an activity A tries to ***read*** data from a shared variable x and an activity B tries to ***write*** data to the same shared variable *x* and vice versa, where A, B are concurrent activities in the same workflow.

Definition 13 (WW Intra-UCID) A situation in which two concurrent activities in the same workflow, A and B, try to ***write*** data to the same shared variable.

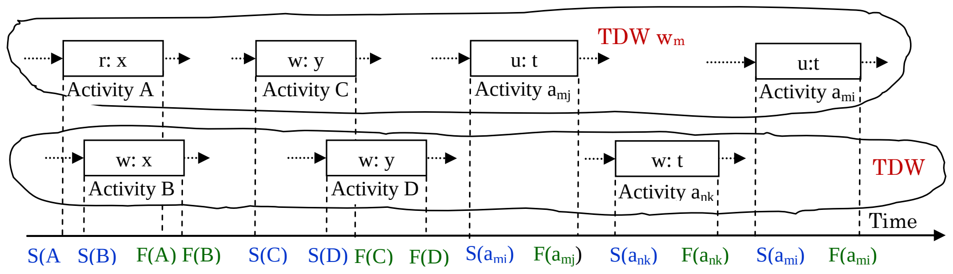


Fig. 3. Inter-UCIDs

Definition 14 (RW Inter-UCID) A situation in which an activity A tries to ***read*** data from a shared variable x and an activity B tries to ***write*** data to the same shared variable *x* and vice versa, where A, B are in different concurrent workflows and have overlapping Active Interval ($[S(A), F(A)] \cap [S(B), F(B)] \neq \emptyset$).

Definition 15 (WW Inter-UCID) A situation in which two activities A and B try to ***write*** data to the same shared variable, where A, B are in different concurrent workflows and have overlapping Active Interval ($[S(A), F(A)] \cap [S(B), F(B)] \neq \emptyset$).

Definition 16 (UWU Inter-UCID) A situation in which there are inconsistent views of shared data by two activities in the same workflow, because their shared data are ***written*** externally by an activity in a different concurrent workflow.

As depicted in Figure 3, two activities a_{mi}, a_{mj} of TDW w_m ***use*** (***read*** or ***write***) data element t, where a_{mj} is the closest to a_{mi} in terms of time and $F(a_{mj}) < S(a_{mi})$, which means $t_{cdrt}(a_{mi}, t) = a_{mj}$. A UWU Inter-UCID happens because activity a_{nk} of a different workflow w_n ***writes*** to t within the time interval $[F(a_{mj}), S(a_{mi})]$. RW Inter-UCID and WW Inter-UCID also happen between activity A and activity B, activity C and activity D respectively.

5 Detection of Potential UCID in a Concurrent TDW Management System

Regarding UCID definitions, inter-UCIDs are identified based on the Active Interval of activities having data relation. However, Active Interval of an activity can only be determined at runtime when it has finished its execution, and hence Estimated Active Interval is used instead of Active Interval to find potential UCID at build time, before a new TDW is put into the Concurrent TDW Management System to start.

5.1 Calculation of Estimated Active Interval [19]

Designating the start time of a TDW w as a reference point, P_w, we can infer the Estimated Active Interval of an activity A [EST(A), LFT(A)] with respect to its minimum and maximum executing durations {min(A), max(A)} and basic control structures.

Let us say that A_s is the Start activity of a TDW w, then we have $EST(A_s) = P_w$ and $LFT(A_s) = P_w + max(A_s)$. For executing activity A, EST(A) = S(A) and LFT(A) = F(A) if A has been completed.

Sequential Connection (Figure 2b)

$EST(A_j) = EST(A_i) + min(A_i)$; $LFT(A_j) = LFT(A_i) + max(A_j)$

AND-Split Connection (Figure 2c)

$EST(A_j) = EST(A_i) + min(A_i)$; $LFT(A_j) = LFT(A_i) + max(A_j)$

$EST(A_k) = EST(A_i) + min(A_i)$; $LFT(A_k) = LFT(A_i) + max(A_k)$

AND-joint Connection (Figure 2d)

$EST(A_k) = MAX\{EST(A_i) + min(A_i); EST(A_j) + min(A_j)\}$

$LFT(A_k) = MAX\{ LFT(A_i), LFT(A_j)\} + max(A_k)$

OR-Split Connection (Figure 2e)

$EST(A_j) = EST(A_i) + min(A_i)$; $LFT(A_j) = LFT(A_i) + max(A_j)$

$EST(A_k) = EST(A_i) + min(A_i)$; $LFT(A_k) = LFT(A_i) + max(A_k)$

OR-joint Connection (Figure 2f)

$EST(A_k) = MIN\{EST(A_i) + min(A_i); EST(A_j) + min(A_j)\}$

$LFT(A_k) = MAX\{ LFT(A_i), LFT(A_j)\} + max(A_k)$

5.2 Potential UCID Detection Algorithm

Given a Concurrent TDW Model cwm = (W, T_{cwm}), where W = {w_1, w_2, …, w_k} and $T_{cwm} = T(w_1) \cup T(w_2) \cup \ldots \cup T(w_k)$, w = <P, T, F, id, D, R, DE, TI>. The main idea of this algorithm is to select one activity and compare it with the other activities. If two activities have data relation, we will check if there is a potential UCID. In the case of concurrent activities in the same workflow, potential intra-UCIDs can be detected with respect to Definitions 12, 13. If two compared activities are in different workflows and have overlapping Estimated Time Intervals, there is a possibility of an RW/WW inter-UCID occurrence (Definitions 14, 15). If only the data relation exists and one activity occurs before the other, we will compare this situation with the definition 16 and the pattern in Figure 3 to find out a potential UWU inter-UCID.

Step 1: **Initialization:**

1.1 Let *S* be a set of unchecked activities. S is initialized with all unfinished activities of T_{cwm};

1.2 Calculate Estimated Active Interval for all activities in S;

1.3 flag = TRUE is a Boolean variable*;*

Step 2: For every pairwise of activities (a_{mi}, a_{nk}) in S, execute the following steps:

2.1 Check their Data Relation

Let U_{mnik} be the set of shared data between a_{mi} and a_{nk}: $U_{mnik} = DE(a_{mi}, u) \cap DE(a_{nk}, u)$;

  2.1.1 If $U_{mnik} = \emptyset$, a_{mi} and a_{nk} do not have data relation. Therefore UCID cannot happen between a_{mi} and a_{nk};

  2.1.2 If $U_{mnik} \neq \emptyset$, a_{mi} and a_{nk} have data relation. Take the next step.

2.2 If a_{mi} and a_{nk} in the same workflow, check intra-UCID possibility. Otherwise, check inter-UCID possibility;

2.2 Check intra-UCID possibility

  2.2.1 If a_{mi} and a_{nk} belong to two parallel branches of a workflow, this means that their Nearest Common Transition, denoted as t_{nct} (a_{mi}, a_{nk}), is an AND-split transition, they are concurrent activities. Take the next step;

  2.2.2 For every data element, denoted as d_{mnikl}, in U_{mnik}, check the access right to d_{mnikl} of a_{mi} and a_{nk}:

    2.2.2.1 If both of them have *write* access right to d_{mnikl}, this means that $d_{mnikl} \in DE(a_{mi}, w)$ and $d_{mnikl} \in DE(a_{nk}, w)$, then flag = FALSE. There is a potential WW Intra-UCID between a_{mi}, a_{nk} on d_{mnikl};

    2.2.2.2 If one activity has *write* access right to d_{mnikl} and the other has *read* access right to d_{mnikl}, this means that ($d_{mnikl} \in DE(a_{mi}, w)$ and $d_{mnikl} \in DE(a_{nk}, r)$) or ($d_{mnikl} \in DE(a_{mi}, r)$ and $d_{mnikl} \in DE(a_{nk}, w)$), then flag = FALSE. There is a potential RW Intra-UCID between a_{mi}, a_{nk} on d_{mnikl};

2.3 Check inter-UCID possibility /* Figure 3***/**

  2.3.1 If a_{mi} and a_{nk} have overlapping Estimated Active Interval, this means that $[EST(a_{mi}), LFT(a_{mi})] \cap [EST(a_{nk}), LFT(a_{nk})] \neq \emptyset$, they are potential concurrent activities: check RW/WW inter-UCID possibility. Otherwise, check UWU inter-UCID possibility;

2.3.2 Check potential RW/WW inter-UCID

For every data element, denoted as d_{mnikl}, in U_{mnik}, check the access right to d_{mnikl} of a_{mi} and a_{nk}:

2.3.2.1 If both of them have *write* access right to d_{mnikl}, this means that $d_{mnikl} \in DE(a_{mi}, w)$ and $d_{mnikl} \in DE(a_{nk}, w)$, then flag = FALSE. There is a potential WW Inter-UCID between a_{mi}, a_{nk} on d_{mnikl};

2.3.2.2 If one activity has *write* access right to d_{mnikl} and the other has *read* access right to d_{mnikl}, this means that ($d_{mnikl} \in DE(a_{mi}, w)$ and $d_{mnikl} \in DE(a_{nk}, r)$) or ($d_{mnikl} \in DE(a_{mi}, r)$ and $d_{mnikl} \in DE(a_{nk}, w)$), then flag = FALSE. There is a potential RW Inter-UCID between a_{mi}, a_{nk} on d_{mnikl};

2.3.3 Check potential UWU inter-UCID

Assume that $LFT(a_{nk}) < EST(a_{mi})$. For each data element, denoted as d_{mnikl}, in U_{mnik} where a_{nk} has *write* access right to d_{mnikl}: $d_{mnikl} \in DE(a_{nk}, w)$, perform the following steps:

2.3.3.1 Find out the Closest Data Relation Transition of a_{mi} on d_{mnikl}, denoted as a_{mj}: $a_{mj} = t_{cdrt}(a_{mi}, d_{mnikl})$. If $a_{mj} = \emptyset$, UWU inter-UCID may not happen;

2.3.3.2 If $[EST(a_{nk}), LFT(a_{nk})] \subset [LFT(a_{mj}), EST(a_{mi})]$, then flag = FALSE. There is a potential UWU Inter-UCID among a_{mi}, a_{mj}, a_{nk} on d_{mnikl};

Step 3: Return flag.

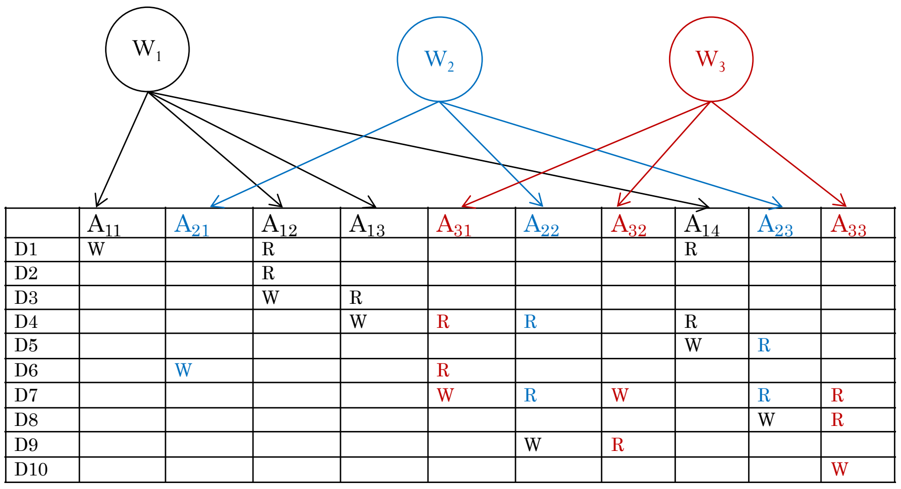
5.3 Algorithm Evaluation

Let's say *n* is the number of unfinished activities in a Concurrent TDW Model *cwm*. In general, we must inspect n^2 combinations of any two unfinished activities to find out some potential UCIDs. This approach allows us to detect not only potential UCID at build time of pre-executed TDWs, but also potential UCID at run time of running TDWs by recalculating the Estimated Active Intervals of their unfinished activities more accurately based on the Active Interval of finished activities. However, depending on applications, we can reduce the number of checking steps by considering some of the following heuristics:

– A two dimensional table can be used to record the access right on data elements of activities in a Concurrent TDW Model *cwm*. Figure 4 describes an example of data flow matrix of a Concurrent TDW Model with three TDWs W_1, W_2 and W_3. $\{D_1, \ldots, D_{10}\}$ is the data set of the Concurrent TDW Model. Parallelization can be applied here to reduce execution time. For each element in the data set of *cwm*, there is a thread being responsible for checking potential UCID caused by activities *using* this data element.
– After designing a new TDW, UCID check is conducted to find potential UCIDs before this TDW is put into the Concurrent TDW Management System for execution. Let's say *m, k, l* are the number of unfinished activities in the being considered pre-executed TDW, other pre-executed TDWs, running TDWs respectively, we have $n = m + k + l$. Because the other pre-executed TDWs have been checked in previous examinations, we can skip combinations of two activities in these TDWs to reduce the number of inspected combination to $n^2 - k^2$. If we just want to detect UCIDs caused by activities in the being considered TDW, we will

verify m x n activity combinations only. A parallel solution in this case is to create m threads. Each thread will be responsible for one activity in this TDW and will verify potential UCIDs on combinations created by this activity with the others in different TDWs.

- Because potential UCIDs just occur in activities that have shared data, we will verify activities having shared data only. Each data element will store identifications of unfinished activities using it. Therefore, the set of checked activities can be limited to unfinished activities having data relation in the Concurrent TDW Model. If the number of data elements is small, we can start from data elements of the being considered pre-executed TDW to pick out unfinished activities in the Concurrent TDW Model having data relations and use UCID patterns to find out potential errors.



	A_{11}	A_{21}	A_{12}	A_{13}	A_{31}	A_{22}	A_{32}	A_{14}	A_{23}	A_{33}
D1	W		R					R		
D2			R							
D3			W	R						
D4				W	R	R		R		
D5								W	R	
D6		W			R					
D7					W	R	W		R	R
D8									W	R
D9						W	R			
D10										W

Fig. 4. Data flow matrix example

6 Potential UCID Resolution

In general, if potential UCIDs happen, there may be some abnormalities in data flows or control flows of the concerned workflows. A review on the workflow design should be conducted to make sure that this situation is not made on purpose.

Our given solutions in which some of them will change the workflow structure are simply reference models. The final decision will depend on workflow designers to perform modifications that actually lead to a resolved model.

6.1 Potential Intra-UCID Resolution

Potential Intra-UCID may be caused by a mistake of workflow designers in designing parallel branches of a workflow. Therefore, our solution for Intra-UCID is to change the workflow structure by sequentializing or combining error-related activities. Two activities causing potential WW Intra-UCID are merged into one by place/transition fusion (Figure 5a). For RW Intra-UCID, sequentialization is applied to the related activities. One option is that read activity happens before write activity and the other is that write activity happens before read activity (Figure 5b). Resolution order will begin from WW Intra-UCID cases to RW Intra-UCID cases. With regard to potential UCIDs belonging to the same group, the priority is the happening order.

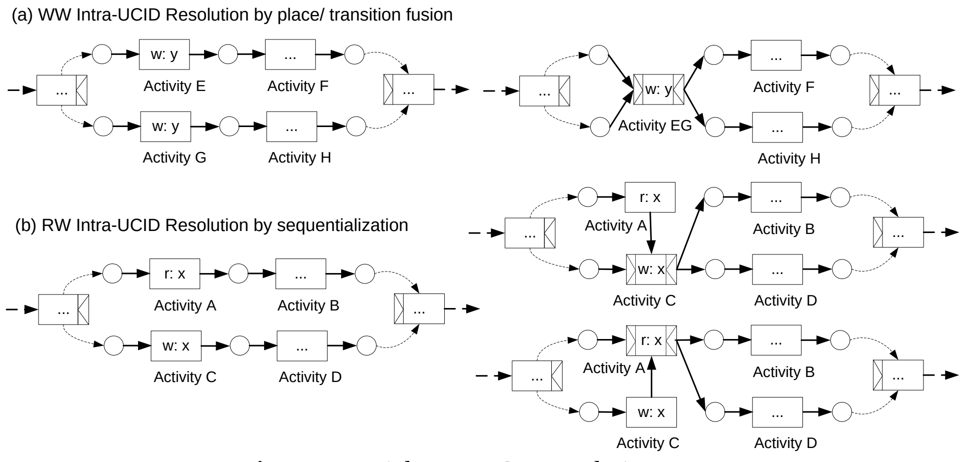


Fig. 5. Potential Intra-UCID resolution

6.2 Potential Inter-UCID Resolution

Resolving potential inter-UCID is more complex because workflows are designed for different purposes by different designers and a designer may know nothing about the work of the others. To resolve inter-UCID, the cooperation of different designers is necessary and the result will highly depend on the willingness of designers to communicate with each other.

A method which does not affect the workflow structures is to adjust the workflow schedule by modifying the workflow start time, maximum and minimum execution durations of activities in workflows so that inter-UCID patterns do not occur. Another solution is to change the workflow structure.

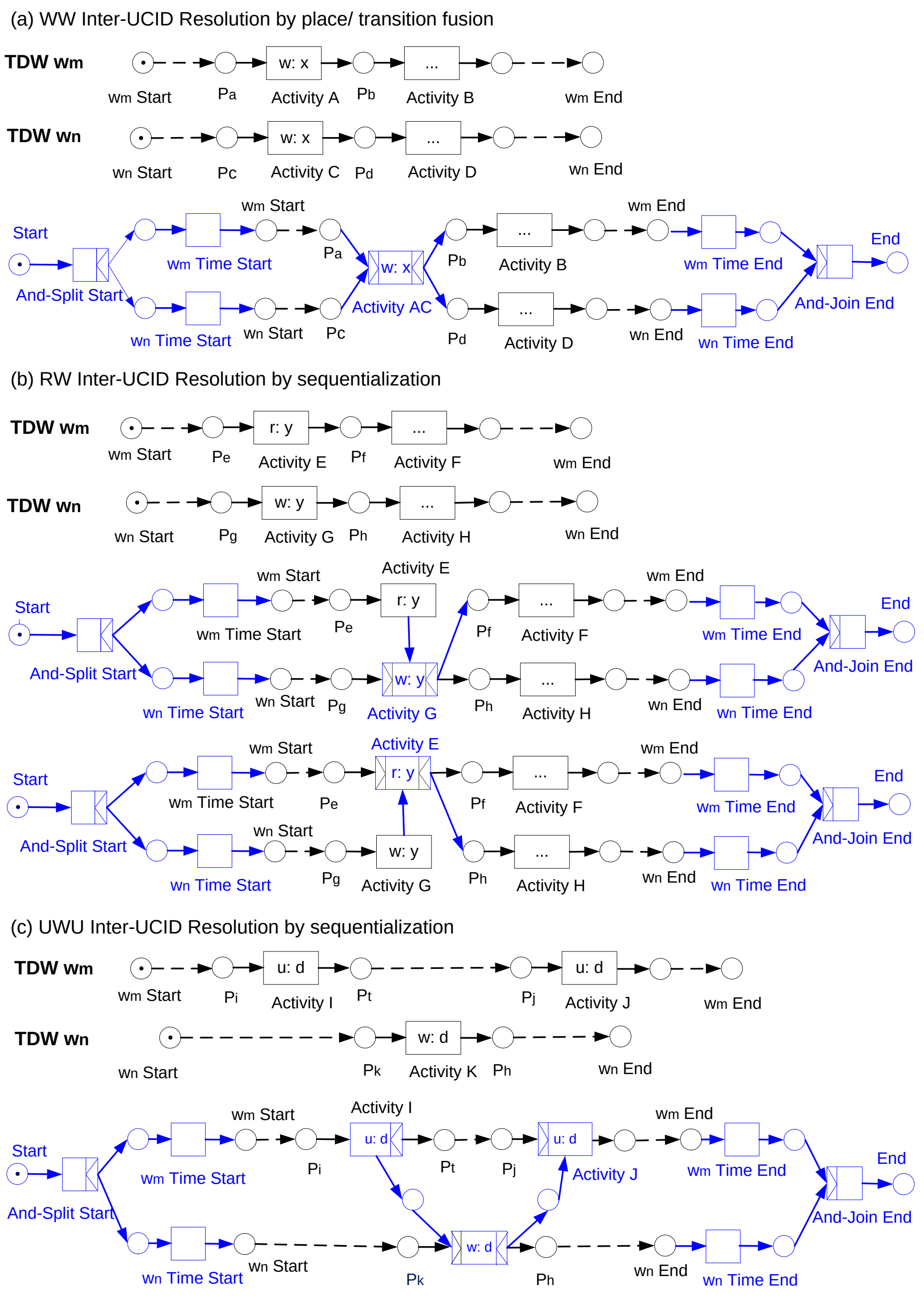


Fig. 6. Potential Inter-UCID resolution

First, we will combine related TDWs into one workflow. In order to preserve the structure of the original TDWs, in the new TDW, the Start place connects to an AND-Split transition and the End place is connected to an AND-join transition. Each merged

TDW corresponds to a subnet starting from the AND-split transition and ending at the AND-join transition. Because the merged TDWs are started at different times, we insert a Time Start transition between the Start place of each merged TDW and the AND-split transition, a Time End transition between the End place of each merged TDW and the END-join transition. Time activities are just null activities with some duration and they help to merge TDWs without modifying the workflow's schedule seriously. The AND-split transitions, AND-join transitions, Time Start transitions, Time End transition, places and arcs connecting the related workflows together represent the dependency relationships between different workflows which play an important role in the recovery process in the case of workflow failure. They will not be used to identify the total order of activities in detecting potential intra-UCID in the synthesis TDW. In the case of a running TDW, we can create a new TDW from the original workflow by removing its finished activities, and this new TDW will be combined with other TDWs in a normal way. Another simpler way is to combine the pre-executed TDWs only. After that, workflow designers can adjust the Estimated Active Interval of activities in the new TDW by modifying workflow start time, maximum and minimum execution duration of its activities so that UCID related activities happen after related activities of the running TDW.

Next, we will deal with activities causing potential Inter-UCID. The mechanism to handle potential WW/RW Intra-UCID is applied to WW/RW Inter-UCID cases (Figure 6a, 6b). Regarding UWU potential UCID, three activities related to this error are connected as shown in Figure 6c. If there are many potential Inter-UCIDs between the same two TDWs, the priority is Inter-UCID types (WW > RW > UWU) and occurring time of activities respectively.

As mentioned earlier, inter-UCID resolution is very complex, especially UWU inter-UCID. Currently, our proposed solution is just a reference model which helps workflow managers to have a more comprehensive view of data related workflows. We will try to improve them in the future work.

7 Application

In this section, we present a project on building a change support environment for cooperative software development. UCID theory is used in this project to detect potential UCID between concurrent workflows.

Software systems must be changed under various circumstances during development and after delivery, such as for new requirement, error correction, performance improvement, etc. However, software change is not an easy task, especially in a cooperative environment where software artifacts with very complex dependency relationships are created based on the cooperation of many people. Besides, other problems such as concurrency of works, synchronization of changes on shared artifacts, etc. also make this task more difficult. Therefore, a change support environment is strongly demanded.

In order to help change workers to perform change activities safety and efficiently in a cooperative environment, we use workflow to represent activities needed to implement a change request. We define Change Support Workflow (CSW) as a sequence of activities required to implement a change. Activities in CSW are responsible for creating new software artifacts or modifying exiting ones. This means that data elements of CSW are software artifacts which need to be read, modified or created in the change implementation process.

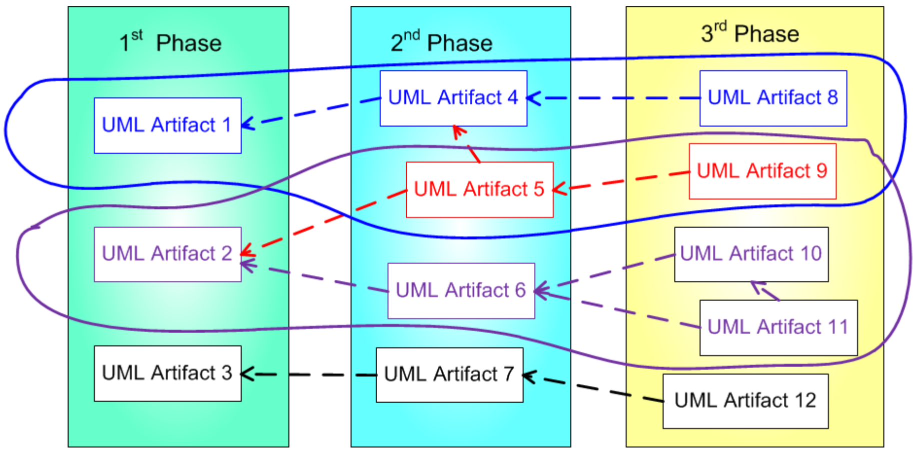


Fig. 7. Example of Relationships between UML Artifacts created during a software development process

In the first phase of the project, a method for automatically generating dependency relationships among UML elements was given [22]. Change impact analysis which identifies potential consequences of a change can be realized by tracing the generated dependency relationships. Result of this process will be used to generate CSW.

In large and cooperative system, there may be hundreds of CSWs executed at the same time to react to change requirements quickly. However, when there are many CSWs running on the same system, that UML artifacts are shared by different CSWs is unavoidable. If CSWs having shared artifacts are executed at the same time, inconsistencies among their data (UML artifacts) can happen. A version control system is used in our change support environment to deal with data loss; however this system does not help in this situation. Therefore, UCID theory is employed in this project to deal with this problem. Potential UCID can be detected automatically at build time to help workflow designers make timely adjustments to original workflows.

Our project supports constructing CSW based on the relationships between impacted UML model elements which are extracted from the result of impact analysis. CSW is modeled by TDW as follows. Each transition corresponds to an activity which creates or modifies at least one UML artifact. Total order of two transitions is identified by examining the dependency relationships between the artifacts modified by these transactions. Access role *write* is assigned to the artifacts which need to be modified or created; the artifacts for reference only are labeled with *read* access role. This draft of CSW will help workflow designers in developing the schedule of the change process.

The other steps in developing change schedule such as estimating activity resources and activity durations will be performed by workflow designers. From Activity Duration Estimates in the schedule, minimum and maximum execution durations of transitions in this CSW can be inferred. To reduce risks at runtime, UCID check on this CSW will be conducted. If some potential UCIDs are reported, data and control structure of this CSW should be adjusted in responding to suggested solutions of the change support system.

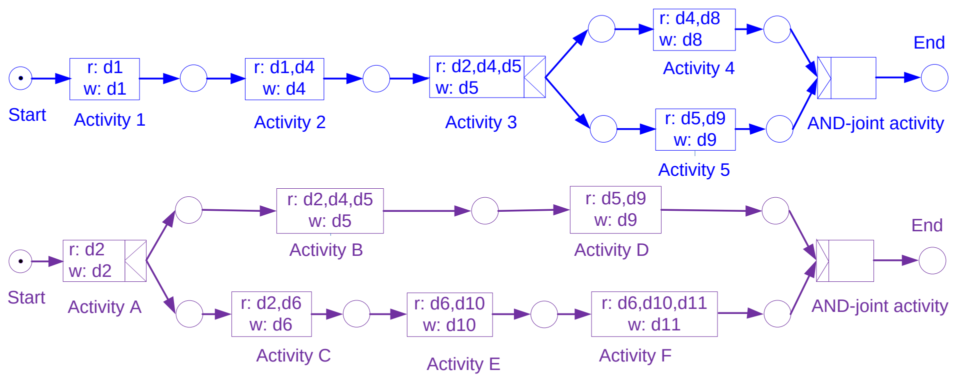


Fig. 8. Example of CSWs created based on the relationships between UML Artifacts

Table 1. Time aspect of activities in CSWs described in Figure 7

CSW ID	Start time P_w	Activity Name	Activity Duration Estimates (days)	Minimum and Maximum execution duration	Estimated Active Interval
W1	5	Activity 1	7.5 ± 0.5	{7,8}	[5,13]
		Activity 2	5.5 ± 0.5	{5,6}	[12,19]
		Activity 3	11 ± 1	{10,12}	[17,31]
		Activity 4	6 ± 1	{5,7}	[27,38]
		Activity 5	7 ± 1	{6,8}	[27,39]
		AND-joint	0	{0,0}	[33,39]
W2	15	Activity A	6 ± 1	{5,7}	[15,22]
		Activity B	5 ± 1	{4,6}	[20,28]
		Activity C	5 ± 1	{4,6}	[20,28]
		Activity D	10 ± 1	{9,11}	[24,39]
		Activity E	5.5 ± 0.5	{5,6}	[24,34]
		Activity F	6 ± 1	{5,7}	[29,41]
		AND-joint	0	{0,0}	[34,41]

Because CSW is constructed based on relationships between software artifacts, potential Intra-UCIDs seldom happen. Besides, if potential UCIDs are reported, the possibility of control flow errors is low too. In this case, workflow designers should review data flow and pay attention to shared data elements among concurrent CSWs. With reference to potential inter-UCID, Estimated Active Intervals of activities play a very important role; therefore a change on project schedule may help overcome this error.

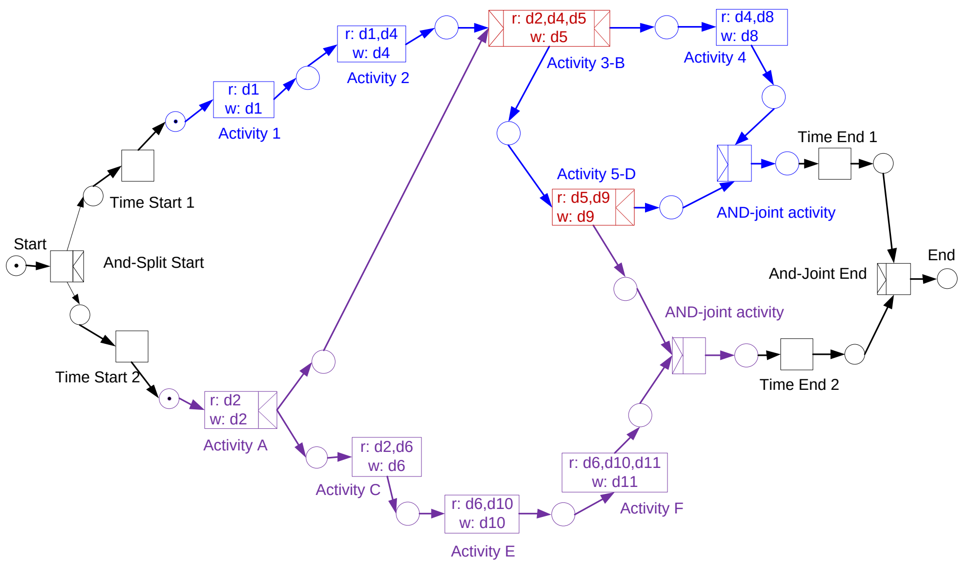


Fig. 9. Modified CSW with potential UCID corrected

Let’s have an example. Figure 7 describes an example of relationships between UML artifacts created in different phases of a software development process. If we change UML Artifact 1, we need to change UML Artifacts 4, 5, 8, 9 because of the relationships between them. Similarly, if we change UML Artifact 2, we need to change UML Artifacts 5, 6, 9, 10, 11. By tracing the relationships starting from UML Artifact 1 and UML Artifact 2, we can create two CSWs to respond to change requirements on UML Artifact 1 and UML Artifact 2 respectively (Figure 8). Based on the generated workflows, project manager can conduct other steps in project time management such as estimating activity resources, estimating activity durations, etc. Information about activity duration is used to detect potential UCIDs. In Table 1, the minimum and maximum execution durations of each activity in CSWs described in Figure 8 are calculated from the Activity Duration Estimate, quantitative assessment of the likely number of work periods that will be required to complete an activity [18], of the corresponding activity in the project time management. Based on these values and the start time of the corresponding workflow, we can calculate the Estimated Active Intervals according to the formulas given in Section 5.1. After using the Inter-UCID detection algorithms, the following potential Inter-UCIDs are reported: WW Inter-UCID between activity 3 and activity B on artifact 5, WW Inter-UCID between activity 5 and activity D on artifact 9, RW Inter-UCID between activity 3 and activity A on artifact 2, RW Inter-UCID between activity 5 and activity B on artifact 5. By applying the second Inter-UCID resolution method, modifying workflow structure, we get the synthesis CSW as described in Figure 9.

Because detecting potential UCIDs at build time is limited to workflows in which Estimated Active Intervals can be given before execution, solving this problem at runtime will be our next step. The model versioning system AMOR [21] offers some methods to resolve collaborative conflict in model versioning. Regarding this approach, all people who performed the changes are involved in eliminating the conflicts to obtain one consistent model version. We will consider applying this approach in our environment to increase the flexibility of the system.

8 Related Work

Workflow verification has attracted a lot of attention, especially control flow aspect. However, little research has been carried out on data verification in the workflow literature.

Reference [3] was one of the first studies to mention the importance of data-flow verification, and identified possible errors in the data-flow, like missing data, redundant data, conflict data, etc. Some general discussions on data flow modeling, specifications and verifications have been given, but without any detailed solution. The authors in [12] used data flow matrix and UML activity diagram to specify data flow. Based on this specification, an algorithm for detection of some data anomalies, such as missing data, redundant data, and potential data conflicts, was given [3]. In [11], a new workflow model, named Dual Workflow Nets, was defined to explicitly describe both control flow and data flow. A graph traversal approach was used in [10] to build an algorithm for detecting lost data, missing data and redundant data. More data flow errors were recognized and conceptualized as data flow anti-patterns and expressed in terms of temporal logic CTL* [5, 6]. By using temporal logic, available model checking techniques can be applied to discover these anti-patterns.

Nevertheless, all of these studies consider data flow errors in a single workflow only and no error removal method is given at all. In contrast to previous work, we address not only the interactions of concurrent activities inside a single workflow, but also the mutual influences between concurrent workflows, which are the sources of data flow errors. In [19], we focused on identifying UCID situations and defining a new workflow model as an extension of Petri Nets. Two algorithms for detecting intra-UCID and inter-UCID were also given in this work. However, there are still many unsolved problems in [19] and this paper is its refined and extended version. In this paper, TDW is defined as an extension of Workflow Nets (WF-Nets) instead of Petri Nets. Because the two algorithms in [19] had many common steps, if we use them separately, execution cost would be high. Therefore, these two algorithms are combined to reduce the cost and to form a more accurate and useful algorithm. Algorithm evaluation is also included in this version. Besides, some heuristics are provided to make the algorithm more flexible and effective. After that, some UCID resolution methods are proposed to help remove UCID errors. Finally, building a change support environment for cooperative software development is introduced as an application domain for our work.

Concerning the mutual influences of the concurrent workflows approach, the research closest to us is [7]. However [7] addressed the verification of workflow resource constraints, and in this work, by nature, handling the resource problem is simpler than the data problem. A Time Constraint Workflow Net was defined to model workflow. Then, they identified the problem of resource constraints in WFMS and proposed a pseudocode algorithm which checked the resource dependency between every two activities. Reference [4] used hybrid automata to model the influences between concurrent workflows, and adopted a model checking technique to detect resource conflict problems.

9 Conclusion and Future Work

In this paper, we have presented ***Unintentional Change in In-use Data*** *(UCID)* concept and classified types of UCID which can occur, between activities in a single workflow or in different concurrent workflows. We have also proposed a Time Data Workflow based on the WF-Nets with many attributes supporting UCID estimation. An algorithm which helps detect intra/inter-UCIDs in a Concurrent TDW Management System has been developed too. After that, algorithms evaluation and some solutions to resolve UCID problem are given. Finally, we have introduced a concrete project supporting software change development process in a cooperative software environment as an application using UCID theory to verify change processes at build time.

As future work, we will implement a prototype of Concurrent TDW Management System and evaluate the effectiveness of UCID detection algorithm by runtime analysis. Then, we will improve inter-UCID resolutions and refine the generated TDW after applying UCID resolution methods in the Concurrent TDW Management System. Detecting and correcting UCID at runtime are our next targets. We also plan to investigate formal verification methods to verify the correctness of our model and method. Finally, we will integrate our system into the open source WoPeD [17]. Another direction of our research is to extend the TDW and improve UCID detection algorithms to address errors in resource and access control constraints.

References

1. Lee, M., Han, D., Shim, J.: Set-based access conflicts analysis of concurrent workflow definition. In: Proceedings of Third International Symposium on Cooperative Database Systems and Applications, pp. 189--196. Beijing, China (2001)
2. Li, H., Yang, Y., and Chen, T. Y.: Resource constraints analysis of workflow specifications. J. Syst. Softw. 73, 2, pp. 271--285 (2004)
3. Sadiq, S., M. Orlowska, W. Sadiq and C. Foulger.: Data flow and validation in workflow modeling. In: Proceedings of 15th Australasian Database Conference. LI, H. pp. 207--214 (2004)

4. Kikuchi S., Tsuchiya S., Adachi M., and Katsuyama T.: Constraint Verification for Concurrent System Management Workflows Sharing Resources. In: Third International Conference on Autonomic and Autonomous Systems (2007)
5. Trˇcka N., van der Aalst W.M.P., and Sidorova N.: Analyzing Control-Flow and Data-Flow in Workfow Processes in a Unified Way. Technical Report CS 08/31, Eindhoven University of Technology (2008)
6. Trˇcka N., van der Aalst W.M.P., and Sidorova N.: Data-Flow Anti- Patterns: Discovering Data-Flow Errors in Workflows. In: 21st International Conference on Advanced Information Systems (CAiSE'09). LNCS, vol. 5565, pp. 425--439. Springer-Verlag Berlin Heidelberg (2009)
7. Zhong, J. and Song, B.: Verification of resource constraints for concurrent workflows. In: Proceedings of the Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 253--261 (2005)
8. Wil van der Aalst, Kees Max van Hee: Workflow Management: Models, Methods, and Systems. MIT press, Cambridge, MA (2004)
9. Zeng, Q., Wang, H. and Xu, D: Conflict detection and resolution for workflows constrained by resources and non-determined duration. Journal of Systems and Software 81(9), pp 1491--1504 (2008)
10. Sundari M.H., Sen A.K., and Bagchi A.: Detecting Data Flow Errors in Work-flows: A Systematic Graph Traversal Approach. In: 17th Workshop on Information Technology & Systems (WITS-2007). Montreal (2007)
11. Fan S., Dou W.C., and Chen J.: Dual Workflow Nets: Mixed Control/Data-Flow Representation for Workflow Modeling and Verification. In: Advances in Web and Network Technologies, and Information Management (APWeb/WAIM 2007Workshops), LNCS, vol. 4537, pp 433--444. Springer-Verlag, Berlin (2007)
12. Sun S.X., Zhao J.L., Nunamaker J.F., and Liu Sheng O.R.: Formulating the Data Flow Perspective for Business Process Management. Information Systems Research, 17(4), pp 374--391 (2006)
13. Heinlein, C.: Workflow and process synchronization with interaction expressions and graphs. In: Proceedings of the 17th International Conference on Data Engineering (ICDE '01), pp. 243–252 (2001)
14. Workflow Patterns, http://www.workflowpatterns.com
15. Russell N., van der Aalst W.M.P., and ter Hofstede A.H.M.: Designing a Workfow System Using Coloured Petri Nets. Transactions on Petri Nets and Other Models of Concurrency (ToPNoC) III, 5800, pp 1--24 (2009)
16. Awad, A., Decker, G. and Lohmann, N.: Diagnosing and Repairing Data Anomalies in Process Models. In: 5th International Workshop on Business Process Design. LNBIP, pp 1--24. Springer, Heidelberg (2009)
17. Workflow Petri Net Designer, http://193.196.7.195:8080/woped
18. PMBOK Guide Fourth Edition. Project Management Institute (2008)
19. Phan Thi Thanh Huyen and Koichiro Ochimizu: Detection of Unintentional Change on In-use Data for Concurrent Workflows. In: Proceedings of the 2010 International Conference on Software Engineering Research and Practice (SERP 10). Las Vegas, Nevada, USA (2010)
20. Elmasri, R. and Navathe, S. B.: Fundamentals of database systems, Benjamin-Cummings Publishing Co., Inc., Redwood City, CA (1989)
21. Adaptable Model Versioning, http://modelversioning.org/
22. Masayuki Kotani and Koichiro Ochimizu: Automatic Generation of Dependency Relationships between UML Elements for Change Impact Analysis. Journal of Information Processing Society of Japan, vol. 49, no.7, pp 2265—2291 (2008)

Taming the Shrew – Resolving Structural Heterogeneities with Hierarchical CPNs*

M. Wimmer[1], G. Kappel[1], A. Kusel[2],
W. Retschitzegger[2], J. Schoenboeck[1], and W. Schwinger[2]

[1] Vienna University of Technology, Austria
{lastname}@big.tuwien.ac.at
[2] Johannes Kepler University Linz, Austria
{firstname.lastname}@jku.at

Abstract. Model transformations play a key role in the vision of Model-Driven Engineering (MDE) whereby the overcoming of structural heterogeneities, being a result of applying different meta-modeling constructs for the same semantic concept, is a challenging, recurring problem, urgently demanding for reuse of transformations. In this respect, an approach is required which (i) abstracts from the concrete execution language allowing to focus on the resolution of structural heterogeneities, (ii) keeps the impedance mismatch between specification and execution low enabling seamless debuggability, and (iii) provides formal underpinnings enabling model checking. Therefore, we propose to specify model transformations by applying a set of abstract mapping operators (MOPs), each resolving a certain kind of structural heterogeneity. For specifying the operational semantics of the MOPs, we propose to use Transformation Nets (TNs), a DSL on top of Colored Petri Nets (CPNs), since it allows (i) to keep the impedance mismatch between specification and execution low and (ii) to analyze model transformations by evaluating behavioral properties of CPNs.

Key words: Model Transformation Reuse, Hierarchical CPNs, Structural Heterogeneities, Mapping

1 Introduction

MDE is a current trend in software engineering where models are used as first-class artifacts throughout the software lifecycle [2], which are then systematically transformed to concrete implementations. In this respect, model transformations play a vital role, representing *the* key mechanism for *vertical transformations* like the generation of code and *horizontal transformations* like model exchange between different modeling tools, to mention just a few. In the context of transformations between different metamodels and their corresponding

* This work has been funded by the Austrian Science Fund (FWF) under grant P21374-N13.

models, the overcoming of *structural heterogeneities*, being a result of applying different meta-modeling constructs for the same semantic concept [11, 13] is a challenging, recurring problem, urgently demanding for reuse of transformations.

In this respect, reusable transformations should abstract from a concrete transformation language, allowing to (preferably graphically) specify transformations in an explicit *specification view* without having to struggle with the intricacies of a certain transformation language. Secondly, for being able to debug and comprehend resulting specifications, the impedance mismatch between the specification view and the executable formalism needs to be minimized, demanding for a *debugging view* which retains the structure of the specification view, i.e., components used in the specification view should not get scattered in the debugging view. Finally, since debugging can only provide limited evidence of correctness by means of a set of test runs, the underlying executable formalism for the *execution view* should provide means to enable *model checking* [3].

We therefore propose to specify horizontal model transformations by means of abstract mappings representing a set of reusable transformation components, called mapping operators (MOPs), to resolve recurring structural heterogeneities. These MOPs operate on different levels of granularity, i.e., we provide a set of *kernel MOPs* representing the basic functionality needed for resolving structural heterogeneities and a set of *composite MOPs* encapsulating several kernel MOPs, thus enhancing scalability of our approach. In order to specify the operational semantics of the MOPs, we propose to use TNs [23], a DSL on top of CPNs [9], since TNs allow to keep the impedance mismatch between specification view and debugging view low by encapsulating the transformation logic of a single MOP together with the metamodels and the models. Thereby debuggability and comprehensibility are fostered, i.e., the ability of finding and reducing the number of bugs. Moreover, the underlying CPNs allow to specify reusable components in the form of modules, which can be nested in a hierarchical way, allowing to accordingly represent composite MOPs. Therefore the main contribution of this paper is to enable reuse also on the execution level, i.e., the Petri Net layer. Finally, the formal underpinnings of CPNs allow the application of generally accepted behavioral properties to analyze the transformation specification. The whole framework is called TROPIC – TRansformations On Petri nets In Color.

The remainder of this paper is structured as follows. Section 2 introduces a motivating example, Section 3 concentrates on the specification of a transformation and Section 4 deals with the debugging thereof. The subsequent Section 5 shows how TNs are represented in standard CPNs and how behavioral properties are exploited to analyze the transformation specification. Lessons learned are discussed in Section 6 and related work is surveyed in Section 7. Finally, Section 8 concludes the paper with an outlook on future work.

2 Motivating Example

Structural heterogeneities between different metamodels occur due to the fact that semantically equivalent concepts can be expressed by different metamodeling concepts, e.g., explicitly by classes or only by attributes. Fig. 1 shows an

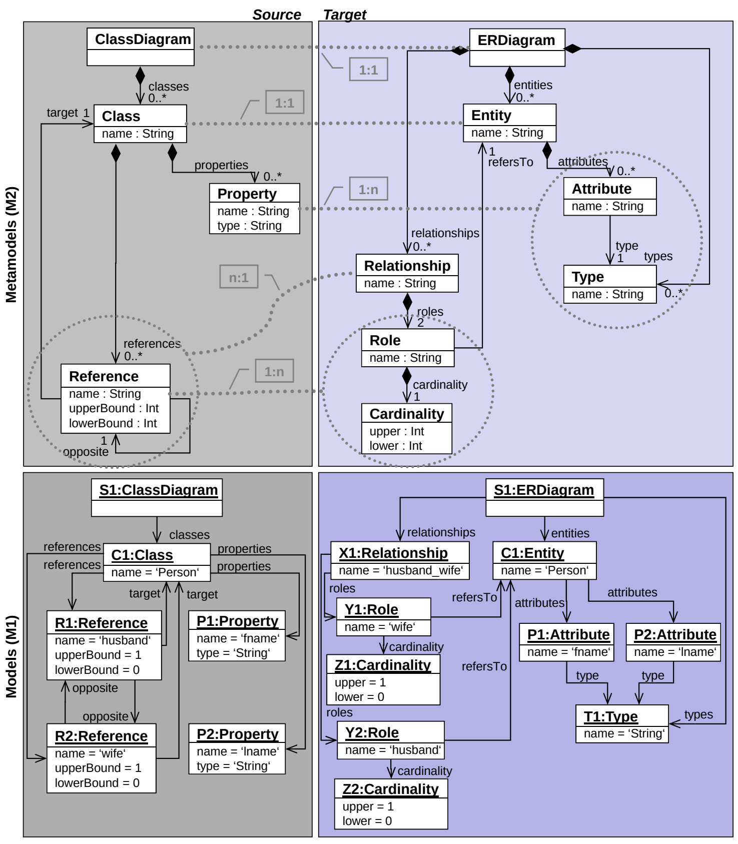


Fig. 1. Metamodels and Models of the Running Example

example used throughout the rest of the paper which exhibits common structural heterogeneities between metamodels, applying different modeling constructs to represent relationships as can be found e.g., in Ecore[3] or in Entity-Relationship Models. The `ClassDiagram` shown on the left side of Fig. 1, only provides unidirectional references, thus bidirectionality needs to be modeled by a pair of opposite references. In contrast to that, the `ERDiagram` explicitly represents bidirectionality, allowing to express relationships in more detail, e.g., using roles.

In the following, the main correspondences between the `ClassDiagram` and the `ERDiagram` are shortly described. On the level of classes, three main correspondences can be recognized, namely *1:1 correspondences*, *1:n correspondences* and *n:1 correspondences*, which are also indicated by dotted lines in Fig. 1. 1:1 correspondences can be found (i) between the root classes `ClassDiagram` and `ERDiagram` as well as (ii) between `Class` and `Entity`. Regarding 1:n correspondences, again two cases can be detected, namely (i) between the class `Property` and the classes `Attribute` and `Type` and (ii) between the class `Reference` and the classes `Role` and `Cardinality`. Although these are two occurrences of a 1:n

[3] http://www.eclipse.org/modeling/emf/

correspondence, there is a slight difference between them, since in the first case only for distinct values of the attribute `Property.type`, an instance of the class `Type` should be generated. Finally, there is one occurrence of a n:1 correspondence, namely between the class `Reference` and the class `Relationship`. It is classified as n:1 correspondence, since for *every pair* of `References`, that are opposite to each other, a corresponding `Relationship` has to be established. Considering attributes, only 1:1 correspondences occur, e.g., between `Class.name` and `Entity.name`, whereas regarding references, 1:1 correspondences and 0:1 correspondences can be detected. Concerning the first category, one example thereof arises between `ClassDiagram.classes` and `ERDiagram.entities`. Regarding the latter category, e.g., the relationship `ERDiagram.types` exists in the target without any corresponding counterpart in the source.

3 Specification View

As mentioned before, the actual specification of a transformation problem should abstract from a concrete transformation language allowing the transformation designer to focus on the resolution of structural heterogeneities without having to struggle with the intricacies of a certain transformation language. Therefore we propose to specify model transformations by means of abstract mappings being a declarative description of the transformation, as known from the area of data engineering [1]. For this we provide a library of composite MOPs [21]. Thereby we identified typical mapping situations being 1:1 copying, 1:n partitioning, n:1 merging, and 0:1 generating of objects, for which different MOPs are provided. In this respect, reuse is leveraged as the proposed MOPs are generic in the sense that they abstract from concrete metamodel types since they are typed by the core concepts of current meta-modeling languages like Ecore or MOF (i.e., class, attributes, references). To further structure the mapping process we propose to specify mappings in two steps.

In a first step, *composite MOPs*, describing mappings between classes are applied, providing an abstract *blackbox-view* (cf. Fig. 2). Every composite MOP consists of so-called *kernel MOPs*, thus the composite behavior is realized by a set of basic building blocks. These kernel MOPs are responsible for resolving structural heterogeneities and therefore they have to be able to map classes, attributes, and references in all possible combinations and mapping cardinalities. In this respect, MOPs are provided for copying exactly one object, value, or link from source to target, respectively (denoted as $\mathrm{C(lass)_2C(lass)}$, $\mathrm{A(ttribute)_2A(ttribute)}$, and $\mathrm{R(eference)_2R(eference)}$). Moreover, MOPs are needed for merging objects, values, and links (denoted as $\mathrm{C_2^nC}$, $\mathrm{A_2^nA}$, and $\mathrm{R_2^nR}$) resolving the structural heterogeneity that concepts in the source metamodel are more fine-grained than in the target metamodel. Finally, MOPs are needed for generating a target element without an obvious source element (denoted as $\mathrm{0_2C}$, $\mathrm{0_2A}$, and $\mathrm{0_2R}$) to resolve heterogeneities resulting from expressing the same modeling concept with different meta-modeling concepts – a situation which often occurs in metamod-

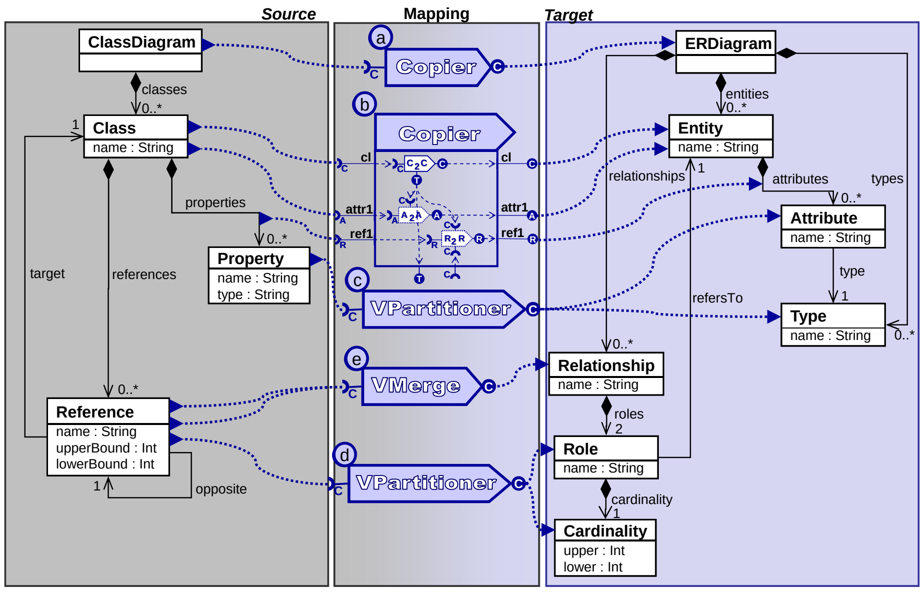


Fig. 2. Solution of the Running Example

eling practice.[4] In a second step, the composite MOPs, which solely describe a mapping between classes at first, have to be refined to also map attributes and references in the so-called *whitebox-view* by the usage of kernel MOPs (cf. expanded Copier (b) in Fig. 2). Furthermore, kernel MOPs can be used to assemble new, user-defined composite MOPs.

As a concrete syntax for MOPs we are using a subset of the UML 2 component diagram concepts enabling the specification of model transformations in a plug & play manner. With this formalism, every MOP is defined as a dedicated component, representing a modular part of the transformation specification which encapsulates an arbitrary complex structure and behavior, providing well-defined interfaces. Every MOP has input ports with required interfaces (left side of the component) as well as output ports with provided interfaces (right side of the component), typed to classes (C), attributes (A), and relationships (R) (cf. `Copier` (b) in Fig. 2). Since there are dependencies between MOPs, e.g., a value can only be set after the owning object has been created, MOPs dealing with the transformations of classes additionally offer a trace port (T) at the bottom providing *context information*, indicating which target object has been produced from which source object(s). This port can be used by dependent MOPs to access context information via required context ports (T). In case of MOPs dealing with the mapping of attributes the corresponding interface is shown via one port on top, or in case of MOPs dealing with the mapping of ref-

[4] Please note, that although composite MOPs for 1:n partitioning are provided, no additional kernel MOps are needed, since such situations can be simulated by n *x* 1:1 MOps.

Table 1. Overview of Composite MOPs used in the Example

Correspondence	MOP	Description	Composition of Kernel MOPs (EBNF)
1:1 - copying	Copier	creates exactly one target object per source object	Copier: $C_2C \{ A_2A \mid A^n{}_2A \mid 0_2A \mid R_2R \mid R^n{}_2R \mid 0_2R \}$
1:n - partitioning	VerticalPartitioner	splits one source object into several target objects	VerticalPartitioner: Copier { ObjectGenerator $\mid$ Copier }
n:1 - merging	VerticalMerger	merges several source objects to one target object	VerticalMerger: $C^n{}_2C \{ A_2A \mid A^n{}_2A \mid 0_2A \mid R_2R \mid R^n{}_2R \mid 0_2R \}$
0:1 - generating	ObjectGenerator	generates a new target object without corresponding source object	ObjectGenerator: $0_2C \{ A_2A \mid A^n{}_2A \mid 0_2A \mid R_2R \mid R^n{}_2R \mid 0_2R \}$

erences via two ports, whereby the top port depicts the required source context and the bottom port the required target context (cf. `Copier` (b) in Fig. 2).

For solving the running example, several composite MOPs have been applied as can be seen in Fig. 2. Table 1 presents an overview of the used composite MOPs to solve the example as well as their composition of kernel MOPs. For a detailed classification and description of all available kernel as well as composite MOPs we refer to [21]. To resolve the 1:1 correspondences between `ClassDiagram` and `ERDiagram` as well as between `Class` and `Entity` in our example we applied two `Copiers` since for every source object a corresponding target object should be generated (cf. MOPs (a) and (b) in Fig. 2)). The whitebox-view of the `Copier` (b) thereby shows the mapping of class `Class` to class `Entity` using a C_2C MOP. Moreover, the attribute `Class.name` is mapped to the attribute `Entity.name` by using an A_2A MOP. Finally, the reference `Class.properties` is mapped to the reference `Entity.attribute` using a R_2R MOP. To split the attributes of the class `Reference` to the target classes `Role` and `Cardinality` a `VerticalPartitioner` is applied (cf. MOP (d) in Fig. 2). Besides this default behavior, aggregation functionality is sometimes needed as is the case when splitting the `Property` concept into the `Attribute` and `Type` concepts, since a `Type` should only be instantiated for distinct `Property.type` values (cf. MOP (c) in Fig. 2). To merge two `Reference` objects to a single `Relationship` object a `VerticalMerger` is applied (cf. MOP (e) in Fig. 2).

4 Debugging View

In the previous section we showed how structural heterogeneities can be resolved by applying MOPs resulting in a declarative mapping specification. In order to execute this specification it has to be translated into an executable formalism, i.e., every MOP has to be assigned an operational semantics. Thereby, the impedance mismatch between the declarative specification and the actual operational semantics should be minimized in order to foster comprehensibility and debuggability. Since current transformation languages (cf. [4] for an overview) provide only a limited view on a model transformation problem, i.e., they do not visualize the actual metamodel and model being transformed, we proposed the TN formalism [23], being a DSL on top of CPNs [9]. The basic idea of TNs is to represent the transformation logic together with the metamodels and the models, whereby metamodel elements are represented by places, model elements by the according markings and the actual transformation logic by a system of transitions. Thus, an explicit runtime model is provided which can be used to observe the runtime behavior of a certain transformation. In the following we describe the core concepts of TNs as well as the adaptations introduced in comparison to standard CPNs to better suit the domain of model transformations.

Representation of Metamodels and Models. Since we rely on the core concepts of an object-oriented meta-metamodel the graph which represents the metamodel consists of classes, attributes, and references which are represented by according places in TNs. Therefore Fig. 3 depicts a place for the class `Class` as well as one place for the attribute `Class.name` and one place for the reference `Class.properties`. The graph which represents a conforming model consists of objects, data values and links which are represented by tokens in the according places. For every object that occurs in a model a one-colored *ObjectToken* is produced, which is put into a place that corresponds to the respective class in the source metamodel, e.g., the token `C1` in the `Class` place and the tokens `P1` and `P2` in the place `Property`, representing the objects of the source model depicted at the bottom of Fig. 1. The color is realized through a unique value that is derived from the object id (OID). For every value, two-colored *AttributeTokens* are produced whereby the upper color represents the object and the lower color the actual value, e.g., the `C1|Person` token represents the value "Person" of the attribute `Class.name` for the object `C1` in Fig. 3. Finally, for every link a two-colored *ReferenceToken* is produced. The outer color refers to the color of the token that corresponds to the owning object. The inner color is given by the color of the token that corresponds to the referenced target object, which is depicted by the corresponding tokens in the `Class.properties` place in Fig. 3.

Specification of Transformation Logic. The actual transformation logic is specified by means of a system of transitions and additional places, so-called *trace places* storing *context information* which reside in-between those places representing the original input and output metamodels. Transitions consist of so-called *query tokens* (LHS of the transition) representing the pre-condition of a certain transition, whereas *production tokens* (RHS of the transition) depict its postcondition. Thereby different query and production tokens for objects, values, links and context information are provided whose colors represent variables that are bound during execution, i.e, colors of query tokens are not the required colors for input tokens, instead they describe configurations that have to be fulfilled by input tokens. In the copying sceanrio the color of the production tokens depend on the color of the query tokens, e.g., the production token and the query token

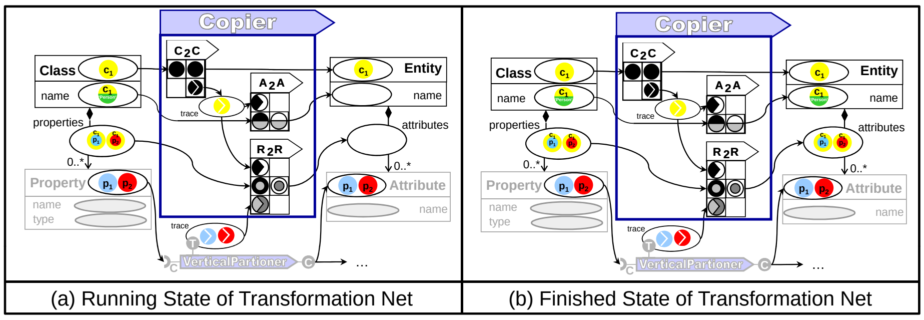


Fig. 3. Debugging View of Copier MOP

of the C_2C transition exhibit the same color and therefore the source and the target object tokens exhibit the same color (cf. Fig. 3). However, it is also possible to produce a token of a not yet existing color if a target object is needed which does not directly correspond to a source object, e.g., in case a C_2^nC MOP which merges several source objects to a single new target object. Furthermore, to represent trace ports of MOPs, *trace places* containing *context tokens* indicate which target object has been created from which source object(s). Thereby the color(s) of the slot (left side) indicate(s) the used source object(s) whereas the generated target object is represented by the color of the remaining slice (right side of token). Since object tokens are simply copied in case of the depicted C_2C transition source and target context tokens exhibit equal colors (cf. Fig. 3(a)). Only if context information is available in a trace place, dependent transitions, e.g., the A_2A and R_2R transitions, are able to fire. For this, they query the context tokens in order to add a value or a link to the target object acquired from the context token (cf. Fig. 3(b)). Please note that in case of creating a new target object, source and target color of the context tokens differ from each other. Thus, dependent transitions must be able to cope with differently colored context tokens and therefore the context query tokens of the dependent A_2A and R_2R transitions in Fig. 3 show different colors (which are only variables and are therefore also able to match for same colored tokens).

Adaptations of Standard CPNs. In contrast to standard CPNs, TNs exhibit a different default firing behavior, i.e., tokens are not consumed per default (therefore source tokens are preserved in their corresponding source places). This is since all possible token combinations must be taken into account. For example, if the R_2R transition would consume `Class` tokens and `Property` tokens from the trace places (cf. Fig. 3), the transition could fire only once although multiple `Properties` would be available, since there is a 1:n relationship between `Class` and `Property`. Moreover, if more than one transition accesses a certain place, consuming firing behavior would lead to erroneous race conditions.

Summarizing, TNs provide a formalism to specify the operational semantics of the provided MOPs. Thereby TNs reduce the impedance mismatch between the abstract declarative mapping specification and the actual operational semantics since there is a 1:1 correspondence between kernel MOPs and transitions. Additionally, all artifacts in a model transformation, i.e., metamodel, transformation logic and the involved models are represented in a homogenous view. Furthermore, as query and production tokens are only typed to the core concepts of object-oriented metamodels (class, attributes and references) the specified transformation logic can be reused between arbitrary metamodels (as intended by the MOPs). Due to the fact that every MOP is realized by an independent set of transitions every MOP can be debugged individually, thus enabling a component-oriented debugging approach.

5 Execution View

Since TNs represent a DSL on top of CPNs they can be fully translated into existing CPN concepts to make use of efficient execution engines and their properties to analyze model transformations [20]. The actual translation is transparent to the user since a TN is automatically converted to an according CPN using the ASAP platform [19]. The ASAP platform provides an EMF-based implementation of the PNML standard[5] for CPNs. The CPN model can then be used to check the syntax of the corresponding TN, to simulate the TN and to calculate behavioral properties for the specified model transformations. Since every MOP is realized by an independent set of TN transitions we provide pre-defined hierarchical CPNs for kernel and composite MOPs, detailed in the following. Furthermore, the application of behavioral properties for analyzing model transformations is shown.

5.1 Representation of Kernel MOPs

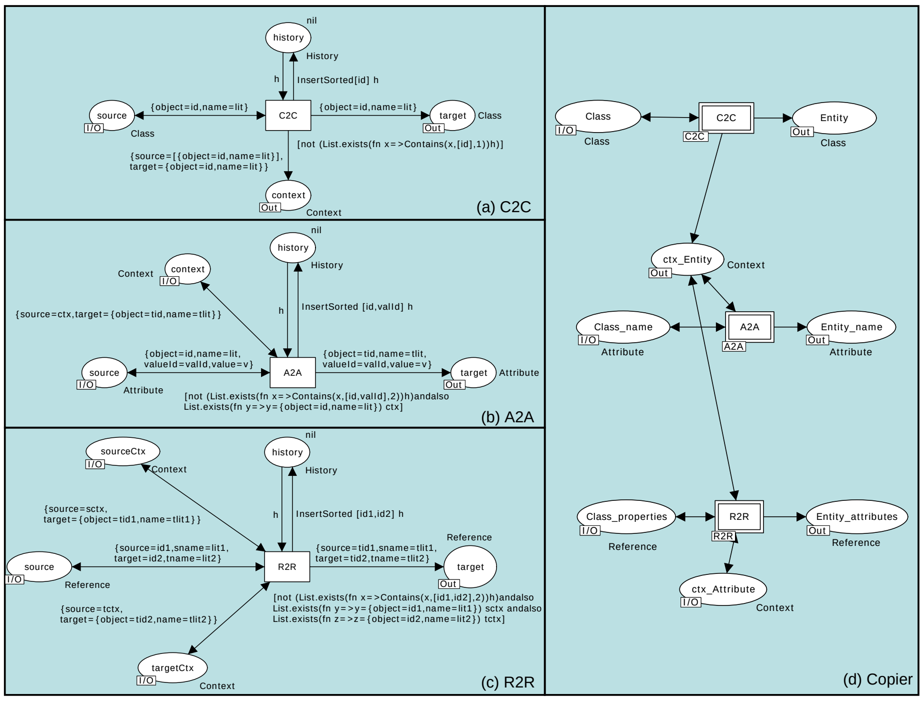
Kernel MOPs and their respective operational semantics in TNs can be represented by means of *modules* or so-called *substitution transitions* in hierarchical CPNs whereby the *ports* of the substitution transitions are only typed by classes, attributes, and references. The ports are then bound to the corresponding *socket places* being the places derived from the source and target metamodel. In the following we show how to realize the non-consuming behavior in CPNs as well as the translation of kernel MOPs to hierarchical CPNs.

Adaptations of Standard CPNs. To realize the non-consuming firing behavior, a so-called *history place* is introduced for every transition. It stores all token combinations that have already been fired by this transition in a sorted list in order not to blow up the state space, i.e., there is no difference if token `P1` or token `P2` has been transformed first in our scenario. The history place is connected to the corresponding transition whereby a guard condition prevents the transition from firing a certain token combination twice. Moreover, the standard arcs are replaced by so-called *test arcs*, which do not consume tokens from the connected input places. For further details on the translation of TNs to CPNs we refer the interested reader to [23].

MOPs mapping Classes. In case of kernel MOPs dealing with the mapping of classes, e.g., a C_2C MOP as depicted in Fig. 4(a), the in- and outports have to be typed to the colorset `Class` (`colset Class = record object : INT * name : STRING`). As a C_2C MOP simply copy tokens, the same arc inscription can be found on the in- and outgoing arcs (represented by the same colors of query and production token in TNs). Furthermore, kernel MOPs mapping classes provide context information stored in the *context* port[6]. The colorset `Context` thereby defines a record consisting of a list of classes (since more than one class can be used to enable the transition in case of a C_2^nC) and a target class (`colset Context = record source:SourceContext * target : Class; colset SourceContext = list Class;`).

[5] http://www.pnml.org/

[6] Note that ports providing context information in MOPs and CPNs are used to enable dependent MOPs or transitions, i.e, they provide required tokens to enable a transition, whereas the history concepts solely hinders multiple firings of transition in CPNs



(a) C2C
(b) A2A
(c) R2R
(d) Copier
history
History
nil
InsertSorted[id] h
source
target
context
Class
Context
{object=id,name=lit}
[not (List.exists(fn x=>Contains(x,[id],1))h)]
{source=[{object=id,name=lit}], target={object=id,name=lit}}
Attribute
{source=ctx,target={object=tid,name=tlit}}
InsertSorted [id,valId] h
{object=id,name=lit, valueId=valId,value=v}
{object=tid,name=tlit, valueId=valId,value=v}
[not (List.exists(fn x=>Contains(x,[id,valId],2))h)andalso List.exists(fn y=>y={object=id,name=lit}) ctx]
sourceCtx
targetCtx
Reference
InsertSorted [id1,id2] h
{source=sctx, target={object=tid1,name=tlit1}}
{source=id1,sname=lit1, target=id2,tname=lit2}
{source=tid1,sname=tlit1, target=tid2,tname=tlit2}
{source=tctx, target={object=tid2,name=tlit2}}
[not (List.exists(fn x=>Contains(x,[id1,id2],2))h)andalso List.exists(fn y=>y={object=id1,name=lit1}) sctx andalso List.exists(fn z=>z={object=id2,name=tlit2}) tctx]
Entity
ctx_Entity
Class_name
Entity_name
Class_properties
Entity_attributes
ctx_Attribute
C2C
A2A
R2R
I/O
Out

Fig. 4. Realization of MOPs by Hierarchical CPNs

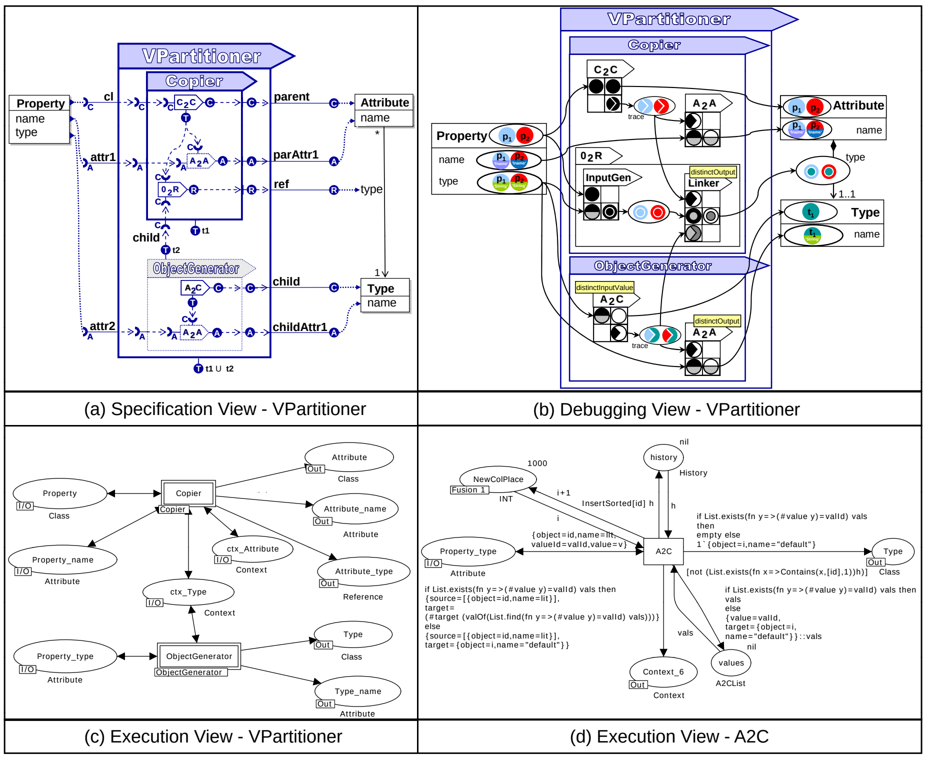
MOPs mapping Attributes or References. In case of kernel MOPs dealing with the mapping of attributes or references, e.g., an A_2A MOP or R_2R MOP as depicted in Fig. 4(b) and (c), the ports have to be typed to the colorset `Attribute` and `Reference` respectively (`colset Attribute = record object:INT * name:STRING * valueId : INT * value : STRING; colset Reference = record source:INT * sname: STRING * target:INT * tname: STRING;`). Since attributes and references should only be transformed if the owning object of an attribute or the source and target objects of a reference have already been transformed, the guard condition of the transition not only prevents the multiple firing but additionally checks if the context place already contains the necessary context information. If the condition is fulfilled, an attribute or reference token is produced whereby the new owning object `tid` is acquired from the context tokens (cf. arc inscription at the in- and outgoing arcs from context places Fig. 4(b) and (c)). These hierarchical CPNs can then be assembled to more coarse-grained hierarchical CPNs to represent, e.g., a `Copier` as shown in Fig. 4(d). In the following, composite MOPs are elaborated in more detail.

5.2 Representation of Composite MOPs

Specification View. In Section 3 we introduced coarse-grained composite MOPs which encapsulate several kernel MOPs, e.g., a `Copier` consists of exactly one C_2C, and several MOPS for mapping attributes or references. As can be seen in

Table 1, composite MOPs can not only consist of kernel MOPs but might encompass composite ones themselves, e.g., `VerticalPartitioner` which consists of a `Copier` and an `ObjectGenerator` (cf. Fig. 5(a)). In our running example this MOP was used to split the source concept `Property` into the concepts `Attribute` (achieved by the contained `Copier`) and `Type`, whereby a `Type` should only be instantiated for distinct `Property.type` values overcoming the heterogeneity that a concept is expressed as an attribute in the source metamodel and as a class in the target metamodel (achieved by the contained `ObjectGenerator`).

Debugging View. The relation between composite and the kernel MOPs can be seen in the debugging view (cf. Fig. 5(b)). First, the C_2C transition of the copier streams the corresponding object tokens, thus creating an `Attribute` for every `Property`. The thereby generated context information enables the A_2A transition in order to set the `Attribute.name` values. Second, the A_2C transition generates a `Type` object token for distinct `Property.type` values, which is indicated by the `distinctInputValue` annotation on the transition meaning that only context information in the according trace place is generated but no new target token in case that a value occurs several times. Therefore, the trace place of the `ObjectGenerator` composite MOP contains two `Property.type` tokens which both have been mapped to the same `Type` object (depicted by the equal target color of the context tokens) since both source tokens have the same



(a) Specification View - VPartitioner (b) Debugging View - VPartitioner

(c) Execution View - VPartitioner (d) Execution View - A2C

Fig. 5. Different Views of VerticalPartitioner

value ``String''. In order not to produce too many attribute tokens the dependent A_2A MOP has to match only for distinct target colors of context tokens resulting in distinct output values (indicated by the according annotation in (cf. Fig. 5(b)). Finally, the generated `Attribute` and `Type` objects have to be accordingly linked by the reference `Attribute.type`. Since there is no according source reference available we have to generate this reference by applying a 0_2R MOP. Nevertheless, the transformation designer has to define during specification how the generated target objects are related to each other in the source model. In our example the intention is to generate a reference for every `Attribute` object having set an according `Attribute.type` value. In order to get this input the `InputGen` transition collects the tokens and thereby generates (self) references. These references can then be processed by the `Linker` component which finally produces the according `Attribute.type` references.

Execution View. In order to represent the different levels of granularity, the corresponding hierarchical CPN again consists of several nested ones, thus leading to multi-level hierarchical CPNs. As shown in Fig. 5(c) the `VerticalPartitioner` consists of two substitution transitions, being a `Copier` and an `ObjectGenerator`. As already shown in the debugging view (cf. Fig. 5(b)), the main part of an `ObjectGenerator` is an A_2C kernel MOP. Since only for distinct values a new target object should be generated, an additional `values` place containing a list of records (`colset A2CList = list A2C; colset A2C = record value:INT * target:Class`), expressing which values have already been converted to a certain class token, is introduced (cf. Fig. 5(d)). The conditions on the outgoing arcs to the target place and to the `values` places ensure that a token is only created if the value has not been contained in the `values` list before. In contrast to that, context information is produced for any firing of the transition whereby the source object is connected to an already existing class target token if the values list contains an according entry, i.e., if a `Type` has already been created for a certain `Property.type` value. To represent the fact that a `Type` object has no according counterpart in the source model, we generate a new object id which is the task of the (fusion) place `NewColPlace` and the according arc inscriptions represented by a newly colored object production token in TNs.

5.3 Behavioral Properties to Analyze Mappings

Although the operational semantics of MOPs is predefined, configuration errors might occur when applying the MOPs in the specification phase leading to an erroneous interplay between MOPs. In the following we show how typical errors can be detected by means of behavioral properties of the underlying CPNs [20].

Model Comparison using Boundedness Properties. Typically, the first step in analyzing the correctness of a transformation is to compare the generated target model to an expected target model. To identify wrong or missing target elements in terms of tokens automatically, *Boundedness* properties can be applied. An example thereof could be the A_2C MOP in the above example which creates target tokens for distinct values only. Therefore dependent transitions need to generate a distinct output as well, e.g., to set the `Type.name` value only once. If this is not specified by the user accordingly, too many `Type.name` tokens are generated which can be detected by comparing the Boundedness properties of the according place of the generated model to the expected target model.

Checking Interplay of MOPs using Liveness Properties. Another source of error during the refinement of composite MOPs by kernel MOPs is the mapping of dependent attributes and references. In case that MOPs dealing with attributes and references are connected to wrong source or target context ports the corresponding transition is not able to fire which can be detected by *Liveness Properties* such as *Dead Transition Instances* or *L0-Liveness*.

Termination and Confluence Analysis using Dead and Home Markings. A transformation specification must always terminate, thus the state space has to contain at least one *Dead Marking*, which is typically ensured by the history concept. Moreover, it has to be ensured that a dead marking is always reachable, meaning that a transformation specification is confluent, which can be checked by the *Home Marking*. Furthermore, it is possible to check if a certain marking, i.e., the target marking derived from the expected target model, is reachable. If this marking is equal to the Dead and Home Marking it is ensured that the specified mapping always generates the expected target model.

6 Lessons Learned

This section presents lessons learned and discusses key features of our approach.

Kernel MOPs Enable Extensibility. Kernel MOPs form the basis for overcoming structural heterogeneities and thereby have to exhibit a well-defined operational semantics. Since composite MOPs are solely based on kernel MOPs, the composite operational semantics results from the operational semantics of the kernel MOPs. Therefore, the library of composite MOPs can be easily extended on basis of the kernel MOPs without the need of adapting the compilation to TNs and CPNs, respectively.

CPNs Allow for Parallel Execution. As CPNs exhibit an inherent concurrency, parallel execution of transformation logic is possible thereby increasing the efficiency of a transformation execution. In particular, mappings between classes are independent from each other and therefore the transformation of objects can be fully parallelized. The same is true for depending attributes and references which can also be transformed in parallel after the owning objects have been created and thus the needed context tokens are available.

Visual Formalism Eases Debugging and Understandability. TNs provide a visual formalism for defining model transformations which is especially useful for debugging purposes, since the actual execution of a certain model transformation can be simulated. In this respect, the transformation of model elements can be directly followed by observing the flow of tokens and therefore undesired results can be detected easily.

History Ensures Termination. As mentioned above, TNs introduce a specific firing behavior in that transitions do not consume the source tokens satisfying the precondition but hold them in a history. Thus, a transition can only fire once for a specific combination of input tokens prohibiting infinite loops, even for test arcs or cycles in the net. Only if a transition occurs in a cycle and if it produces new objects every time it fires, the history concept can not ensure termination. Such cycles, however, can be detected at design time and are automatically prevented for TNs. In contrast to model transformation languages based on graph grammars, where termination is undecidable in general [14], TNs ensure termination already at design time.

State Space Explosion Limits Model Size. A known problem of model checking and thus also of behavioral properties of Petri Nets is that the state space might become very large. Currently, the full occurrence graph is constructed to calculate properties leading to memory and performance problems for large source models and transformation specifications. Often a marking M has n concurrently enabled, different binding elements leading all to the same marking. Nevertheless, the enabled markings can be sorted in $n!$ ways, resulting in an explosion of the state space. As model transformations typically do not care about the order how certain elements are bound, the number of bindings can be reduced to 2^n bindings, thus enhancing scalability of our approach.

7 Related Work

In the following, related work is summarized according to the proposed views.

Specification View. In the area of model engineering only the ATLAS Model Weaver (AMW) [7] provides a dedicated mapping tool allowing the definition of model transformations independent of a concrete transformation language. By extending the weaving metamodel, one can define the abstract syntax of new weaving operators which roughly correspond to our MOPs. The semantics of weaving operators is determined by a higher-order transformation [16], taking a model transformation as input and generating another model transformation as output. Compared to our approach, the weaving models are compiled into low-level ATL [10] transformation code which is in fact a mixture of declarative and imperative language constructs. Thus, this solution exhibits an impedance mismatch, hindering the understanding and debugging of the resulting code.

Debugging View. Concerning model transformations in general, there is little debugging support available. Most often only low-level information available through the execution engine is provided, but traceability according to the higher-level correspondence specifications is missing. For example, in the Fujaba environment, a plugin called MoTE [18] compiles TGG rules [12] into Fujaba story diagrams that are implemented in Java, which obstructs a direct debugging on the level of TGG rules. In [8], the generated source code is annotated accordingly to allow the visualization of debugging information in the generated story diagrams, but not on the TGG level. Concerning the understandability of model transformations in terms of a visual representation and a possibility for a graphical simulation, only graph transformation approaches like Fujaba allow for a similar functionality. However, these approaches neither provide an integrated view on all transformation artifacts nor do they provide an integrated view on the whole transformation process in terms of the past state, i.e., which rules fired already, the current state, and the prospective future state, i.e., which rules are now enabled to fire. Therefore, these approaches provide snapshots of the current transformation state, only.

Execution View. Current transformation languages provide only limited support to analyze transformation specifications as summarized in the following. In the area of graph transformations some work has been conducted that uses Petri Nets to check properties of graph production rules. Thereby, the approach proposed in [17] translates individual graph rules into a Place/Transition Net and checks for its termination. Another approach is described in [6], where the operational semantics of a visual language in the domain of production systems is described with graph transformations. The production system models as well as the graph transformations are transformed into Petri Nets in order to make use of analysis techniques for checking properties of the production system models. Finally, a recent work by de Lara and Guerra [5] proposes to translate QVT-Relations into CPNs - on the one hand to provide a formal semantics for QVT Relations and on the other hand to analyze QVT Relations specifications - pursuing similar ideas as followed in our previous work [22]. Nevertheless, these approaches are using Petri Nets only as a back-end for analyzing properties of transformations, whereas we are using a DSL on top of CPNs as a front-end for model transformations, thereby fostering debuggability.

8 Future Work

Currently, only the most important concepts of modeling languages, i.e., classes, attributes and relationships have been considered by our MOPs. It would be desirable, however, to extend our MOP library to be able to deal also with concepts such as inheritance or complex mathematical operations. Furthermore, only a basic prototype of the proposed debugging view is available. We therefore focus on improving our prototype, e.g., by accordingly visualizing the findings of the formal properties. Concerning verification support, we focused on small mapping scenarios up to now only, not least due to the state space explosion problem. Nevertheless the ASAP platforms provides the possibility to specify own algorithms to explore the state space which could additionally be adopted to the domain of model transformation to enable verification support for larger scenarios. To further support the transformation designer in complementing the mapping in the whitebox-view, auto-completion strategies should be incorporated. In this respect, we will investigate on matching strategies [15] which may be applied to automatically derive attribute and relationship mappings.

References

1. P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *Proc. of SIGMOD'07*, pages 1–12. ACM, 2007.
2. J. Bézivin. On the Unification Power of Models. *Journal on Software and Systems Modeling*, 4(2):31, 2005.
3. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* The MIT Press, 1999.

4. K. Czarnecki and S. Helsen. Feature-based Survey of Model Transformation Approaches. *IBM Systems Journal*, 45(3):621–645, 2006.
5. J. de Lara and E. Guerra. Formal Support for QVT-Relations with Coloured Petri Nets. In *Proc. of MoDELS'09*, 2009.
6. J. de Lara and H. Vangheluwe. Automating the Transformation-Based Analysis of Visual Languages. *Formal Aspects of Computing*, 21, Mai 2009.
7. M. Del Fabro and P. Valduriez. Towards the efficient development of model transformations using model weaving and matching transformations. *SoSyM*, 8(3):305–324, 2009.
8. L. Geiger. Model Level Debugging with Fujaba. In *Proceedings of 6th International Fujaba Days*, pages 23–28, Dresden, Germany, 2008.
9. K. Jensen and L. M. Kristensen. *Coloured Petri Nets - Modeling and Validation of Concurrent Systems.* Springer, 2009.
10. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A Model Transformation Tool. *Science of Computer Programming*, 72(1-2):31–39, June 2008.
11. V. Kashyap and A. Sheth. Semantic and schematic similarities between database objects: A context-based approach. *The VLDB Journal*, 5(4):276–304, 1996.
12. A. Koenigs. Model Transformation with TGGs. In *Proc. of Model Transformations in Practice Workshop of MoDELS'05*, Montego Bay, Jamaica, 2005.
13. F. Legler and F. Naumann. A Classification of Schema Mappings and Analysis of Mapping Tools. In *Proc. of BTW'07*, 2007.
14. D. Plump. Termination of graph rewriting is undecidable. *Fundamental Informatics*, 33(2), 1998.
15. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.
16. M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin. On the Use of Higher-Order Model Transformations. In *Proc. of ECMDA-FA'09*, pages 18–33, 2009.
17. D. Varró, S. Varró-Gyapay, H. Ehrig, U. Prange, and G. Taentzer. Termination Analysis of Model Transformation by Petri Nets. In *Proc. of ICGT'06*, pages 260–274, 2006.
18. R. Wagner. Developing Model Transformations with Fujaba. In *Proc. of the 4th Int. Fujaba Days 2006*, pages 79–82, 2006.
19. M. Westergaard and L. M. Kristensen. The Access/CPN Framework: A Tool for Interacting with the CPN Tools Simulator. In *Proc. of the 30th Int. Conf. on Applications and Theory of Petri Nets*, pages 313–322, 2009.
20. M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schoenboeck, and W. Schwinger. Right or Wrong? - Verification of Model Transformations using Colored Petri Nets. In *Proc. of 9th DSM Workshop*, 2009.
21. M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger. Surviving the Heterogeneity Jungle with Composite Mapping Operators. In *Proc. of ICMT'10*, 2010.
22. M. Wimmer, A. Kusel, J. Schoenboeck, G. Kappel, W. Retschitzegger, and W. Schwinger. Reviving QVT Relations: Model-based Debugging using Colored Petri Nets. In *Proc. of MoDELS'09*, pages 727–732, 2009.
23. M. Wimmer, A. Kusel, J. Schönböck, T. Reiter, W. Retschitzegger, and W. Schwinger. Lets's Play the Token Game – Model Transformations Powered By Transformation Nets. In *Proc. of PNSE'09*, pages 35–50, 2009.

Abstractions for Petri Nets and Other models of Concurrency (APNOC) and Scalable and Usable Model checking for Petri nets and Other models of concurrency (SUMo)

Introduction

This chapter contains the papers presented at the joint SUMo/APNOC 2010 event: International Workshop on Scalable and Usable Model Checking for Petri Nets and other models of concurrency and the Second International Workshop on Abstractions for Petri Nets and Other Models of Concurrency held on June 22, 2010 in Braga, Portugal as a part of the International Conference on Applications and Theory of Petri Nets (PETRI NETS 2010). These joint workshops aim at bringing together, in an informal setting, researchers interested in all aspects of model checking and abstractions for different models of concurrency, whether this interest be practical or theoretical, primary or derived.

SUMo PC members have reviewed five submissions by researchers from three countries, including a number of submissions by authors from different countries. Each submission was reviewed by 4 program committee members.

APNOC PC members have reviewed six submissions by researchers from six countries, including a number of submissions by authors from different countries. Each submission was reviewed by at least 3 program committee members.

The workshops organizers would like to thank Daniel Kröning, from Oxford University (UK) for his Keynote talk entitled *"Unbounded is back"*.

The workshops organizers would also like to thank the authors of submitted papers for their interest in SUMo/APNOC. We also thank the program committee members and the external reviewers for their outstanding work during the reviewing process.

Last but not least, we thank the authors of the EasyChair conference management system which made the practical organization of the reviewing process considerably easier.

Didier Buchs (General chair SUMo)
Fabrice Kordon (General chair SUMo)
Alexander Serebrenik (PC chair APNOC)
Natalia Sidorova (PC chair APNOC)
Jeremy Spronston (PC chair SUMo)
Yann Thierry-Mieg (PC chair SUMo)

On Persistent Reachability in Petri Nets[*]

Kamila Barylska[1], Łukasz Mikulski[1], Edward Ochmanski[1,2]

[1] Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Torun
[2] Institute of Computer Science, Polish Academy of Sciences, Warszawa, Poland
{khama,frodo,edoch}@mat.uni.torun.pl

Abstract. The notion of persistency, based on the rule "no action can disable another one" is one of the classical notions in concurrency theory. In this paper, we deal with arbitrary place/transition nets, but concentrate on their persistent computations. It leads to an interesting decision problem: Is a given marking reachable with a persistent run? In order to study the persistent-reachability problem we define a class of nets, called nonviolence nets. We show that inhibitor nets can be simulated by the nonviolence nets, and that reachability and coverability problems are undecidable in the class of the nonviolence nets. Then we prove more: nonviolence nets can be simulated by the inhibitor nets, thus they are computationally equivalent to Turing machines.

1 Introduction

An action of a concurrent system is said to be persistent if, whenever it becomes enabled, it remains enabled until executed. This classical notion, introduced by Karp/Miller [9], is one of the most frequently discussed issues in the Petri net theory (papers [1,2,3,6,8,11,12] a.m.o.). A net is said to be persistent if each of its actions is persistent. And most of the papers about persistency deal with this subclass of place/transition nets). In this paper, we deal with arbitrary place/transition nets, but concentrate on their persistent computations. It leads to an interesting persistent-reachability problems: Is a given marking reachable (coverable) with a persistent run?

It is well known that the classical versions of the problems (Is a given marking reachable (coverable) in a given place/transition net?) are decidable (coverability: Karp/Miller [9], Hack [8]; reachability: Mayr [13], Kosaraju [10]). In order to study the persistent-reachability problem we introduce a class of nets, called nonviolence nets (Definition 3.1). They differ from place/transition nets only by the execution rule. Namely, only persistent executions are permitted. We show that inhibitor nets can be simulated by nonviolence nets (Proposition 4.4). Using this fact we prove that the reachability and coverability problems are undecidable in the class of the nonviolence nets (Propositions 4.5 and 4.7, respectively). Then we prove more: nonviolence nets can be simulated by the inhibitor nets (Proposition 4.8), thus the both are computationally equivalent to Turing machines.

[*] The research supported by Ministry of Science and Higher Education of Poland – grant N N206 258035

Many extensions of Petri nets are known to be Turing powerful: inhibitor nets, priority nets (Hack [7]), self-modifying nets (Valk [17]), for instance. There is also a Turing powerful model restricting the standard execution rules to maximal concurrent steps (Burkhard [4], see also Starke [16]). But all the models allow a fight for sharing resources (tokens), whereas our model works in a completely peaceful way.

In the concluding section we notice that the free-choice nonviolence nets are easy transformable to place/transition nets (not necessarily free-choice ones). Hence, the coverability and reachability problems are decidable in the class of the free-choice nonviolence nets.

2 Petri Nets – Basic Definitions

The set of non-negative integers is denoted by $\mathbb{N}$. Given a set X, the cardinality (number of elements) of X is denoted by $|X|$, the powerset (set of all subsets) by 2^X, the cardinality of the powerset is $2^{|X|}$. Multisets over X are members of $\mathbb{N}^X$, i.e. functions from X into $\mathbb{N}$. For convenience, if the set X is finite, multisets of $\mathbb{N}^X$ will be represented by vectors of $\mathbb{N}^{|X|}$.

2.1 Petri Nets and Their Computations

The definitions concerning Petri nets are mostly based on Desel/Reisig [5].

Net is a triple $N = (P,T,F)$, where:

- P and T are finite disjoint sets, of *places* and *transitions*, respectively;
- $F \subseteq P\times T \cup T\times P$ is a relation, called the *flow relation*.

For all $a\in T$ we denote: $^\bullet a = \{p\in P \mid (p,a)\in F\}$ – the set of *entries* to a
$a^\bullet = \{p\in P \mid (a,p)\in F\}$ – the set of *exits* from a

Petri nets admit a natural graphical representation. Nodes represent places and transitions, arcs represent the flow relation. Places are depicted by circles, and transitions by boxes. The set of all finite strings of transitions is denoted by T^*, the empty string is denoted by ε, the length of $w\in T^*$ is denoted by $|w|$, number of occurrences of a transition a in a string w is denoted by $|w|_a$.

Place/transition net (shortly, *p/t-net*) is a quadruple $S = (P,T,F,M_0)$, where:

- $N = (P,T,F)$ is a net, as defined above;
- $M_0\in\mathbb{N}^P$ is a multiset of places, named the *initial marking*; it is marked by *tokens* inside the circles, capacity of places is unlimited.

Multisets of places are named *markings*. In the context of place/transition nets, they are mostly represented by nonnegative integer vectors of dimension $|P|$, assuming that P is strictly ordered. The natural generalizations, for vectors, of arithmetic operations + and –, as well as the partial order $\leq$, all defined componentwise, are well known and their formal definitions are omitted.

A transition $a \in T$ is *enabled* in a marking M whenever ${}^\bullet a \leq M$ (all its entries are marked). If a is enabled in M, then it can be *executed*, but the execution is not forced. The execution of a transition a changes the current marking M to the new marking $M' = (M - {}^\bullet a) + a^\bullet$ (tokens are removed from entries, then put to exits). We shall denote: Ma for "a is enabled in M" and MaM' for "a is enabled in M and M' is the resulting marking". Then we say that MaM' is a *step*. This denotation we extend to strings of transitions: the empty string ε is enabled in any marking (always $M\varepsilon M$), a string $w=au$ ($a \in T$, $u \in T^*$) is enabled in a marking M whenever MaM' and u is enabled in M'. Predicates Mw and MwM' are defined like those for single transitions. If MwM' then we say that MwM' is a *computation* from M to M'. Note that any computation MwM' unambiguously defines all intermediate markings between M and M'.

If MwM', for some $w \in T^*$, then M' is said to be *reachable from M*. The set of all markings reachable from M is denoted by $[M\rangle$. Given a place/transition net $S=(P,T,F,M_0)$, the set $[M_0\rangle$ of all markings reachable from the initial marking M_0 is called the *reachability set* of S, and markings in $[M_0\rangle$ are said to be *reachable* in S.

We assume that the notions of *reachability* and *coverability graphs* are known to the reader. Their definitions can be found in any monograph or survey about Petri nets (see [5,16] or arbitrary else). Let us recall only that reachability graphs represent completely behaviours of nets, but are mostly infinite, while coverability graphs represent behaviours only partially, but are always finite. In Examples 2.2 and 2.3 we also use a notion of persistency graph – the reachability graph restricted to persistent steps.

2.2 Persistent Computations of Place/Transition Nets

The notion of persistency, proposed by Karp/Miller [9], belongs to the most important notions in concurrency theory. It is based on the behaviourally oriented rule "no action can disable another one", and generalizes the structurally defined notion of conflict-freeness.

Let $S=(P,T,F,M_0)$ be a place/transition net, and let M be a marking. The step MaM' is *persistent* iff $(\forall b \neq a)$ if Mb then $M'b$. The empty computation $M\varepsilon M$ is *persistent*; the computation $MaM'uM''$ is *persistent* iff the step MaM' is persistent and the computation $M'uM''$ is persistent. [In words: A computation is said to be *persistent* if any transition once enabled during this computation remains enabled until executed.] A p/t net is said to be *persistent* if it admits only persistent computations.

Example 2.1. *Non-persistent and persistent nets*

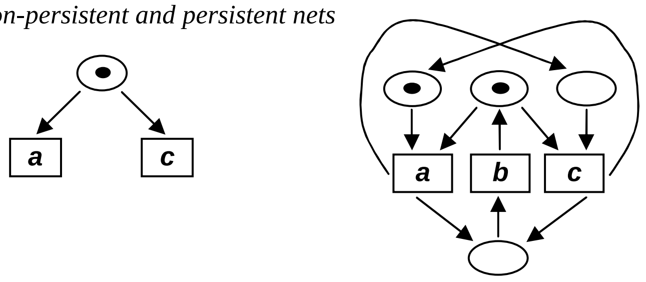


Fig. 1. A non-persistent (left) and persistent (right) place/transition nets

2.3 Persistent Reachability Problem

The problem of persistency ("Is a place/transition net persistent?"), raised by Landweber and Robertson in [11], has been proved to be decidable by Grabowski [6] and Mayr [12]. Most of p/t-nets, however, are not persistent, but some of their computations are persistent. In this paper, we are interested in markings that are reachable with persistent computations.

Let $S=(P,T,F,M_0)$ be a place/transition net, and let $M\in\mathbb{N}^P$ be a marking.

Reachability Problem: Is there a computation M_0wM?
In other words: Is the marking M reachable in the net S?

The Reachability Problem has been proved to be decidable by Mayr [13] and Kosaraju [10], after years of many author's efforts. A broad discussion, with a detailed proof, can be found in the book [15] of Reutenauer.

Let $S=(P,T,F,M_0)$ be a place/transition net, and let $M\in\mathbb{N}^P$ be a marking.

Persistent-Reachability Problem: Is there a persistent computation M_0wM?
In other words: Is the marking M reachable in the net S with a persistent run?

Obviously, if a p/t-net is persistent, then the persistent-reachability problem is equivalent to the classical one, thus decidable. We shall study the problem in general, for arbitrary p/t-nets. The following examples show difference between complete behaviours and persistent behaviours.

Example 2.2. *Comparison of the complete and persistent behaviours*

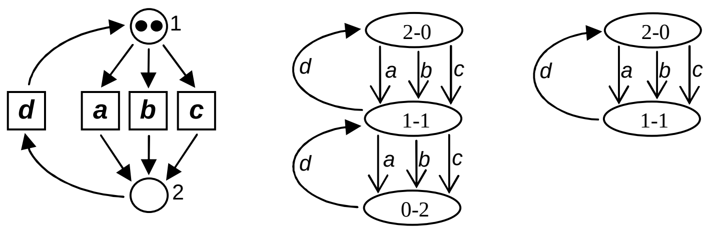


Fig. 2. A place/transition net and its reachability and persistency graphs

The above net is bounded (i.e. its reachability set is finite) and has infinite set of persistent computations. The example below shows an unbounded net (i.e. with infinite reachability set) with finite set (a singleton) of persistent computations.

Example 2.3. *Unbounded p/t-net with finite persistency graph*

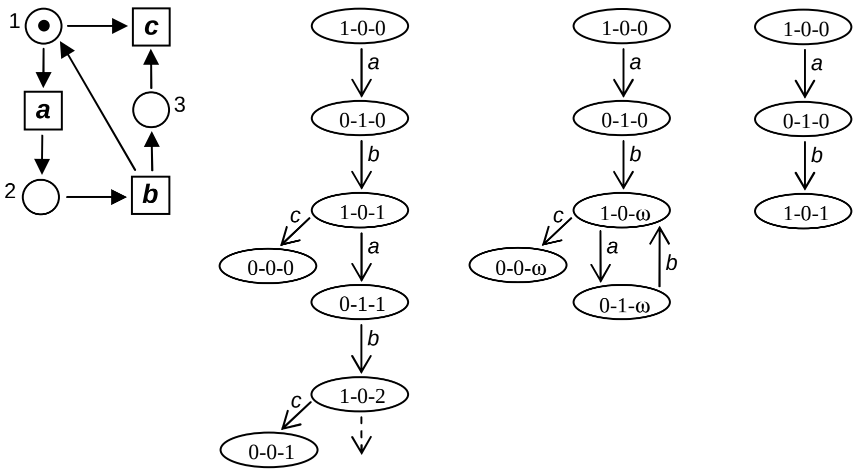


Fig. 3. A place/transition net and its reachability, coverability and persistency graphs

3 Nonviolence Petri Nets

In this section, we introduce the notion of nonviolence Petri nets. They differ from place/transition nets only by the execution rule. Namely, an enabled transition can be executed only if it is executable persistently (i.e. if its execution does not disable any other enabled transition). Therefore, we have to distinguish the notion "enabled" and "executable", that are synonymic in place/transition nets, but not in nonviolence nets.

Definition 3.1. *Nonviolence Petri Nets*
Nonviolence net is a quadruple $S = (P,T,F,M_0)$, exactly the same as in definition of place/transition nets. It differs from p/t-net by execution rules: A transition $a\in T$ is *enabled* in a marking M whenever ${}^\bullet a\leq M$ (all its entries are marked). A transition $a\in T$ is *executable* in M if it is enabled in M, and moreover the step MaM' is persistent. The execution of a leads to the resulting marking $M'=(M-{}^\bullet a)+a^\bullet$ (exactly same as in p/t-nets). We shall denote: $M\underline{a}$ for "a is executable in M" and $M\underline{a}M'$ for "a is executable in M and M' is the resulting marking". Then we say that $M\underline{a}M'$ is a *nonviolent step*. This denotation is naturally extended to strings $w\in T^*$. If $M\underline{w}M'$ then we say that $M\underline{w}M'$ is a *nonviolent computation*. Only nonviolent steps and computations are permitted in the nonviolence nets.

And now we can formulate the reachability and coverability problems for the nonviolence nets.

Let $S = (P,T,F,M_0)$ be a nonviolence net, and let $M\in\mathbb{N}^P$ be a marking.

NV-Reachability Problem:
Is there a nonviolence computation $M_0\underline{w}M$ in S?

NV-Coverability Problem:
Is there a marking $M'\geq M$ and a nonviolence computation $M_0\underline{w}M'$ in S?

3.2 From Place/Transition Nets to Nonviolence Nets

We shall show that every p/t-net can be simulated by a nonviolence net. It will be done by joining an external control to each transition of the net.

Let us consider an arbitrary p/t-net S. We transform it to the nonviolence net S' in the following way. To each transition a in the net S we join a switching transition a' and two new places p_a and q_a. We add the place q_a to the set of entries to a and to the set of exits from a'. We also add the place p_a to the set of exits from a and to the set of entries to a'. In initial marking we add one token to the place p_a. One can treat the constructed loop as a preparation of the transition a to execution.

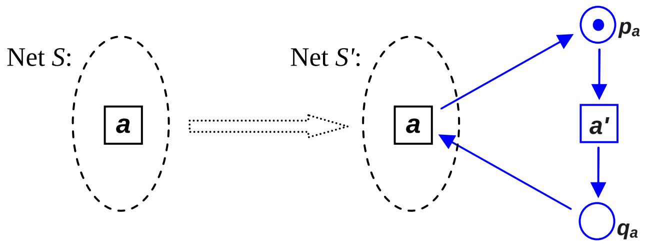


Fig. 4. Transforming a place/transition net into a nonviolence net

Let us also define, for every marking M in the net S, the marking $10M$ in the net S' as follows. For each place p in the net S we set $10M(p)=M(p)$, for each new place p_a we set $10M(p_a)$=1 and for each new place q_a we set $10M(q_a)$=0. With such definition, the initial marking in the net S' is $10M_0$, where M_0 is the initial marking in S. An obvious observation is that if transition a is executable in a marking M in the net S, then the sequence $a'a$ is executable in the marking $10M$ in the net S'.

Proposition 3.2. A marking M is reachable in a place/transition net S if and only if the marking $10M$ is reachable in the nonviolence net S'.
Proof. ($\Rightarrow$) Let M_0wM be a computation in S. Then replacing each a in w by $a'a$ we get a computation $10M_0\underline{w'}10M$ in the nonviolence net S'.
($\Leftarrow$) Let $10M_0\underline{w'}10M$ in the nonviolence net S'. The only difference between behaviours of S and S' is that before every transition a a transition a' must be executed. Subsequent execution of two (or more) primed actions may sometimes disable the nonviolence execution of actions that were executable in S. However, it would not make any new action executable. Therefore, erasing all primed actions in $\underline{w'}$, we get a computation w such that M_0wM is a computation in the p/t-net S. □

4 Comparison of Nonviolence Nets and Inhibitor Nets

In this section, we recall the notion of inhibitor nets and some of their properties (undecidability of the the reachability and coverability problems). Then we show that their computational power is equal to that of the nonviolence nets.

Definition 4.1. *Inhibitor Petri Nets*
Inhibitor net is a quintuple $S = (P,T,F,I,M_0)$, where (P,T,F,M_0) is a place/transition net and $I \subseteq P\times T$ is the set of inhibitor arcs (depicted by edges ended with a small empty circle). Sets of entries and exits are denoted by ${}^\bullet a$ and $a^\bullet$, as in p/t-nets; the set of *inhibitor entries* to a is denoted by ${}^\circ a=\{p\in P \mid (p,a)\in I\}$.

A transition $a\in T$ is *enabled* in a marking M whenever ${}^\bullet a\le M$ (all its entries are marked) and $(\forall p\in {}^\circ a)\ M(p)=0$ – all *inhibitor entries* to a are empty. And “executable” means “enabled”, like in p/t-nets. The execution of a leads to the resulting marking $M'= (M-{}^\bullet a)+a^\bullet$.

It is known that the inhibitor nets are computationally equivalent to Turing machines and the reachability problem in them is undecidable (Minsky [14], Hack [7]).

Fact 4.2. Reachability Problem is undecidable in the class of inhibitor nets.

Also the coverability problem is known to be undecidable in the class of inhibitor nets. We recall here the proving construction.

Let $S = (P,T,F,I,M_0)$ be an arbitrary inhibitor net with $P = \{p_1, \dots , p_k\}$, and let $M = [i_1, \dots , i_k]$ be a marking to be checked to be reachable. We extend it to an inhibitor net S', as follows: We add three new places p_0, p_{k+1}, p_{k+2} and two new transitions x, y, connected $p_0 \to x \to p_{k+1} \to y \to p_{k+2}$ (see figure 5). Moreover, we join the place p_0 with every transition of the net *S* by a self-loop (it is depicted symbolically on Figure 5), we add the arcs from p_n to *x*, weighted by i_n (for n=1,…,k), [We use here the weighted arcs; see remark below.] and the inhibitory arcs from all original places of *S* to *y*. And the initial marking M'_0 in *S'* is the following: $M'_0(p_0)=1$, $M'_0(p_n)=M_0(p_n)$ for n=1,…,k and $M'_0(p_{k+1})=M'_0(p_{k+2})=0$.

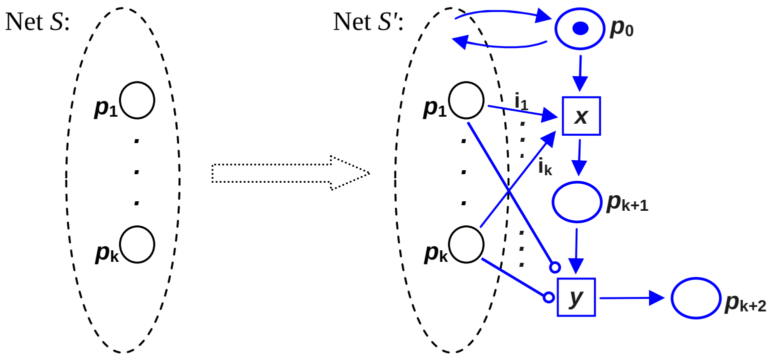


Fig. 5. Checking reachability with coverability in inhibitor nets

Remark. In this construction, we have used weighted (multiple) arcs. They are not mentioned, for simplicity, in our definitions; we assume that the notion is commonly known. Moreover, (place/transition or inhibitor) nets with multiple arcs can be transformed to the equivalent nets without them (Hack [8], Starke [16]).

Clearly, the marking $M = [i_1, \dots, i_k]$ is reachable in *S* if and only if the marking $M'(p_0)=M'(p_1)=\ldots=M'(p_k)=M'(p_{k+1})=0$ and $M'(p_{k+2})=1$ is coverable in *S'*. Hence, because of Fact 4.2, we have got

Fact 4.3. Coverability Problem is undecidable in the class of inhibitor nets.

4.1 From Inhibitor Nets to Nonviolence Nets

Let us consider an arbitrary inhibitor net *S*. In the transformation to the nonviolence net *S'* we will use the same idea as in the previous transformation. Like in that one, to each transition *a*, we add a transition *a'* and places p_a and q_a. Moreover, to each place *p*, in the net *S*, belonging to $^\circ a$ (i.e. being an inhibitor entry to *a*), we add a new transition a_p. Both places, *p* and q_a, are joined by self-loops with the new transition a_p. Finally, we remove all inhibitory arcs.

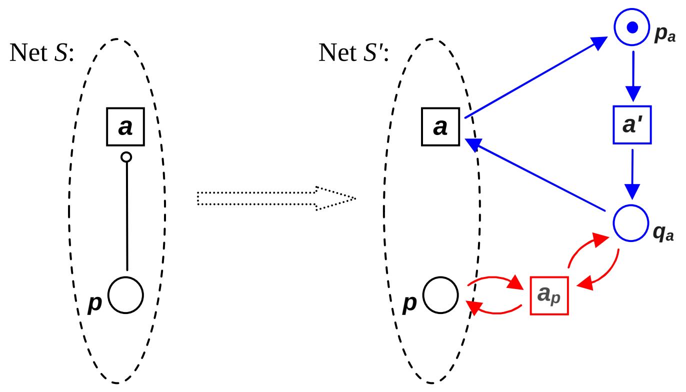


Fig. 6. Transforming an inhibitor net into a nonviolence net

Similarly to the construction of figure 4, for every marking M in the net S we define the marking $10M$ in the net S', in the same way. And the initial marking in the net S' is $10M_0$, where M_0 is the initial marking in the net S. In the same manner as in the previous case, if the transition a is executable in the marking M in the inhibitor net S, then the computation $a'a$ is executable in the marking $10M$ in the nonviolence net S'.

Proposition 4.4. The marking M is reachable in the inhibitor net S if and only if the marking $10M$ is reachable in the nonviolence net S'.
Proof. The proof is similar to that of Proposition 3.2. Remark that if an action a' is executed while a token resides in the place p (so a is not enabled in S), then a token will stuck in the place q_a and no marking of the form $10M$ will be reachable. □

Corollary 4.5. The NV-Reachability Problem is undecidable.
Proof. Directly from Proposition 4.4 and Fact 4.2. □

Proposition 4.6. The marking M is coverable in the inhibitor net S if and only if the marking $10M$ is coverable in the nonviolence net S'.
Proof. (⇒) If M is coverable in S then there is a marking $M' \geq M$ reachable in S. Hence, by Proposition 4.4, the marking $10M'$ is reachable in the nonviolence net S'. And clearly, $10M'$ covers $10M$.
(⇐) Notice that any marking (reachable in S') covering $10M$ is of the form $10M'$. And then $M' \geq M$ and M' is reachable in S (Proposition 4.4). So M is coverable in S. □

Corollary 4.7. The NV-Coverability Problem is undecidable.
Proof. Directly from Proposition 4.6 and Fact 4.3. □

4.2 From Nonviolence Nets to Inhibitor Nets

The inverse construction, transforming a nonviolence net into an inhibitor net, is more involved. Once more, we use the idea of predicting an executablement of transitions.

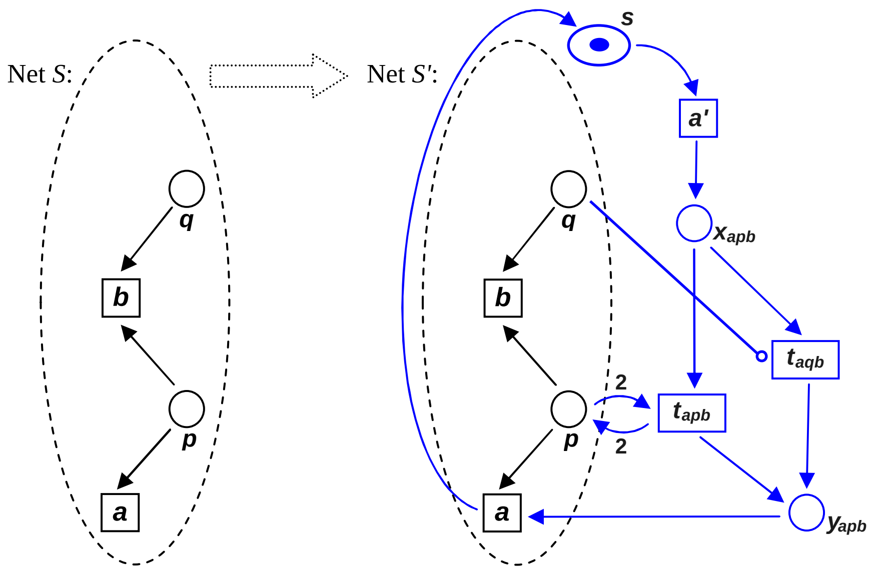


Fig. 7. Transforming a nonviolence net into an inhibitor net

Let S be a nonviolence net; we extend it to an inhibitor net S', as follows. First, we add one global place s, called the switch, which is an exit from every transition of the net S. Then, for every transition a of the net S, we add a transition a'; the switch place s is an entry to each of these primed transitions. An execution of a transition a' means a belief in the executablement of transition a in the nonviolence net S. After executing the transition a', the net S' checks, if the transition a was really executable in the nonviolence net S. It means that transition a is enabled and no other transition blocks its execution. In order to check it, we add, for every pair (p,b) such that $b \neq a$ and the place p is a common entry to the transitions a and b, the places x_{apb} and y_{apb} and the transition t_{apb}, as depicted on figure 7 above. The transition t_{apb} is able to move the token from x_{apb} to y_{apb} only if at least two tokens reside in the place p. [We use here the weighted arcs; see remark after figure 5.] By the nonviolence rules, a transition b blocks nothing if it is not enabled; it means that one of its entries is empty. In order to check it, we add the transitions t_{aqb}, for every entry q to transition b, different from p. Each of the transitions has one entry x_{apb}, one exit y_{apb} and one inhibitor arc from the place q, checking if q is empty. This construction allows to check, whether the transition b blocks the execution of a or not. If not, then a token moves from x_{apb} to y_{apb}, enabling a in S' if and only if it was executable in S.

For every marking M in the nonviolence net S we define a marking $10M$ in the inhibitor net S' as follows. For each place p inside the net S we set $10M(p)=M(p)$. For additional switch place s we set $10M(s)=1$ and for every place r added by the construction (i.e. for all places x_{apb} and y_{apb}) we set $10M(r)=0$. Directly from our construction, if a transition a is executable in a marking M in nonviolence net S then it is potentially executable in the marking $10M$ in inhibitor net S' (before executing a transition a we execute a transition a' and positively check all conditions to fill all places y_{apb}). The initial marking of S' is assumed to be $10M_0$.

Proposition 4.8. The marking M in the nonviolence net S is reachable if and only if the marking $10M$ is reachable in the inhibitor net S'.
Proof. (⇒) In the net S' we can reach marking $10M$, from the initial marking $10M_0$, by executing a transition a' before each transition a and checking the conditions.
(⇐) Executing any transition a from the original net S is possible only by predicting this execution by executing the transition a'. If we do a mistake, making wrong prediction, our net S' would reach a dead marking and stops. It means that if a marking $10M$ in the inhibitor net S' is reachable, then the only scenario of reaching that marking is correctly predicting and executing transitions from the net S. The correctness of our process of predicting means that we could just execute these transitions in the original, nonviolence net S, reaching marking M. Finally, marking M is reachable in the nonviolence net S, which ends the proof. □

Conclusions

We have proved (Propositions 4.4 + 4.8) that nonviolence nets are equivalent (in the marking reachability sense) to inhibitor nets. As the latter are Turing powerful, one can say that the former allow to do everything what possible without any fight. It is quite surprising, because persistent executions are only a part of arbitrary executions. But the price for the peace is undecidability. We have shown (Corollary 4.7) that even coverability, decidable in many extensions of place/transition nets, is undecidable in the class of the nonviolence nets.

Notice that free-choice (if $^{\bullet}a \cap {^{\bullet}b} \neq \emptyset$ then $^{\bullet}a = {^{\bullet}b}$) nonviolence nets can be simulated by place/transition nets (Figure 8), thus the classical decision problems (reachability, coverability) are decidable in the class of free-choice nonviolence nets.

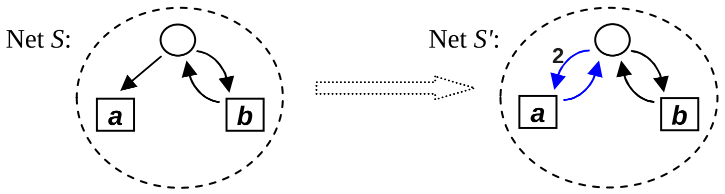


Fig. 8. Transforming a free-choice nonviolence net into a place/transition net

Let S be a free-choice nonviolence net. We replace every arc from a place, being a common entry of two (or more) transitions and is not a part of a self-loop, by two arcs: an arc from the place to the transition, weighted with 2, and an arc from the transition to the place, weighted with 1. And self-loops remain not changed. And the initial marking remains the same. Clearly, the place/transition net S' built this way works exactly as the free-choice nonviolence net S. A case of the free-choice nonviolence net is shown by Example 2.2. The above construction does not work for non-free-choice nonviolence nets, see Example 2.3, for instance.

It would be interesting to study some other subclasses of the class of nonviolence nets. Especially, to find a subclass of the class of nonviolence nets, computationally equivalent to the class of place/transition nets.

Acknowledgments

The paper was inspired by the Example 2.2. Great thanks are due to Ulla Goltz, who recalled to us that very old example.

References

1. E. Best: A Note on Persistent Petri Nets. Concurrency, Graphs and Models, LNCS 5065, pp. 427-438. Springer 2008.
2. E. Best, P. Darondeau: Decomposition Theorems for Bounded Persistent Petri Nets. Petri Nets 2008, LNCS 5062, pp. 33-51. Springer 2008.
3. E. Best, P. Darondeau: Separability in Persistent Petri Nets. Petri Nets 2010, LNCS 6128, pp. 246-266. Springer 2010.
4. H.-D. Burkhard: Ordered Firing in Petri Nets. Elektronische Informationsverarbeitung und Kybernetik 17(2/3), pp. 71-86, 1981.
5. J. Desel, W. Reisig: Place/Transition Petri Nets. Lectures on Petri Nets, LNCS 1491, pp. 122-173. Springer 1998.
6. J. Grabowski: The Decidability of Persistence for Vector Addition Systems. Information Processing Letters 11(1), pp. 20-23, 1980.
7. M. Hack: Petri Net Languages. Report MIT 124, 1975.
8. M. Hack: Decidability Questions for Petri Nets. Ph. D. Thesis, M.I.T. 1976.
9. R.M. Karp, R.E. Miller: Parallel program schemata. JCSS 3, pp. 147-195, 1969.
10. S.R. Kosaraju: Decidability of Reachability in Vector Addition Systems. Proc. of the 14th Annual ACM Symposium on Theory of Computing, pp. 267-281, 1982.
11. L.H. Landweber, E.L. Robertson: Properties of conflict-free and persistent Petri nets. Journal of ACM 25, pp. 352-364, 1978.
12. E. Mayr: Persistence of Vector Replacement Systems is Decidable. Acta Informatica 15, pp. 309-318, 1981.
13. E. Mayr: An algorithm for the general Petri net reachability problem. Proc. of the 13th Annual ACM Symposium on Theory of Computing, pp. 238-246, 1981.
14. M. Minsky: Computation: Finite and Infinite Machines. Prentice-Hall 1967.
15. C. Reutenauer: The Mathematics of Petri Nets. Prentice-Hall 1990.
16. P. H. Starke: Petri-Netze. VEB Berlin 1980.
17. R. Valk: Self-Modifying Nets, a Natural Extension of Petri Nets. ICALP 1978, LNCS 62, pp. 464-476. Springer 1978.

Process Refinement and Asynchronous Composition with Modalities

Dorsaf Elhog-Benzina[1], Serge Haddad[1], and Rolf Hennicker[2]

[1] LSV, CNRS & Ecole Normale Supérieure de Cachan
94235 CACHAN, 61 avenue du Président Wilson, France
{*elhog*, *haddad*}@lsv.ens-cachan.fr
[2] Institut für Informatik Universität München
Oettingenstraße 67 D-80538 München, Germany
hennicke@pst.ifi.lmu.de

Abstract. We propose a framework for the specification of infinite state systems based on Petri nets with distinguished *may*- and *must*-transitions (called modalities) which specify the allowed and the required behavior of refinements and hence of implementations. Formally, refinements are defined by relating the modal language specifications generated by two modal Petri nets according to the refinement relation for modal language specifications. We show that this refinement relation is decidable if the underlying modal Petri nets are weakly deterministic. We also show that the membership problem for the class of weakly deterministic modal Petri nets is decidable. As an important application of our approach we consider I/O-Petri nets which are obtained by asynchronous composition and thus exhibit inherently an infinite behavior.

Key words: Modal language specification and refinement, modal Petri net, weak determinacy, asynchronous composition, infinite state system.

1 Introduction

Specification in component-based software products. In component based software development, *specification* is an important phase of the components' life cycle. It aims to produce a formal description of the component's desired properties and behavior. A behavior specification can be presented either in terms of transition systems or in terms of logic, which both cannot be processed by a machine. Thus an *implementation phase* is required to produce concrete executable programs.

Modal specifications. Modal specifications have been introduced in [11] as a formal model for specification and implementation. A modal specification explicitly distinguishes between required actions and allowed ones. Required actions, denoted with the modality *must*, are compulsory in all possible implementations while allowed actions, denoted with the modality *may*, may happen but are not mandatory for an implementation. An implementation is seen as a specific specification in which all actions are required. Modal specifications are adequate for loose specifications as decisions can be delayed to later steps of the component's life cycle. Two different modal formalisms have been adopted in the literature, the first one, introduced in [8], is based on transition systems while the second one, introduced in [16], is a language-based model defining *modal language specifications*.

Modal refinement. A transformation step from a more abstract specification to a more concrete one is called a *refinement*. It produces a specification that is more precise, i.e. has less possible implementations. It follows that the set of implementations of a refinement is included in the set of possible implementations of the original specification.

Computational issues. While dealing with modal specifications, three main decision problems have been raised:

- (C) Consistency problem: Deciding whether a modal specification is consistent, i.e deciding whether it admits an implementation. Consistency is guaranteed for modal transition systems by definition [11] as every required transition is also an allowed one. In the case of mixed transition systems, i.e. if must-transitions are not necessarily may-transitions, the consistency decision problem is EXPTIME-complete in the size of the specification [2]. A better result is obtained with modal language specifications since a polynomial time algorithm has been proposed in [16].
- (CI) Common implementation: Deciding whether $k > 1$ modal specifications have a common implementation. Deciding common implementation of k modal transition systems is PSPACE-hard in the sum of the sizes of the k specifications [1] while it is EXPTIME-complete when dealing with mixed transition systems [2].
- (TR) Thorough refinement: Deciding whether one modal specification $\mathcal{S}$ thoroughly refines another modal specification $\mathcal{S}'$, i.e deciding whether the class of possible implementations of $\mathcal{S}$ is included in the class of possible implementations of $\mathcal{S}'$. The problem is also PSPACE-hard if $\mathcal{S}$ and $\mathcal{S}'$ are modeled with modal transition systems and it is EXPTIME-complete when they are modeled with mixed transition systems.

Limits of the existing formalisms. Both formalisms consider finite state systems. This restriction limits the existing approaches to synchronous composition. In fact asynchronous composition introduces a delay between the actions of sending and receiving a message between the communication partners. For instance, if a sender is always active while a receiver is always blocked, we naturally obtain an infinite state system.

Our contribution. We aim in this paper to deal with asynchronous composition of modal specifications while keeping most of the problems decidable. Petri nets seem to be an appropriate formalism to our needs since they allow for a finite representation of infinite state systems. Automata with queues might be another alternative for modeling infinite state systems, but all significant problems (e.g. the reachability problem) are known to be undecidable while they are decidable when considering deterministic Petri nets. In our approach we consider

Petri nets with silent transitions and we define the generalized notion of a *weakly deterministic* Petri net, as a variant of deterministic Petri nets that keeps decidability for our targeted decision problems. We also follow a language approach and use weakly deterministic Petri nets as a device to generate languages in the same way as Raclet et. al. use deterministic finite transition systems as a device to generate regular languages. Moreover, we extend Petri nets by modalities for the transitions. In this setting, we are mainly interested in the following decision problems:

1. Decide whether a given Petri net is weakly deterministic.
2. Decide whether a given language generated by a Petri net $\mathcal{N}$ is included in the language generated by a given *weakly deterministic* Petri net $\mathcal{N}'$.
3. Decide whether a given modal Petri net is (modally) weakly deterministic.
4. Given two modal language specifications $\mathcal{S}(\mathcal{M})$ and $\mathcal{S}(\mathcal{M})$ generated by two *weakly deterministic* modal Petri nets $\mathcal{M}$ and $\mathcal{M}'$ respectively, decide whether $\mathcal{S}(\mathcal{M}')$ is a modal language specification refinement of $\mathcal{S}(\mathcal{M})$.

We show that all the above mentioned problems are decidable. As a particular important application of our approach we consider I/O-Petri nets which are obtained by asynchronous composition and thus exhibit an infinite behavior. Since transitions with internal labels, in particular those obtained by internal communications, are not relevant for refinements we abstract them away by a general abstraction operator on modal I/O-Petri nets.

Outline of the paper. We proceed by reviewing in Sect. 2 modal language specifications and the associated notion of refinement. In Sect. 3 we summarize basics of Petri net theory and introduce *weakly deterministic* Petri nets. We then introduce *modal* Petri nets, their generated language specifications and we extend the concept of weak determinacy to Petri nets with modalities. In Sect. 4, we consider modal Petri nets over an I/O-alphabet (with distinguished input, output, and internal labels) and define their asynchronous composition. We also define the *abstraction* of an I/O-Petri net by relabeling internal labels to the empty word. In Sect. 5 we present the decision algorithms of the issues mentioned above. Finally we conclude in Sect. 6.

2 Modal Language Specifications

Modal specification was first introduced by Larsen and Thomsen in [11] with finite state modal transition systems by defining restrictions on specifications transitions by the mean of *may* (allowed) and *must* (required) modalities. This notion has then been adapted by Raclet in his PhD thesis who applied it to regular languages. Finite transition systems are low level models based on states, which limits the level of abstraction of the specification. They also lead to state number explosion while composing systems with an important number of states. Moreover, the complexity of the decision problems discussed above are better with modal languages than with modal transition systems. Besides, modal refinement is sound and complete with the language-based formalism while it is non-complete with transition system based formalism [9]. So a language approach is more convenient to deal with modal specification issues. Next we review the definition of modal language specification and refinement between specifications as introduced in [16].

Definition 1 (Modal language specification). *A* modal language specification *$\mathcal{S}$ over an alphabet Σ is a triple $\langle \mathcal{L}, may, must \rangle$ where $\mathcal{L} \subseteq \Sigma^*$ is a prefix-closed language over Σ and $may, must : \mathcal{L} \rightarrow P(\Sigma)$ are partial functions. For every trace $u \in \mathcal{L}$,*

- *$a \in may(u)$ means that the action a is allowed after u,*
- *$a \in must(u)$ means that the action a is required after u,*[1]
- *$a \notin may(u)$ means that a is forbidden after u.*

The modal language specification S is consistent *if the following two conditions hold:*
(C1) $\forall\ u \in \mathcal{L}\ must(u) \subseteq may(u)$
(C2) $\forall\ u \in \mathcal{L}\ may(u) = \{a \in \Sigma \mid u.a \in \mathcal{L}\}$

Example 1. Let us consider the example of a message producer and a message consumer represented in figure 1.

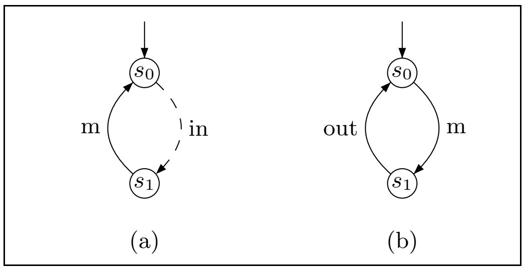


Fig. 1. Modal transition systems for a producer (a) and a consumer (b)

In the producer system, transition $s_0 \xrightarrow{in} s_1$ is allowed but not required (dashed line) while transition $s_1 \xrightarrow{m} s_0$ is required (continuous line). In the consumer model, all transitions are required. The languages associated with the the producer is $\mathcal{L} \equiv (in.m)^* + in.(m.in)^*$. The associated modal language specification is then $\langle \mathcal{L}, may, must \rangle$ with:

- $\forall\ u\ \in\ (in.m)^*\ must(u) = \emptyset \wedge may(u) = \{in\}$

[1] If $must(u)$ contains more than one element, this means that any correct implementation must have after the trace u (at least) the choice between all actions in $must(u)$.

– $\forall\ u \in in.(m.in)^* \ must(u) = may(u) = \{m\}$

Similarly, the modal language specification associated with the consumer is $\langle (m.out)^* + m.(out.m)^*, may, must\rangle$ with:

– $\forall\ u\ \in\ (m.out)^* \ must(u) = may(u) = m$
– $\forall\ u \in m.(out.m)^* \ must(u) = may(u) = out$

Modal language specifications are related by a refinement relation that translates the degree of specialization. One can obtain a possible refinement by either removing some allowed events or changing them to required events. So we review the formal definition of modal language specification refinement.

Definition 2 (Modal language specification refinement).
Let $\mathcal{S} = \langle \mathcal{L}, may, must\rangle$ and $\mathcal{S}' = \langle \mathcal{L}', may', must'\rangle$ be two consistent modal language specifications over the same alphabet Σ. $\mathcal{S}'$ is a modal language specification refinement *of $\mathcal{S}$, denoted by $\mathcal{S}' \sqsubseteq \mathcal{S}$, if:*

– $\mathcal{L}' \subseteq \mathcal{L}$,
– *for every $u \in \mathcal{L}'$, $must(u) \subseteq must'(u)$, i.e every required action after the trace u in $\mathcal{L}$ is a required action after u in $\mathcal{L}'$.*

3 Modal Petri Nets

In contrast to modal language specifications, Petri nets provide an appropriate tool to specify the behavior of infinite state systems in a finitary way. Therefore we are interested in the following to combine the advantages of Petri nets with the flexibility provided by modalities for the definition of refinements. Of particular interest are Petri nets which support silent transitions and hence are able to characterize observable system behaviors. In the following we will consider such Petri nets and extend them by modalities. Then we will use modal Petri nets as a device to generate modal language specifications as the basis for refinement. First, we review basic definitions of Petri net theory and we will introduce weakly deterministic Petri nets.

3.1 Basics of Petri Nets

Definition 3 (Labeled Petri Net). *A* labeled Petri net *over an alphabet Σ is a tuple $\mathcal{N} = (P, T, W^-, W^+, \lambda, m_0)$ where:*

– *P is a finite set of places,*
– *T is a finite set of transitions with $P \cap T = \emptyset$,*
– *W^- (resp. W^+) is a matrix indexed by $P \times T$ with values in $\mathbb{N}$;*
  W^- (resp. W^+) is called the backward (forward) incidence matrix*,*
– *$\lambda : T \to \Sigma \cup \{\epsilon\}$ is a transition labeling function where ϵ denotes the empty word, and*
– *$m_0 : P \mapsto \mathbb{N}$ is an initial marking.*

A marking is a mapping $m : P \mapsto \mathbb{N}$. The labeling function is extended to sequences of transitions $\sigma = t_1 t_2 ... t_n \in T^*$ where $\lambda(\sigma) = \lambda(t_1)\lambda(t_2)...\lambda(t_n)$. For each $t \in T$, ${}^\bullet t$ ($t^\bullet$ resp.) denotes the set of *input (output) places* of t. i.e. ${}^\bullet t = \{p \in P \mid W^-(p,t) > 0\}$ ($t^\bullet = \{p \in P \mid W^+(p,t) > 0\}$ resp.). Likewise for each $p \in P$, ${}^\bullet p$ ($p^\bullet$) denotes the set of *input (output) transitions* of p i.e. ${}^\bullet p = \{t \in T \mid W^+(p,t) > 0\}$ ($p^\bullet = \{t \in T \mid W^-(p,t) > 0\}$ resp.). The input (output resp.) vector of a transition t is the column vector of matrix W^- (W^+ resp.) indexed by t.

In the sequel labeled Petri nets are simply called Petri nets. We have not included final markings in the definition of a Petri net here, because we are interested in potentially infinite system behaviors. We now introduce the semantic of a net.

Definition 4 (Firing rule). *Let $\mathcal{N}$ be a Petri net. A transition $t \in T$ is firable in a marking m, denoted by $m[t\rangle$, iff $\forall p \in {}^\bullet t,\ m(p) \geq W^-(p,t)$. The set of firable transitions in a marking m is defined by $firable(m) = \{t \in T \mid m[t\rangle\}$. For a marking m and $t \in firable(m)$, the firing of t from m leads to the marking m', denoted by $m[t\rangle m'$, and defined by $\forall p \in P, m'(p) = m(p) - W^-(p,t) + W^+(p,t)$.*

Definition 5 (Firing sequence). *Let $\mathcal{N}$ be a Petri net with the initial marking m_0. A finite sequence $\sigma \in T^*$ is firable in a marking m and leads to a marking m', also denoted by $m[\sigma\rangle m'$, iff either $\sigma = \epsilon$ or $\sigma = \sigma_1.t$ with $t \in T$ and there exists m_1 such that $m[\sigma_1\rangle m_1$ and $m_1[t\rangle m'$. For a marking m and $\sigma \in T^*$, we write $m[\sigma\rangle$ if σ is firable in m. The set of reachable markings is defined by $Reach(\mathcal{N}, m_0) = \{m \mid \exists \sigma \in T^* \text{such that } m_0[\sigma\rangle m\}$.*

The reachable markings of a Petri net correspond to the reachable states of the modeled system. Since the capacity of places are not restricted, the set of the reachable markings of the Petri nets considered here may be infinite. Thus, Petri nets can model infinite state systems. Now, let us define the language generated by a labeled Petri net.

Definition 6 (Petri net language). *Let $\mathcal{N}$ be a labeled Petri net over the alphabet Σ. The language generated by $\mathcal{N}$ is:*

$$\mathcal{L}(\mathcal{N}) = \{u \in \Sigma^* \mid \exists \sigma \in T^* \text{ and } m \text{ such that } \lambda(\sigma) = u \text{ and } m_0[\sigma\rangle m\} \ .$$

A particular interesting class of Petri nets are deterministic Petri nets as defined, e.g., in [14]. Since in our approach we also deal with silent transitions, which are not taken into account for the generated languages, we are interested in a more general notion of determinacy which allows us to abstract from silent transitions. This leads to our notion of weakly deterministic Petri net. We call a Petri net weakly deterministic if any two firing sequences σ and σ' which produce the same word u can be mutually extended to produce the same continuations of u. In this sense our notion of weak deterministic Petri net corresponds to Milner's (weak) determinacy [13] and to the concept of a weakly deterministic transition system in [5]. It is also related to to the notion of output-determinacy in [7].

Definition 7 (Weakly Deterministic Petri net). *Let $\mathcal{N}$ be a labeled Petri net with initial marking m_0 and labeling function $\lambda : T \to \Sigma \cup \{\epsilon\}$. For any marking m, let*

$$may_{mk}(m) = \{a \in \Sigma \mid \exists \sigma \in T^* \text{ such that } \lambda(\sigma) = a \text{ and } m[\sigma\rangle\}$$

$\mathcal{N}$ is called weakly deterministic*, if for each $\sigma, \sigma' \in T^*$ with $\lambda(\sigma) = \lambda(\sigma')$ and for any markings m and m' with $m_0[\sigma\rangle m$ and $m_0[\sigma'\rangle m'$, we have $may_{mk}(m) = may_{mk}(m')$.*

Since weakly deterministic Petri nets play an important role in our further development it is crucial to know, whether a given Petri net belongs to the class of weakly deterministic Petri nets. This leads to our first decision problem stated below. Our second decision problem is motivated by the major goal of this work to provide formal support for refinement in system development. Since, in a simple form, refinement can be defined by language inclusion, we want to be able to decide this. Unfortunately, it is well-known that the language inclusion problem for Petri nets is undecidable [4]. However, in [14] it has been shown that for languages generated by deterministic Petri nets the language inclusion problem is decidable. Therefore we are interested in a generalization of this result for languages generated by weakly deterministic Petri nets which leads to our second decision problem. Observe that we do not require $\mathcal{N}'$ to be weakly deterministic.

First decision problem. Given a labeled Petri net $\mathcal{N}$, decide whether $\mathcal{N}$ is weakly deterministic.

Second decision problem. Let $\mathcal{L}(\mathcal{N})$ and $\mathcal{L}(\mathcal{N}')$ be two languages with the same alphabet Σ such that $\mathcal{L}(\mathcal{N})$ is generated by a weakly deterministic Petri net $\mathcal{N}$ and $\mathcal{L}(\mathcal{N}')$ is generated by a Petri net $\mathcal{N}'$. Decide whether $\mathcal{L}(\mathcal{N}')$ is included in $\mathcal{L}(\mathcal{N})$.

3.2 Modal Petri Nets

In the following we introduce modal Petri nets which extend, similarly to modal language specifications, Petri nets with modalities *may* and *must* on its transitions.

Definition 8 (Modal Petri net). *A* modal Petri net *$\mathcal{M}$ over an alphabet Σ is a pair $\mathcal{M} = (\mathcal{N}, T_\Box)$ where $\mathcal{N} = (P, T, W^-, W^+, \lambda, m_0)$ is a labeled Petri net over Σ and $T_\Box \subseteq T$ is a set of* must (required) *transitions. The set of* may (allowed) *transitions is the set of transitions T.*

Example 2. Let us consider the same example of a message producer and a message consumer (see Fig. 2). The consumer may receive an input *in* (white transition) but must produce a message *m* (black transition). The consumer must receive a message *m* and produce an output *out*.

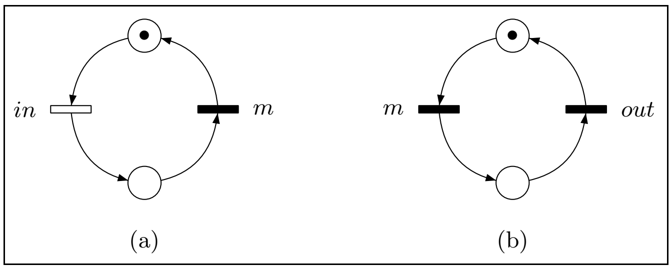


Fig. 2. Modal Petri nets for a producer (a) and a consumer (b)

Any modal Petri net $\mathcal{M} = (\mathcal{N}, T_\square)$ gives rise to the construction of a modal language specification (see Def. 1) which extends the language $\mathcal{L}(\mathcal{N})$ by *may* and *must* modalities. Similarly to the construction of $\mathcal{L}(\mathcal{N})$ the definition of the modalities should abstract from silent transitions in an appropriate way. While for the *may* modality this is rather straightforward, special care has to be taken for the definition of the *must* modality. For this purpose, we introduce the following auxiliary definition which expresses, for each marking m, the set $must_{mk}(m)$ of all labels $a \in \Sigma$ which must be produced by firing (in m) some silent *must*-transitions succeeded by a *must*-transition labeled by a. This means that the label a must be produced as the next visible label by some firing sequence of m. Formally, for any marking m, let

$$must_{mk}(m) = \{a \in \Sigma \mid \exists \sigma \in T_\square^*, t \in T_\square \textit{ such that } \lambda(\sigma) = \epsilon, \lambda(t) = a \textit{ and } m[\sigma t\rangle\}$$

On this basis we can now compute for each word $u \in \mathcal{L}(\mathcal{N})$ and for each marking m reachable by firing a sequence of transitions which produces u (and which has no silent transition at the end[2]), the set $must_{mk}(m)$. Then the labels in $must_{mk}(m)$ must be the possible continuations of u in the generated modal language specification.

Definition 9 (Modal Petri Net Language Specification). *Let* $\mathcal{M} = (\mathcal{N}, T_\square)$ *be a modal Petri net over an alphabet* Σ *such that* $\lambda : T \to \Sigma \cup \{\epsilon\}$ *is the labeling function and* m_0 *is the initial marking of* $\mathcal{N}$. $\mathcal{M}$ *generates the* modal language specification $\mathcal{S}(\mathcal{M}) = \langle \mathcal{L}(\mathcal{N}), may, must\rangle$ *where:*

– $\mathcal{L}(\mathcal{N})$ *is the language generated by the Petri net* $\mathcal{N}$*,*
– $\forall u \in \mathcal{L}(\mathcal{N})$*,* $may(u) =$
  $\{a \in \Sigma \mid \exists \sigma \in T^* \textit{ and } m \textit{ such that } \lambda(\sigma) = u, m_0[\sigma\rangle m \textit{ and } a \in may_{mk}(m)\}$*,*
– $\forall u \in \mathcal{L}(\mathcal{N}), x \in \Sigma$*,*
  - $must(\epsilon) = must_{mk}(m_0)$*,*
  - $must(ux) = \{a \in \Sigma \mid \exists \sigma \in T^*, t \in T \textit{ and } m \textit{ such that } \lambda(\sigma) = u, \lambda(t) = x, m_0[\sigma t\rangle m \textit{ and } a \in must_{mk}(m)\}$*.*

[2] We require it to avoid false detection of must transitions starting from intermediate markings.

Remark 1. Any modal language specification generated by a modal Petri net is consistent.

Example 3. Let us consider the modal Petri net in Fig. 3.

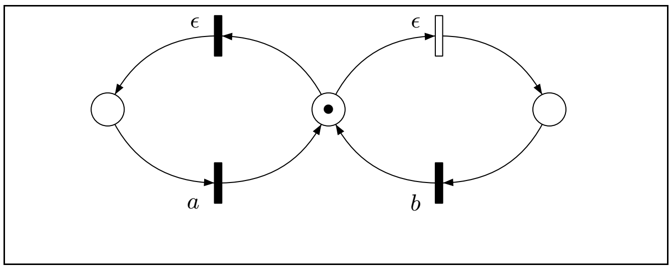


Fig. 3. Modal Petri net with silent transitions

The modal language specification generated by this net consists of the language $\mathcal{L}$ presented by the regular expression $(a^*b^*)^*$ and of the modalities $may(u) = \{a, b\}$, and $must(u) = \{a\}$ for $u \in \mathcal{L}$. Note that b is not a *must* as it is preceeded by a silent *may*-transition (which can be omitted in a refinement).

The notion of weakly deterministic Petri net can be extended to modal Petri nets by taking into account an additional condition for *must*-transitions. This condition ensures that for any two firing sequences σ and σ' which produce the same word u, the continuations of u produced by firing sequences of *must*-transitions after σ and σ' are the same.

Definition 10 (Weakly Deterministic Modal Petri Net). *Let $\mathcal{M} = (\mathcal{N}, T_\Box)$ be a modal Petri net over an alphabet Σ such that $\lambda : T \to \Sigma \cup \{\epsilon\}$ is the labeling function of $\mathcal{N}$. For any marking m, let*

$$must_{mk}(m) = \{a \in \Sigma \mid \exists \sigma \in T_\Box^*, t \in T_\Box \textit{ such that } \lambda(\sigma) = \epsilon, \lambda(t) = a \textit{ and } m[\sigma t\rangle\}$$

$\mathcal{M}$ is (modally) weakly deterministic*, if*

1. *$\mathcal{N}$ is weakly deterministic, and*
2. *for each $\sigma, \sigma' \in T^*$ with $\lambda(\sigma) = \lambda(\sigma')$ and for any markings m and m' with $m_0[\sigma\rangle m$ and $m_0[\sigma'\rangle m'$, we have $must_{mk}(m) = must_{mk}(m')$.*

Remark 2. For any weakly deterministic modal Petri net $\mathcal{M} = (\mathcal{N}, T_\Box)$ the definition of the modalities of its generated modal language specification $\mathcal{L}(\mathcal{M}) = \langle \mathcal{L}(\mathcal{N}), may, must\rangle$ can be simplified as follows:

– $\forall u \in \mathcal{L}(\mathcal{N})$, let $\sigma \in T^*$ and let m be a marking such that $\lambda(\sigma) = u$ and $m_0[\sigma\rangle m$, then $may(u) = may_{mk}(m)$.

– $\forall u \in \mathcal{L}(\mathcal{N}), x \in \Sigma$, let $\sigma \in T^*, t \in T$ and let m be a marking such that $\lambda(\sigma) = u, \lambda(t) = x$ and $m_0[\sigma t\rangle m$, then $must(ux) = must_{mk}(m)$. Moreover, $must(\epsilon) = must_{mk}(m_0)$.

Example 4. Let us consider the modal Petri net in Fig. 4.

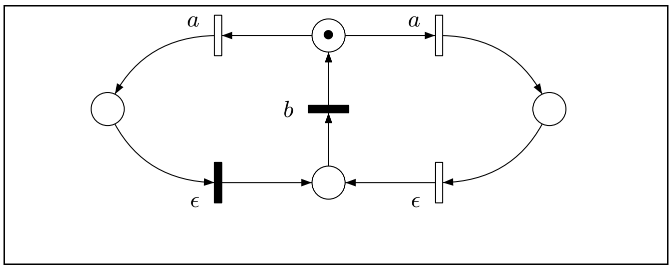


Fig. 4. Non weakly deterministic modal Petri net

Let t_l (t_r resp.) be the left (right resp.) transition labeled with a and let m_l (m_r resp.) be the marking obtained by firing the transition t_l (t_r resp.). Obviously, both transitions produce the same letter but $must(m_l) = \{b\}$ while $must(m_r) = \emptyset$ (since the silent transition firable in m_r is only a *may*-transition).

The two decision problems of Sect. 3.1 induce the following obvious extensions in the context of modal Petri nets and their generated modal language specifications. Observe that for the refinement problem we require that both nets are weakly deterministic.

Third decision problem. Given a modal Petri net $\mathcal{M}$, decide whether $\mathcal{M}$ is (modally) weakly deterministic.

Fourth decision problem. Let $\mathcal{S}(\mathcal{M})$ and $\mathcal{S}(\mathcal{M}')$ be two modal language specifications over the same alphabet Σ such that $\mathcal{S}(\mathcal{M})$ ($\mathcal{S}(\mathcal{M}')$ resp.) is generated by a weakly deterministic modal Petri net $\mathcal{M}$ ($\mathcal{M}$ resp.). Decide whether $\mathcal{S}(\mathcal{M}')$ is a modal language specification refinement of $\mathcal{S}(\mathcal{M})$.

4 Modal I/O-Petri Nets

In this section we consider modal Petri nets where the underlying alphabet Σ is partitioned into disjoint sets *in*, *out*, and *int* of input, output and internal labels resp., i.e $\Sigma = in \uplus out \uplus int$. Such alphabets are called *I/O-alphabets* and modal Petri nets over an I/O-alphabet are called *modal I/O-Petri nets.* The discrimination of input, output and internal labels provides a means to specify the communication abilities of a Petri net and hence provides an appropriate basis for Petri net composition. A syntactic requirement for the composability of two modal I/O-Petri nets is that their labels overlap only on complementary types, i.e. their underlying alphabets must be composable. Formally, two I/O-alphabets $\Sigma_1 = in_1 \uplus out_1 \uplus int_1$ and $\Sigma_2 = in_2 \uplus out_2 \uplus int_2$ are *composable* if $\Sigma_1 \cap \Sigma_2 \subseteq (in_1 \cap out_2) \cup (in_2 \cap out_1)$.[3]

Definition 11 (Alphabet Composition). *Let $\Sigma_1 = in_1 \uplus out_1 \uplus int_1$ and $\Sigma_2 = in_2 \uplus out_2 \uplus int_2$ be two composable I/O-alphabets. The composition of Σ_1 and Σ_2 is the I/O-alphabet $\Sigma_c = in_c \uplus out_c \uplus int_c$ where:*

- $in_c = (in_1 \setminus out_2) \uplus (in_2 \setminus out_1)$,
- $out_c = (out_1 \setminus in_2) \uplus (out_2 \setminus in_1)$,
- $int_c = \{*a \mid * \in \{!, ?\},\ a \in \Sigma_1 \cap \Sigma_2\} \uplus int_1 \uplus int_2$.

The input and output labels of the alphabet composition are the input and output labels of the underlying alphabets which are not used for communication, and hence are "left open". The internal labels of the alphabet composition are obtained from the internal labels of the underlying alphabets and from their shared input/output labels. Since we are interested here in asynchronous communication each shared label a is duplicated to $!a$ and $?a$ where the former represents the asynchronous sending of a message and the latter represents the receiption of the message (at some later point in time). We are now able to define the asynchronous composition of composable modal I/O-Petri nets. For the realization of the asynchronous communication, for each shared label a a new place p_a is introduced in the composition.

Definition 12 (Asynchronous Composition). *Let $\mathcal{M}_1 = (\mathcal{N}_1, T_{1_\Box})$, $\mathcal{N}_1 = (P_1, T_1, W_1^-, W_1^+, \lambda_1, m_{1_0})$ be a modal I/O-Petri net over the I/O-alphabet $\Sigma_1 = in_1 \uplus out_1 \uplus int_1$ and let $\mathcal{M}_2 = (\mathcal{N}_2, T_{2_\Box})$, $\mathcal{N}_2 = (P_2, T_2, W_2^-, W_2^+, \lambda_2, m_{2_0})$ be a modal I/O-Petri net over the I/O-alphabet $\Sigma_2 = in_2 \uplus out_2 \uplus int_2$. $\mathcal{M}_1$ and $\mathcal{M}_2$ are* composable *if $P_1 \cap P_2 = \emptyset$, $T_1 \cap T_2 = \emptyset$ and if Σ_1 and Σ_2 are composable. In this case, their* asynchronous composition *$\mathcal{M}_c$, also denoted by $\mathcal{M}_1 \otimes_{as} \mathcal{M}_2$, is the modal Petri net over the alphabet composition Σ_c, defined as follows:*

- $P_c = P_1 \uplus P_2 \uplus \{p_a \mid a \in \Sigma_1 \cap \Sigma_2\}$ *(each p_a is a new place)*
- $T_c = T_1 \uplus T_2$ *and* $T_{c,\Box} = T_{1_\Box} \uplus T_{2_\Box}$
- W_c^- *(resp. W_c^+) is the $P_c \times T_c$ backward (forward) incidence matrix defined by:*
  - *for each $p \in P_1 \cup P_2,\ t \in T_c$,*

$$W_c^-(p,t) = \begin{cases} W_1^-(p,t) & \textit{if } p \in P_1 \textit{ and } t \in T_1 \\ W_2^-(p,t) & \textit{if } p \in P_2 \textit{ and } t \in T_2 \\ 0 & \textit{otherwise} \end{cases}$$

[3] Note that for composable alphabets $in_1 \cap in_2 = \emptyset$ and $out_1 \cap out_2 = \emptyset$.

$$W_c^+(p,t) = \begin{cases} W_1^+(p,t) \text{ } \textit{if } p \in P_1 \textit{ and } t \in T_1 \\ W_2^+(p,t) \text{ } \textit{if } p \in P_2 \textit{ and } t \in T_2 \\ 0 \qquad\qquad \textit{otherwise} \end{cases}$$

- *for each $p_a \in P_c \setminus \{P_1 \cup P_2\}$ with $a \in \Sigma_1 \cap \Sigma_2$ and for each $t \in T_i$ with $i \in \{1,2\}$,*

$$W_c^-(p_a,t) = \begin{cases} 1 \textit{ if } a = \lambda_i(t) \in in_i \cap out_j \textit{ with } i \neq j \\ 0 \textit{ otherwise} \end{cases}$$

$$W_c^+(p_a,t) = \begin{cases} 1 \textit{ if } a = \lambda_i(t) \in in_j \cap out_i \textit{ with } i \neq j \\ 0 \textit{ otherwise} \end{cases}$$

– *$\lambda_c : T_c \to \Sigma_c$ is defined, for all $t \in T_c$ and for $i \in \{1,2\}$, by*

$$\lambda_c(t) = \begin{cases} \lambda_i(t) \quad \textit{if } t \in T_i, \ \lambda_i(t) \notin \Sigma_1 \cap \Sigma_2 \\ ?\lambda_i(t) \textit{ if } t \in T_i, \ \lambda_i(t) \in in_i \cap out_j \textit{ with } i \neq j \\ !\lambda_i(t) \quad \textit{if } t \in T_i, \ \lambda_i(t) \in in_j \cap out_i \textit{ with } i \neq j \end{cases}$$

– *m_{c_0} is defined, for each place $p \in P_c$, by*

$$m_{c_0}(p) = \begin{cases} m_{1_0}(p) \textit{ if } p \in P_1 \\ m_{2_0}(p) \textit{ if } p \in P_2 \\ 0 \qquad\quad \textit{otherwise} \end{cases}$$

Proposition 1. *The asynchronous composition of two weakly deterministic modal I/O-Petri nets is again a weakly deterministic modal I/O-Petri net.*

We will not give a proof of this fact here, since it is not a main result of this work.

Example 5. We consider the two modal producer and consumer Petri nets of Fig. 2 as I/O-nets where the producer alphabet has the input label *in*, the output label m and no internal labels while the consumer has the input label m, the output label *out* and no internal labels as well. Obviously, both nets are composable and their asynchronous composition yields the net shown in Fig. 5. The alphabet of the composed net has the input label *in*, the output label *out* and the internal labels $?m$ and $!m$. The Petri net composition describes an infinite state system and its generated modal language specification has a language which is no more regular.

When studying refinements it is crucial to rely on the observable behavior specified by a requirements specification while one can abstract from internal actions performed by a more concrete specification or an implementation. An important case is the situation where the concrete specification is given by the composition of (already available) specifications of single components. Then their

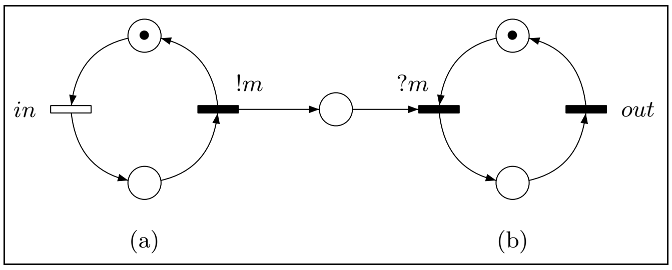


Fig. 5. Composition of the producer and consumer Petri nets

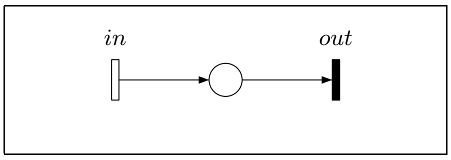


Fig. 6. Requirements specification for an infinite state producer/consumer system

composition must exhibit the required observable behaviour of a given abstract specification.

As an example we consider the requirements specification for an infinite state producer/consumer system presented by the modal I/O-Petri net in Fig. 6. Obviously, the asynchronous composition of the single producer and consumer nets in Fig. 5 is not (yet) a correct refinement since there are still the internal labels which do not correspond to silent transitions (yet) and therefore must be taken into account when comparing the two generated modal language specifications. However, since internal lables describe internal actions which are invisible from the outside, we can apply an abstraction to the composition which relabels all internal actions to the empty word ϵ. In the example the internal label are just the labels $!m$ and $?m$ used for the communication but, in general, transitions with internal labels may also describe internal computation steps of an implementation and then its is also meaningful to abstract them away. Hence we define a general abstraction operator which can be applied to any (modal) I/O-Petri net.

Definition 13 (Abstraction). *Let $\mathcal{M} = (\mathcal{N}, T_\Box)$ be a modal I/O-Petri net over the I/O-alphabet $\Sigma = in \uplus out \uplus int$ with underlying Petri net $\mathcal{N} = (P, T, W^-, W^+, \lambda, m_0)$. Let $\alpha(\Sigma) = in \uplus out \uplus \emptyset$ and let $\alpha : \Sigma \cup \{\epsilon\} \to \alpha(\Sigma) \cup \{\epsilon\}$ be the relabeling defined by $\alpha(a) = a$ if $a \in in \uplus out$, $\alpha(a) = \epsilon$ otherwise. Then the* abstraction *from $\mathcal{M}$ is the modal I/O-Petri net $\alpha(\mathcal{M}) = (\alpha(\mathcal{N}), T_\Box)$ over the I/O-alphabet $\alpha(\Sigma)$ with underlying Petri net $\alpha(\mathcal{N}) = (P, T, W^-, W^+, \alpha \circ \lambda, m_0)$.*

Coming back to our example the "abstract" Petri net in Fig. 6 is obviously modally weakly deterministic. For the abstraction of the composed Petri nets in Fig. 5 this is not obvious but, according to the results of the next section, we can decide it (and get a positive answer). Note, however, that in the case where one of the transitions used for the communication would not be a "must" the abstraction of the Petri net composition would not satisy the second condition of modal weak determinacy. The next problem is to decide whether the refinement relation holds between their generated modal language specifications. Again, according to the results of the next section, we can decide this (and also get a positive answer).

5 Decision Algorithms

We begin this section by some recalls about semi-linear sets and decision procedures in Petri nets.

Let $E \subseteq \mathbb{N}^k$, E is a *linear set* if there exists a finite set of vectors of $\mathbb{N}^k$ $\{v_0, \dots, v_n\}$ such that $E = \{v_0 + \sum_{1 \leq i \leq n} \lambda_i v_i \mid \forall i \; \lambda_i \in \mathbb{N}\}$. A *semi-linear set* is a finite union of linear sets; a representation of it is given by the family of finite sets of vectors defining the corresponding linear sets. Semi-linear sets are *effectively* closed by union, intersection and complementation. This means that one can compute a representation of the union, intersection and complementation starting from a representation of the original semi-linear sets. E is an *upward closed set* if $\forall v \in E \; v' \geq v \Rightarrow v' \in E$. An upward closed set has a finite set of minimal vectors denoted $\min(E)$. An upward closed set is a semi-linear set which has a representation that can be derived from the equation $E = \min(E) + \mathbb{N}^k$ if $\min(E)$ is computable.

Given a Petri net $\mathcal{N}$ and a marking m, the reachability problem consists in deciding whether m is reachable from m_0 in $\mathcal{N}$. This problem is decidable [12]. Furthermore this procedure can be adapted to semi-linear sets. Given a semi-linear set E of markings, in order to decide whether there exists a marking of E which is reachable, we proceed as follows. For any linear set $E' = \{v_0 + \sum_{1 \leq i \leq n} \lambda_i v_i \mid \forall i \; \lambda_i \in \mathbb{N}\}$ associated with E we build a net $\mathcal{N}_{E'}$ by adding transitions $t_1, \dots, t_n$. Transition t_i has v_i as input vector and the null vector as output vector. Then one checks whether v_0 is reachable in $\mathcal{N}_{E'}$. E is reachable from m_0 iff one of these tests is positive.

In [18] given a Petri net, several procedures have been designed to compute the minimal set of markings of several interesting upward closed sets. In particular, given a transition t, the set of markings m from which there exists a transition sequence σ with $m[\sigma t\rangle$ is effectively computable.

Now we solve the decision problems stated in the previous sections.

Proposition 2. *Let $\mathcal{N}$ be a labeled Petri net, then it is decidable whether $\mathcal{N}$ is weakly deterministic.*

Proof. First we build a net $\mathcal{N}'$ defined as follows.

– Its set of places is the union of two disjoint copies P_1 and P_2 of P.
– There is one transition (t,t') for every t and t' s.t. $\lambda(t) = \lambda(t') \neq \varepsilon$. The input (resp. output) vector of this transition is the one of t with P substituted by P_1 plus the one of t' with P substituted by P_2.
– There are two transitions t_1, t_2 for every t s.t. $\lambda(t) = \varepsilon$. The input (resp. output) vector of t_1 (resp t_2) is the one of t with P substituted by P_1 (resp. P_2).
– The initial marking is m_0 with P substituted by P_1 plus m_0 with P substituted by P_2.

Then for every $a \in \Sigma$, we compute a representation of the set E_a from which, in $\mathcal{N}$ a transition labelled by a is eventually fireable after the firing of silent transitions (using results of [18]) and a representation of its complementary set $\overline{E_a}$. Afterwards we compute the representation of the semi-linear set F_a whose projection on P_1 is a vector of E_a with P substituted by P_1 and whose projection on P_2 is a vector of $\overline{E_a}$ with P substituted by P_2. Let $F = \bigcup_{a \in \Sigma} F_a$ then $\mathcal{N}$ is weakly deterministic iff F is not reachable which is decidable.

Proposition 3. *Let $\mathcal{N}$ be a weakly deterministic labeled Petri net and $\mathcal{N}'$ be a labeled Petri net then it is decidable whether $\mathcal{L}(\mathcal{N}) \subseteq \mathcal{L}'(\mathcal{N}')$.*

Proof. W.l.o.g. we assume that P and P' are disjoint. First we build a net $\mathcal{N}''$ defined as follows.

– Its set of places is the union of P and P'.
– There is one transition (t,t') for every $t \in T$ and $t' \in T'$ s.t. $\lambda(t) = \lambda(t') \neq \varepsilon$. The input (resp. output) vector of this transition is the one of t plus the one of t'.
– Every transition $t \in T \cup T'$ s.t. $\lambda(t) = \varepsilon$ is a transtion of $\mathcal{N}''$.
– The initial marking is the $m_0 + m'_0$.

Then for every $a \in \Sigma$, we compute a representation of the set $E_{\mathcal{N},a}$ (resp. $E_{\mathcal{N}',a}$) from which in $\mathcal{N}$ a transition labelled by a is eventually fireable preceeded only by silent transitions and a representation of its complementary set $\overline{E_{\mathcal{N},a}}$ (resp. $\overline{E_{\mathcal{N}',a}}$). Afterwards we compute the representation of the semi-linear set F_a whose projection on P is a vector of $\overline{E_{\mathcal{N},a}}$ and whose projection on P' is a vector of $E_{\mathcal{N}',a}$. Let $F = \bigcup_{a \in \Sigma} F_a$ then $\mathcal{L}(\mathcal{N}) \subseteq \mathcal{L}'(\mathcal{N}')$ iff F is not reachable. This procedure is sound. Indeed assume that some marking $(m,m') \in F_a$ is reachable in $\mathcal{N}''$ witnessing that after some word w, some firing sequences $\sigma \in \mathcal{N}, \sigma' \in \mathcal{N}'$ s.t. $m_0[\sigma\rangle m$, $m'_0[\sigma'\rangle m'$ and $\lambda(\sigma) = \lambda'(\sigma')$ from m one cannot "observe" a and from m' one can "observe" a. Then due to weak determinism of $\mathcal{N}$ for every m^* s.t. there exists a sequence σ^* with $m_0[\sigma^*\rangle m^*$ and $\lambda(\sigma^*) = \lambda(\sigma)$, m^* is also in $\overline{E_{\mathcal{N},a}}$.

Proposition 4. *Let $\mathcal{M}$ be a modal Petri net, then it is decidable whether $\mathcal{M}$ is (modally) weakly deterministic.*

Proof. Observe that the first condition for being weakly deterministic is decidable by proposition 2. In order to decide the second condition, we build as in the corresponding proof the net $\mathcal{N}'$. Then we build representations for the following semi-linear sets. G_a is the set of markings m of $\mathcal{N}$ such that from m a transition of $T_\Box$ labelled by a is eventually fireable after firing silent transitions of $T_\Box$. Afterwards we compute the representation of the semi-linear set H_a whose projection on P_1 is a vector of G_a with P substituted by P_1 and whose projection on P_2 is a vector of $\overline{G_a}$ with P substituted by P_2. Let $H = \bigcup_{a\in\Sigma} H_a$ then $\mathcal{M}$ fulfills the second condition of weak determinism iff H is not reachable.

Proposition 5. *Let $\mathcal{M}, \mathcal{M}'$ be two weakly deterministic modal Petri nets then it is decidable whether the modal specification $\mathcal{S}(\mathcal{M})$ refines $\mathcal{S}(\mathcal{M}')$.*

Proof. Observe that the first condition for refinement is decidable by proposition 3. In order to decide the second condition, we build as in the corresponding proof the net $\mathcal{N}''$. Then we build representations for the semi-linear sets G_a (as in the previous proof) and similarly G'_a in the case of $\mathcal{N}'$. Afterwards we compute the representation of the semi-linear set H_a whose projection on P is a vector of G_a and whose projection on P' is a vector of $\overline{G'_a}$. Let $H = \bigcup_{a\in\Sigma} H_a$ then the second condition for refinement holds iff H is not reachable. This procedure is sound. Indeed assume that some marking $(m, m') \in H_a$ is reachable in $\mathcal{N}''$ witnessing that after some word w, some firing sequences $\sigma \in \mathcal{N}, \sigma' \in \mathcal{N}'$ s.t. $m_0[\sigma\rangle m$, $m'_0[\sigma'\rangle m'$ and $\lambda(\sigma) = \lambda'(\sigma') = w$ and from m one can "observe" b by a "must" sequence and from m' one cannot observe a by a must sequence. Then due to (the second condition of) weak determinism of $\mathcal{N}'$ for every m^* s.t. there exists a sequence σ^* with $m'_0[\sigma^*\rangle m^*$ and $\lambda'(\sigma^*) = \lambda'(\sigma')$, m^* is also in $\overline{G'_a}$.

6 Conclusion

In the present work, we have introduced modal I/O-Petri nets and we have provided decision procedures to decide whether such Petri nets are weakly deterministic and whether two modal language specifications generated by weakly deterministic modal Petri nets are related by the modal refinement relation. An important role has been played by the notion of modal weak determinacy and by the abstraction operator which considers internal transitions to be silent. Since, in general, the abstraction operator does not preserve modal weak determinacy, we are interested in the investigation of conditions which ensure this preservation property. This concerns also conditions for single components such that the abstraction of their composition is modally weakly deterministic. Another direction of future research concerns the study of compatibility of component behaviours represented by modal I/O-Petri nets and the establishment of an interface theory for this framework along the lines of [3].

References

1. A. Antonik, M. Huth, K. G. Larsen, U. Nyman and A. Wasowski. Complexity of decision problems for mixed and modal specifications. In *Proc. of the 10th Int.*

Conf. on Found. of Software Science and Comp. Struct. (FoSSaCS'08), vol. 4962 of *LNCS*, Springer, 2008.

2. A. Antonik, M. Huth, K.G. Larsen, U. Nyman and A. Wasowski. EXPTIME-complete decision problems for mixed and modal specifications. In: *Proc. of EXPRESS*, July 2008.
3. S. Bauer and P. Mayer and A. Schroeder and R. Hennicker. On Weak Modal Compatibility, Refinement, and the MIO Workbench. In Proc. 16^{th} Int. Conf. Tools and Algor. for the Constr. and Analysis of Systems (TACAS'10), 2010
4. M.H.T. Hack. Decidability questions for Petri Nets. Ph.D.Thesis. M.I.T (1976)
5. R. Hennicker and S. Janisch and A. Knapp. On the Observable Behaviour of Composite Components. In Electr. Notes Theor. Comput. Sci. Volume 260, 2010, pages 125-153. http://dx.doi.org/10.1016/j.entcs.2009.12.035, DBLP, http://dblp.uni-trier.de
6. R.M. Karp, R.E. Miller Parallel program schemata. In: *JTSS* 4, 1969, pp 147-195.
7. V. Khomenko, M. Schaefer and W. Vogler. Output-Determinacy and Asynchronous Circuit Synthesis. In: Fundam. Inf. 88, 4, pages 541-579 (Dec. 2008)
8. K.G. Larsen. Modal specifications. In: *Joseph Sifakis, editor, Automatic Verification Methods for Finite State Systems*, volume 407 of LNCS, pages 232–246, 1989.
9. K.G. Larsen, U. Nyman and A. Wasowski. On modal refinement and consistency. In *Proc. of the 18th International Conference on Concurrency Theory, (CONCUR07)*, LNCS pages 105–119, Springer Verlag, 2007.
10. K.G. Larsen, U. Nyman and A. Wasowski. Modal I/O automata for interface and product line theories. In *Programming Languages and Systems, 16th European Symposium on Programming (ESOP07)*, vol. 4421 of *LNCS*, pages 64–79. Springer, 2007.
11. K.G. Larsen and b. Thomsen. A modal process logic. In:*Third Annual IEEE Symposium on Logic in Computer Science LICS*, pages 203–210, 1988.
12. E. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. of the 13th Annual ACM Symposium on Theory of Computing (STOC'81)* pages 238–246, 1981.
13. R. Milner Communication and concurrency. Prentice Hall, 1989.
14. E. Pelz. Closure properties of deterministic Petri net languages. In *STACS'87*, vol. 247 of *LNCS*, Springer 1987.
15. J.L. Peterson, Petri net theory and the modeling of systems, Prentice-Hall, Englewood Cliffs, NJ, 1981
16. J.-B. Raclet. Residual for Component Specifications. In: *Proc. of the 4th International Workshop on Formal Aspects of Component Software (FACS07), Sophia-Antipolis, France*, September 2007.
17. J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay and R. Passerone. Modal Interfaces: Unifying Interface Automata and Modal Specifications. In *Proc. of 9th International Conference on Embedded Software (EMSOFT'09), Grenoble, France, ACM*, October 2009.
18. R. Valk, M. Jantzen The residue of vector sets with applications to decidability problems in Petri nets Advances in Petri Nets 1984, LNCS volume 188 pp. 234-258

Generating Benchmarks by Random Stepwise Refinement of Petri Nets

Kees M. van Hee and Zheng Liu

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{k.m.v.hee, z.liu3}@tue.nl

Abstract. The quality of algorithms is often determined by benchmarking, i.e., testing the algorithm on a predetermined data set. In contrast to traditional benchmarking, with fixed data set, we present a way to generate random sets of test data. In this paper we present random classes of Petri nets and a method to generate finite samples from such a class. The classes may contain infinitely many Petri nets, each net with its own probability to be generated. This generation method is based on stepwise application of construction rules such as refinement rules. Each random class of Petri nets has a probability distribution for each of its characteristics. We illustrate the approach by estimating this distribution for some simple characteristics.
Keywords: Petri nets, benchmarking, random graphs

1 Introduction

In computer science research we often lack a method to evaluate the *quality* of an algorithm in an analytical way. The quality may concern the *efficiency* of the algorithm (how fast is a solution found) or the *effectiveness* (how good is the solution found). Although it is sometimes possible to give bounds for the worst case behavior, it is seldom possible to determine the average-case behavior analytically. What we normally do is fixing a set of test data as *benchmark* and then we test the algorithm on that set, which is called *benchmarking*. Generally a benchmark is either pre-defined or generated. Both have disadvantages [15]. Pre-defined datasets are designed to be representative examples in a particular domain. Researchers evaluate new algorithms with respect to such a fixed benchmark, but how good are they if applied to other test data? In order to increase the quality of benchmarking we will consider methods to generate *random benchmarks*, which are *samples* from an infinite set of tests, each having its own probability of being selected for a sample. Note that it is impossible to have an infinite set of tests each having the same probability, so the probabilities of the tests have to be non-uniform. Due to the fast increasing computing power we are able to generate samples that are so big that all non-selected tests together have a probability below some chosen bound. This allows us to obtain *statistical* statements of the quality of an algorithm.

In this paper we focus on Petri nets and algorithms to compute their characteristic properties. Hence we define *random classes* of Petri nets. Such a class contains Petri nets with some structural property, like free choice nets. There may be infinite number of nets in one class. Each net in a class has its own probability of being selected for a benchmark. Petri nets are used to model complex processes [11], for example computational processes in a computer system or business processes within or between enterprises. These models are often produced by a stepwise refinement process (top-down approach) or by gluing together existing components (bottom-up approach). The *generation method* uses two kinds of construction rules, *refinement rules* and *bridge rules.* The refinement rules were firstly studied by Berthelot in [5] and Murata in [18] as reduction rules, in this paper we use them in the inverse direction to expand Petri nets by refinement. The bridge rules connect a pair of nodes in a Petri net, so we can use them to glue components. In most cases these construction rules preserve some property which means that if the initial Petri net has that property, then all elements of the class have the same property. The generation method determines, in a random way, (1) which construction rule will be applied in the current Petri net, (2) to which part of the net and (3)if we continue or stop. So the construction rules have weights. Rephrasing what we are doing, we define a *graph grammar* with weights on the *construction rules*, such that each graph of the graph language has a certain probability of occurring.

Although our generation method for benchmarks can be applied to all kinds of algorithms on Petri nets, we use it here to determine *characteristics* of the random classes of Petri nets. Simple examples of such characteristics are the number of nodes, and the average fanin and fanout of nodes. More complicated ones are the occurrence (or the number) of deadlocks or livelocks. Each characteristic of a Petri net is expressed by a real number. In some cases we consider only 0 and 1 and we consider them as 'false' and 'true'. Since every Petri net in a class has a certain probability of being selected in a benchmark, we may consider every characteristic as a random variable having a *probability distribution* over the class. For the given examples, we can speak of the *expectation* and *variance* of the number of nodes in a class, or the *probability* of having a deadlock. We have developed a software tool in the form of a plugin for ProM (cf [16]) to realize our approach.

The rest of this paper is organized as follows. Section 2 introduces the necessary preliminaries. In Section 3 we represent our construction rules and discuss our methodology of generating Petri nets. Section 4 presents some characteristics. The software tool is introduced in Section 5 with characteristic examples. Related work is discussed in Section 6. Finally Section 7 concludes this paper and discusses some of our future researches on identifying class parameters.

2 Preliminaries

Let S be a set. With $|S|$ we denote the number of elements in S. The empty set, e.g., the set without any elements is denoted by $\emptyset$. Two sets S and R are disjoint if $S \cap R = \emptyset$. We denote the set of all natural numbers as $\mathbb{N} = \{0, 1, 2, \cdots\}$. A sequence σ of length $l \in \mathbb{N}$ over S is a function $\sigma : \{1, \cdots, l\} \to S$. We denote a sequence by $\sigma =< \sigma(1), \sigma(2), \cdots, \sigma(l) >$, such that $\forall i(1 \leq i < l) : \sigma(i) \in \bullet\sigma(i+1)$. We write here this as $n_1 \xrightarrow{\sigma} n_l$. We denote the length of a sequence by $|\sigma|$. The set of all finite sequences over S is denoted as Σ. Let $\nu, \gamma \in \Sigma$ be two sequences. *Concatenation*, denoted by $\sigma = \nu \circ \gamma$, is defined as $\sigma : \{1, \cdots, |\nu| + |\gamma|\} \to S$, such that for $1 \leq i \leq |\nu| : \sigma(i) = \nu(i)$, and for $|\nu| + 1 \leq i \leq |\nu| + |\gamma| : \sigma(i) = \gamma(i - |\nu|)$. The Parikh vector of a sequence σ, denoted by $\overrightarrow{\sigma}$, is a bag representing the number of occurrences of each element in σ. A *bag m (multiset)* over S is a function $m : S \to \mathbb{N}$. For $s \in S$, $m(s)$ denotes the number of occurrences of s in m. We denote a bay by square brackets. e.g., in a bag $[a, b^2, c]$, element a occurs once, element b twice, and element c once. All other elements have a multiplicity of 0. $\prec$ is the prefix operator, such that $\sigma' \prec \sigma$ if and only if $\exists \sigma'' : \sigma = \sigma_{'} \circ \sigma_{''}$.

A Petri net is a tuple $N = (P, T, F)$ where P is the set of *places*, T is the set of *transitions*, P and T are disjoint, and $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs. An element of $P \cup T$ is called a node. We call an element of $P \cup T \cup F$ is an element of N. A *path* in N is a sequence σ over the set $P \cup T$. Graphically, we denote places by circles, transitions by squares, and arcs as arrows between places and transitions. The *state* of a Petri net, called a *marking* is a bag over the places P of N. A marking is graphically represented by placing *tokens* in each place. A marked Petri net is a pair (N, m_0), where N is a Petri net and m_0 is a marking of N. A transition $t \in T$ is enabled in (N, m_0), denoted by $(N : m_0 \xrightarrow{t})$ if $\bullet t \leq m_0$. An enabled transition in (N, m_0) can *fire* resulting in a new marking $m' = m_0 - \bullet t + t\bullet$, denoted by $(N : m_0 \xrightarrow{t} m')$.

A special class of Petri nets are *workflow nets.* A workflow net is a 5-tuple $W = (P, T, F, i, f)$ where (P, T, F) is a Petri net, $i \in P$ is the initial place, such that $\bullet i = \emptyset$, $f \in P$ is the final place, such that $f\bullet = \emptyset$, in graph of W each node $n \in P \cup T$ is on a directed path from i to f. If for a workflow net W, we have $\forall p \in P \setminus \{i, f\}, |p\bullet| = |\bullet p| = 1, |i\bullet| = |\bullet f| = 1$, the workflow net is a *T-net*, also called a *marked graph* workflow net. If $\forall t \in T, |t\bullet| = |\bullet t| = 1$, then it is a *S-net*, also called a *state machine* workflow net. In a workflow net, two places $p, s \in P$, if either $p\bullet = s\bullet$ or $p\bullet \cap s\bullet = \emptyset$, then this workflow net is a *free-choice* workflow net. A *firing sequence* or a *trace* is a sequence σ over T such that all the transitions of σ can fire in that order starting from the initial marking. A trace for a workflow net is complete if it leads to the final marking with only f marked.

A workflow net is k sound, for $k \in \mathbb{N}$ if for each marking m that is reachable from an initial marking m_0 with only k tokens in the initial place i, the final marking with only k tokens in the final place f can be reached. A workflow net is *generalized* sound if it is k-sound for all $k \geq 1$ ([14]). Note that *1-sound* is usually called *sound* ([1]). Soundness can be considered as a general sanity check for workflow nets.

3 Net Generation

In this section we firstly define the construction rules. The rules enable us to generate all Petri nets, here we focus upon workflow nets. Moreover, we show that different subclasses of workflow nets can be generated by different construction rules. Finally, we discuss our method of randomly generating Petri nets.

3.1 Construction Rules

Based upon how they can modify the structure of Petri nets, the construction rules are divided into two classes, *refinement* rules and *bridge* rules. The refinement rules ([5], [8], [12], and [18]) were firstly studied by Berthelot and Murata as abstraction rules to reduce Petri nets, here we use them in the inverse direction to expand Petri nets. The bridge rules connect a pair of nodes in Petri nets, so we can use them to glue components. Let N be the original net and R be one of the construction rules. If we apply R to N then we get the generated net $N^{'}$. If we use R in the opposite direction, then we can get N again by reducing $N^{'}$. In such a case we say $(N, N^{'}) \in \varphi_R$. Based upon such a relationship, we define the rules as follows.

Refinement rules. Figure 1 is an example of the refinement rules.

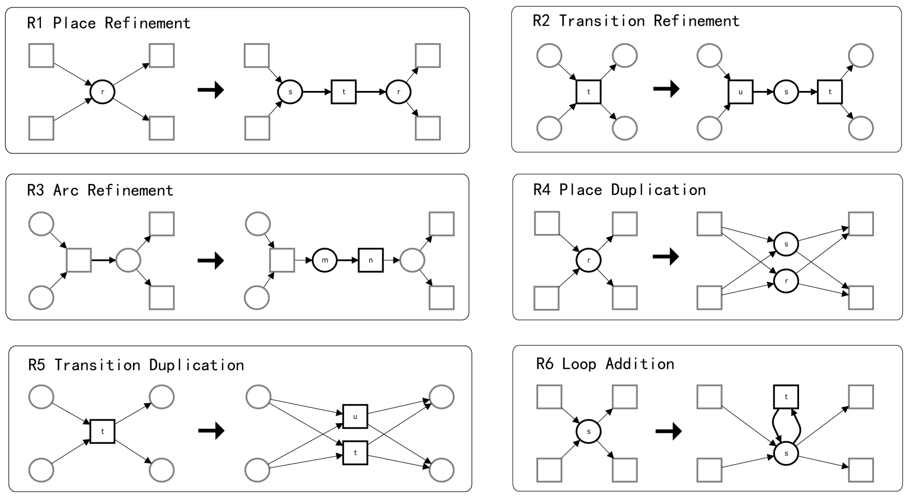


Fig. 1. Refinement Rules

Definition 1 (Place refinement rule R1). *Let* $N = (P, T, F)$ *and* $N^{'} = (P^{'}, T^{'}, F^{'})$ *be two Petri nets. We say* $(N, N^{'}) \in \varphi_{R_1}$ *if and only if there exist places* $s, r \in P^{'}, s \neq r$ *and a transition* $t \in T^{'}$ *such that:* $\bullet t = \{s\}, t\bullet = \{r\}$, $s\bullet = \{t\}, \bullet s \neq \emptyset$, $\bullet s \not\subseteq \bullet r$. *The net* N *satisfies:* $P = P^{'} \setminus \{s\}$, $T = T^{'} \setminus \{t\}$, $F = (F^{'} \cap ((P \times T) \cup (T \times P))) \cup (\bullet s \times t\bullet)$.

Definition 2 (Transition refinement rule R2). *Let $N = (P, T, F)$ and $N^{'} = (P^{'}, T^{'}, F^{'})$ be two Petri nets. We say $(N, N^{'}) \in \varphi_{R_2}$ if and only if there exist a place $s \in P^{'}$ and transitions $t, u \in T^{'}, t \neq u$ such that: $\bullet s = \{u\}, s\bullet = \{t\}$, $\bullet t = \{s\}, t\bullet \neq \emptyset$, $u\bullet \not\subseteq t\bullet$. The net N satisfies: $P = P^{'} \setminus \{s\}$, $T = T^{'} \setminus \{u\}$, $F = (F^{'} \cap ((P \times T) \cup (T \times P))) \cup (\bullet u \times s\bullet)$.*

Definition 3 (Arc refinement rule R3). *Let $N = (P, T, F)$ and $N^{'} = (P^{'}, T^{'}, F^{'})$ be two Petri nets. We say $(N, N^{'}) \in \varphi_{R_3}$ if and only if there exist two nodes $m, n \in P^{'} \cup T^{'}$, such that: $| \bullet m| = 1$, $m\bullet = \{n\}$, $|n \bullet | = 1$, $\bullet n = \{m\}$, $(\bullet m \times n\bullet) \cap F^{'} = \emptyset$. The net N satisfies: $P \cup T = (P^{'} \cup T^{'}) \setminus \{m, n\}$, $F = (F^{'} \cap ((P \times T) \cup (T \times P))) \cup (\bullet m \times n\bullet)$.*

Note that in Figure 1, R3 shows only one instance of the arc refinement rule. We can also refine any arc with a place as its source node and a transition as its target node. This case is not shown in Fig 1.

Definition 4 (Place duplication rule R4). *Let $N = (P, T, F)$ and $N^{'} = (P^{'}, T^{'}, F^{'})$ be two Petri nets. We say $(N, N^{'}) \in \varphi_{R_4}$ if and only if there exist two places $s, r \in P^{'}, s \neq r$ such that: $\bullet s = \bullet r$, $s\bullet = r\bullet$. The net N satisfies: $P = P^{'} \setminus \{s\}$, $T = T^{'}$, $F = F^{'} \cap ((P \times T) \cup (T \times P))$.*

Definition 5 (Transition duplication rule R5). *Let $N = (P, T, F)$ and $N^{'} = (P^{'}, T^{'}, F^{'})$ be two Petri nets. We say $(N, N^{'}) \in \varphi_{R_5}$ if and only if there exist two transitions $t, u \in T^{'}, t \neq u$ such that: $\bullet t = \bullet u$, $t\bullet = u\bullet$. The net $P = P^{'}$, $T = T^{'} \setminus \{u\}$, $F = F^{'} \cap ((P \times T) \cup (T \times P))$.*

Definition 6 (Loop addition rule R6). *Let $N = (P, T, F)$ and $N^{'} = (P^{'}, T^{'}, F^{'})$ be two Petri nets. We say $(N, N^{'}) \in \varphi_{R_6}$ if and only if there exists a place $s \in P^{'}$ and a transition $t \in T^{'}$ such that: $\bullet t = \{s\}$, $t\bullet = \{s\}$. The net N satisfies: $P = P^{'}$, $T = T^{'} \setminus \{t\}$, $F = F^{'} \cap ((P \times T) \cup (T \times P))$.*

Bridge rules. Figure 2 is an example of the bridge rules.

Definition 7 (Place bridge rule R7). *Let $N = (P, T, F)$ and $N^{'} = (P^{'}, T^{'}, F^{'})$ be two Petri nets. We say $(N, N^{'}) \in \varphi_{R_7}$ if and only if there exist one place $s \in P^{'}$ and two transitions $u, t \in T^{'}$ such that: $\bullet s = \{u\}$, $s\bullet = \{t\}$. The net N satisfies: $P = P^{'} \setminus \{s\}$, $T = T^{'}$, $F = F^{'} \cap ((P \times T) \cup (T \times P))$.*

Definition 8 (Transition bridge rule R8). *Let $N = (P, T, F)$ and $N^{'} = (P^{'}, T^{'}, F^{'})$ be two Petri nets. We say $(N, N^{'}) \in \varphi_{R_8}$ if and only if there exist one transition $t \in T^{'}$ and two places $s, r \in P^{'}$ such that: $\bullet t = \{s\}$, $t\bullet = \{r\}$. The net N satisfies: $P = P^{'}$, $T = T^{'} \setminus \{t\}$, $F = F^{'} \cap ((P \times T) \cup (T \times P))$.*

Definition 9 (Arc bridge rule R9). *Let $N = (P, T, F)$ and $N^{'} = (P^{'}, T^{'}, F^{'})$ be two Petri nets. We say $(N, N^{'}) \in \varphi_{R_9}$ if and only if there exist two nodes $s, r \in P^{'} \cup T^{'}$, such that $(s, r) \in F^{'}$. The net N satisfies: $P = P^{'}$, $T = T^{'}$, $F = F^{'} \setminus \{(s, r)\}$.*

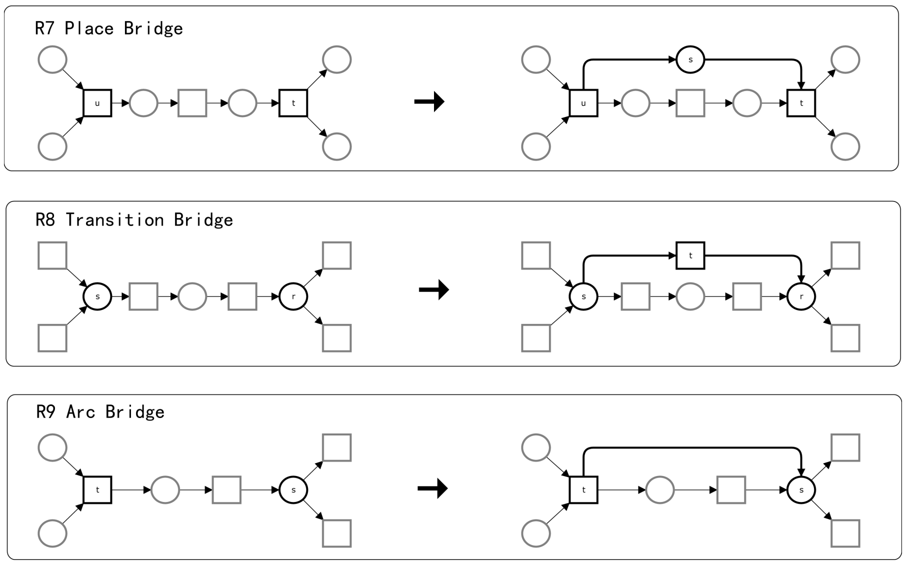


Fig. 2. Bridge Rules

Note that in Figure 2, R9 merely shows one instance of the arc bridge rule. We can also bridge a place and a transition with an arc. This case is not shown in Figure 2.

In R8, if s and r are the same place, it becomes R6. Thus the loop addition rule R6 is a special case of the transition bridge rule R8. All the refinement rules (R1,...,R6) preserve *liveness* and *boundedness* properties of Petri nets (with respect to a given marking) (proofs can be found in [12]). Although without formal proof, it is still easy to observe that all the bridge rules (R7,...,R9) generally do not preserve those properties. In fact, the bridge rules can break any good structures. For instance, the transition bridge rule can model the *goto* structure in programming languages, and the danger of such a structure is discussed in [9].

3.2 Structural Classes of Generated Nets

In this part we study different classes of the generated Petri nets based on their structures. We focus on workflow nets and their subclasses, namely, *Jackson nets*, *state machine* workflow nets, *marked graph* workflow nets, and *free-choice* workflow nets. These workflow nets without Jackson nets are defined in Section 2, so firstly we define a Jackson net as follows. For a formal description and analysis of Jackson nets see [12].

Definition 10 (Jackson net). *A Jackson net is a workflow net that can be generated, from one single place, by applying the rules* $R1$*,* $R2$*,* $R4$*,* $R5$*, and* $R6$ *recursively. However, only the rule* $R1$ *can be applied in the first step, and no rules can be applied to the initial place and the final place after the first step.*

In order to generate all workflow nets (starting with two places), it is sufficient to use the arc refinement rule R3 and the bridge rules R7, R8, R9 only. To prove that, we give the following definition.

Definition 11 (Copy of a net). *Let N be a workflow net. $N^{'}$ is a copy of N if and only if there is a bijection φ such that $\varphi(P) = P^{'}$, $\varphi(T) = T^{'}$, $\varphi(F) = F^{'}$ and $\forall (x, y) \in F : \varphi((x, y)) = (\varphi(x), \varphi(y))$.*

A path taken from a Petri net defines a new Petri net, so copying a path can be treated as copying its corresponding Petri net.

Lemma 1. *Given two nodes, m_1, $m_2 \in P \cup T$, and a path σ, such that $m_1 = \sigma(1)$, $m_2 = \sigma(|\sigma|)$, and $\forall i \in dom(\sigma) : \overrightarrow{\sigma}(i) = 1$. Then we can copy σ by firstly applying a bridge rule on (m_1, m_2) and afterwards repeating the arc refinement rule.*

Lemma 2. *We have a path σ in a Petri net, if there is some node n such that $\overrightarrow{\sigma}(n) > 1$, then there is a path $\sigma^{'} = \sigma_1 \circ \sigma_3$ if and only if $\sigma = \sigma_1 \circ \sigma_2 \circ \sigma_3$ and $n \xrightarrow{\sigma_2} n$.*

Theorem 1. *Let N be a workflow net. Then N can be copied into $N^{'}$ by only using the bridge rules and the arc refinement rule, starting with only two places.*

Proof. Initially $P^{'} = \{i^{'}, f^{'}\}$, $T^{'} = \emptyset$, $F^{'} = \emptyset$, $i^{'} = \varphi(i)$, and $f^{'} = \varphi(f)$. We select an arbitrary node n from N such that $n \in P \cup T$ and $n \notin P^{'} \cup T^{'}$. In N, from n we find two paths, $n_1 \xrightarrow{\sigma_1} n$ and $n \xrightarrow{\sigma_2} n_2$, such that $n_1, n_2 \in P^{'} \cup T^{'}$ and all other nodes on σ_1 and σ_2 are not in $P^{'} \cup T^{'}$. This is always possible by the definition of a workflow net. By Lemma 2 we can reduce σ_1 and σ_2 by removing all internal loops along the paths.

Consider case (1): suppose σ_1 and σ_2 have no common internal nodes, then we may apply Lemma 1 by adding the path $\sigma_1 \circ \sigma_2$ from n_1 to n_2, and add $n^{'} = \varphi(n)$ to the copy.

Consider case (2): suppose σ_1 and σ_2 have common internal nodes, e.g., $\exists i, j : \sigma_1(i) = \sigma_2(j) \neq n$, then there is a loop. We then look for the first node m on σ_1 which is also on σ_2. Then $\exists \sigma_3, \sigma_4, \sigma_5, \sigma_6 : n_1 \xrightarrow{\sigma_3} m \xrightarrow{\sigma_4} n \wedge n \xrightarrow{\sigma_5} m \xrightarrow{\sigma_6} n_2$ where $\sigma_1 = \sigma_3 \circ \sigma_4$ and $\sigma_2 = \sigma_5 \circ \sigma_6$. Now we have found $n_1 \xrightarrow{\sigma_3} m$ and $m \xrightarrow{\sigma_6} n_2$, and σ_3 and σ_6 have no internal nodes in common as m is the first common node on σ_1. We replace n with m, then we apply Lemma 1 by adding the path $\sigma_3 \circ \sigma_6$ from n_1 to n_2, and we add node $m^{'} = \varphi(m)$ to the copy.

In each step, therefore, we add at least one node. We repeat this procedure until all the nodes of N are copied into $N^{'}$. Finally, we add all arcs $F \setminus \varphi^{-1}(F^{'})$ by arc bridge rule. Now $N^{'}$ is a copy of N. □

Now let us consider which rules allow us to generate the subclasses of workflow nets. Based upon Theorem 1, it is easy to prove that we only need the transition bridge rule and the arc refinement rule to generate all state machine workflow nets, and we need the place bridge rule and the arc refinement rule to generate all marked graph workflow nets. They yield Corollary 1 and Corollary 2. As defined in Definition 10, Jackson nets are generated by the rules $R1$, $R2$, $R4$, $R5$, and $R6$. For a free-choice workflow net, the rules R1, R2, R4, and R5 can always preserve its properties.

Corollary 1. *If N is a state machine workflow net, then N can be copied into $N^{'}$ by only using the transition bridge rule with the arc refinement rule.*

Corollary 2. *If N is a marked graph workflow net, then N can be copied into $N^{'}$ by only using the place bridge rule with the arc refinement rule.*

Theorem 2. *The place refinement rule, the transition refinement rule, the place duplication rule, and the transition duplication rule preserve the free-choice workflow net property.*

[12] proves that each Jackson net is a sound net. From [6] we can derive that each generated state machine workflow net is sound as well. All free-choice workflow nets are sound. The generated marked graph workflow nets cannot be sound as the rules applied do not preserve the properties.

3.3 Generation of Nets

Our approach of generating Petri nets has two determinants, construction rules and probabilities of the rules. We have discussed the necessary rules to generate different classes of workflow nets. In this part, we focus on the probability-based rule selection.

Given two nets N and $N^{'}$, we say that N generates $N^{'}$ if and only if $N^{'}$ can be obtained from N by applying zero or more times a rule from our construction rules. To generate a workflow net we adopt a stepwise refinement technique starting with one single place. In the first step we apply the place refinement rule to generate the initial place and the final place. For any subsequent steps, we select a rule from all the rules whose conditions hold (we say those rules are *enabled*). For the initial and the final places there are some restrictions such that the place refinement rule, the place duplication rule, the loop addition rule cannot be applied to the initial and the final places, and for the transition bridge and the arc bridge rules, the initial place cannot be the target place (i.e., r in Definition 8 and 9), and the final place cannot be the source place (i.e., s in Definition 8 and 9). Figure 3 depicts an example of net refinement.

In order to select a rule from all the enabled rules, we attach a *rule weight* to each rule. A rule weight is a random number ranging from 0 (least likely) to 100 (most likely). Therefore, a rule is randomly selected based on the rule weight. Similarly, we also attach an *element weight* (also ranging from 0 to 100) to each of the elements (each of the places, transitions, and arcs) in a net. Hence each rule can randomly select an element or a pair of them based on the element weight to expand a net. Because we always start with an initial net to generate another net, for all the elements in the initial net, we give each of them a default element weight. When we apply rules, new elements are added into the net. In

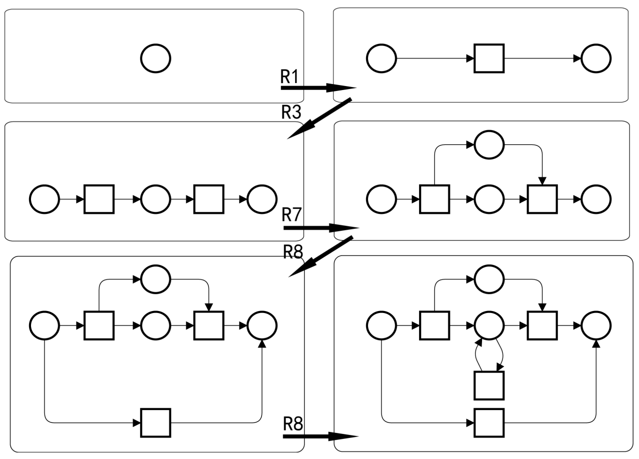


Fig. 3. An Example of Workflow Net Refinement

order to determine weights of the newly added elements, we define three *type parameters* (ranging from 0 to 1) for each element type, namely, *place parameter*, *transition parameter*, and *arc parameter*. Each rule can have some or all of the type parameters based on what kinds of elements it can introduce. For instance, as the place refinement rule can introduce two places, a transition, and two arcs, the rule has all the type parameters. While, the place bridge rule can add a place and two arcs, so it only has the place parameter and the arc parameter. All the refinement rules (R1,...,R6) only refine one element, so the weight of the newly added elements are determined by the multiplication of the weight of the refined element and the type parameters. For example, if we refine a place having a weight of 50, and we let the place parameter be 0.2, the transition parameter be 0.5, and the arc parameter be 0.1, then any newly added place has a weight of (0.2 x 50 =) 10, any newly added transition has a weight of (0.5 x 50 =) 25, and any newly added arc has a weight of (0.1 x 50 =) 5. On the other hand, all the bridge rules bridge a pair of nodes, so the weight of the newly added elements are determined by the multiplication of the sum of the bridged nodes and the type parameters. For instance, if we bridge two transitions having a weight of 50 and 70, respectively, then any newly added place has a weight of ((50 + 70) x 0.2 =) 24, and any newly added arc has a weight of ((50 + 70) x 0.1) = 12.

Therefore, such a mechanism makes net generation random. We are currently considering other mechanisms besides this one as well.

4 Characteristics of Net

Characteristics of Petri nets in different classes are decidable with the generated benchmarks. Such characteristics we consider currently consist of *length of the shortest path* (number of transitions in a path is the length of this path), *number* *of nodes*, *average number of fanin and fanout of nodes*, *deadlock and livelock*, and *coverage*. We use a real number to express each characteristic, in some cases we use 1 for *true* and 0 for *false*. Each Petri net in a class has a certain probability of being selected in a benchmark, and each characteristic can be considered as a random variable having a probability distribution over that class. For instance, with the given samples from a class, we can speak of the expectation and variance of the number of nodes in the class, or the probability of having a deadlock. Therefore, we can estimate the characteristics of any new nets from the same class with ceratin confidence. In this section, we focus on the characteristic *coverage* and give two examples to illustrate how it can help in testing.

We restrict ourselves to a Petri net with an initial node such as a workflow net. Each transition in the net is attached with a label. The goal of the coverage test is to find all the transitions with a certain label. The coverage is measured by *trace coverage* and *transition coverage*. We start a path with the initial node, at each choice point we select one of the enabled transitions at random. As soon as we find a transition with the label that we look for, we stop and start a new trace from the initial node again. Therefore, the trace coverage is the number of traces that we need to find all the transitions with the label, and the transition coverage is the number of transitions fired to find all the transitions with the label. We show two applications of how to use the coverage in testing.

Finding labels. We may attach labels to transitions in a workflow net. In this application we need to find all the transitions with a special label. We start in the initial state. A transition is randomly fired when there are two or more enabled transitions. After a transition has fired, we test whether it has the special label. If so we remove this label and we start a new trace from the initial state again. Otherwise we continue until we reach the final state and then we start from the initial state again. We stop if all the transitions with the label have been found. The coverage returns the number of traces followed and the number of transitions fired in order to find all the labeled transitions. Algorithm 1 describes this application. In this algorithm all transitions have a label either 0 or 1. We are looking for the transitions with label 1.

As labeling can be associated with different concepts, this algorithm can be used in model-based software testing as done in [7]. In this application we use a workflow net to model a software system, where each transition represents a software component. A software component either behaves correctly or has an error that can only be detected by firing the transition. Only transitions with an error are labeled.

Finding causal pairs. In this example we test how long (measured in coverage) it takes to cover all the *causal pairs*, i.e., a possible pair of consecutive transitions, in a workflow net. For process miners, i.e., the alpha algorithm [2], we need *complete logs*, i.e., log with a set of complete traces. This means that we need so many complete traces that every causal pair has occurred. We can get all the possible causal pairs for a given net by static analysis, i.e., by using the technique of reachability graph. Equipped with this result, we use Algorithm 2 to get the coverage for each given sound workflow net. This means that we have

Algorithm 1: Finding Labels

```
   input  : a sound $N = (P, T, F, i, f)$
   output: $pathCoverage, transitionCoverage$
1  var $A, B : A \subseteq T, B \subseteq T$
   var $pathCoverage, transitionCoverage : int$
   var $stop : boolean$
   var $m : P \longrightarrow \mathbb{N}$
   begin
2      $A := \emptyset; B := \emptyset; pathCoverage := 0; transitionCoverage := 0; stop :=$
       $false; m_0 := [i]$;
       while $|A| < |T|$ do
3          $pathCoverage := pathCoverage + 1; m^{'} := m_0$;
4          repeat
5              $B := \{t \in T|(N : m^{'} \xrightarrow{t})\}$;
6              $(N : m^{'} \xrightarrow{t} m^{''})$, for some $t \in B$;
7              $m^{'} := m^{''}$; $transitionCoverage := transitionCoverage + 1$;
               $A := A \cup \{t\}$;
8              if $transitionLabel(t) \neq 0$ then
9                  $transitionLabel(t) := 0$; $stop := true$;
10             endif
11         until $m^{'} = [f]$ $or\ stop = true$
12     end
13 end
```

an estimate of the length of a log in order to be complete for random classes of Petri nets.

Finally, we show the result of an empirical study we did. We used the tool we developed (see Section 5) to randomly generated 3000 well-structured workflow nets. Those nets were used as benchmarks to get the results in Table 1.

5 Tool Implementation

We realized a supporting tool for our methodology. The tool is developed as a plugin for the ProM framework [10] in Java, the distribution can be downloaded from [16]. The tool has the following relevant features:

Graphical user interface. Figure 4 is a snapshot of the tool interface.This graphical user interface is easy to use. For example, it lists all the rules visually so that it is very intuitive for the user to select a particular rule. The generated nets can be visualized using the viewer provided by ProM. Testing results are output in a table for the user.

Selection of net class. The interface contains all the classes of the workflow nets that the user can generate. Once a particular class has been selected, all the rules which are not allowed in such a class are disabled automatically by giving their probabilities a value of zero.

Algorithm 2: Finding Causal Pairs

```
input  : a sound $N = (P, T, F, i, f)$, a set $C$ of all the causal pairs in $N$
output: $pathCoverage, transitionCoverage$
1  var $A, B : A \subseteq C, B \subseteq T$
   var $pathCoverage, transitionCoverage : int$
   var $u : u \in T$
   var $m : P \longrightarrow \mathbb{N}$
   begin
2      $A := \emptyset; B := \emptyset; pathCoverage := 0; transitionCoverage := 0; u :=$
       $null; m_0 := [i];$
       while $A \neq C$ do
3          $pathCoverage := pathCoverage + 1; m^{'} := m_0;$
4          $B := \{t \in T | (N : m^{'} \xrightarrow{t})\};$
5          $(N : m^{'} \xrightarrow{t} m^{''})$, for some $t \in B$;
6          $u := t; m^{'} := m^{''}; transitionCoverage := transitionCoverage + 1;$
7          repeat
8              $B := \{t \in T | (N : m^{'} \xrightarrow{t})\};$
9              $(N : m^{'} \xrightarrow{t} m^{''})$, for some $t \in B$;
10             $transitionCoverage := transitionCoverage + 1;$
11             $A := A \cup \{(u, t)\}; u := t; m^{'} := m^{''};$
12         until $m^{'} = [f]$
13     end
14 end
```

Random weight of rules. The user can change the weight of any enabled rule using the sliding bar under the rule image, the probability of the rule is calculated and displayed next to the sliding bar.

Table 1. Mean number, standard deviation, 25th percentile (Q1), Median (Q2), 75th percentile (Q3) of the characteristics of 3000 well-structured workflow nets.

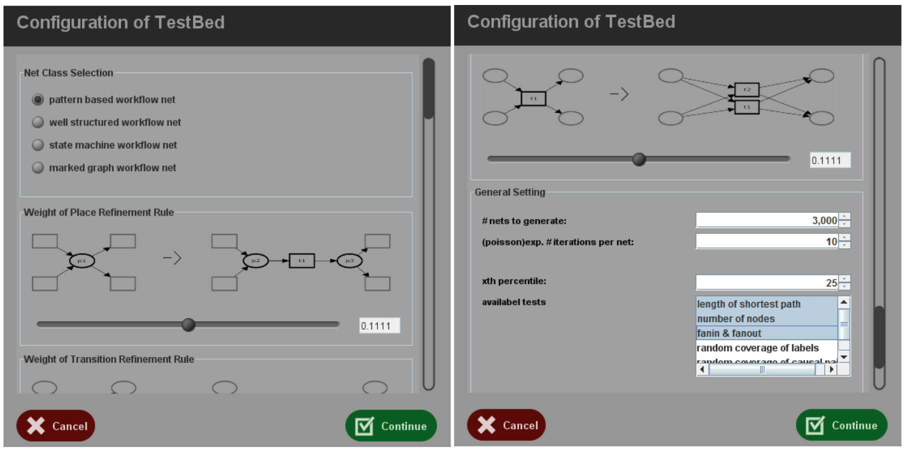
Characteristics		**Avg.**	**Std.dev.**	**Q1**	**Q2**	**Q3**
length of the shortest path		2.525	1.882	1.000	2.000	4.000
number of nodes	place	7.186	2.408	6.000	7.000	9.000
	transition	8.649	2.797	7.000	8.000	10.000
avg. fanin/out of nodes	place fanin	1.575	0.491	1.250	1.500	1.800
	place fanout	1.578	0.494	1.250	1.500	1.800
	transition fanin	1.283	0.332	1.000	1.200	1.400
	transition fanout	1.282	0.335	1.000	1.200	1.400
soundness		1.000	0.000	1.000	1.000	1.000
coverage	path coverage	15.032	10.125	8.400	12.100	18.500
	transition coverage	45.655	31.909	24.000	38.350	57.975

Specification of sample size and net size. The user can specify the number of nets to generate and the number of times to apply rules to generate a net. The sizes of all nets have a poisson distribution, the user only needs to input the average number, then the tool is able to calculate the size of each individual net.

Selection of characteristics. The user can select the characteristics introduced in Section 4 to investigate the nets.

Reusability of samples. The tool saves all the generated nets in PNML format in a dedicated folder on disk. This enables the user to reuse or share any benchmarks.

[13] proposed a systematic approach to design software in coloured Petri nets and transfer the CPN models into Java code, we practised the approach in the design and implementation of this tool. The tool currently runs on a single processor thus it is not possible to generate a very large size of benchmarks. In order to overcome this limit, we are considering to distribute the tool over multiple processors (e.g., over grid) in the future.


Configuration of TestBed

Net Class Selection

[✓ pattern based workflow net]
well structured workflow net
state machine workflow net
marked graph workflow net

Weight of Place Refinement Rule: 0.1111

Weight of Transition Refinement Rule

Configuration of TestBed

0.1111

General Setting

# nets to generate:	3,000
(poisson)exp. # iterations per net:	10
xth percentile:	25
availabel tests	length of shortest path number of nodes fanin & fanout random coverage of labels

Fig. 4. Interface Snapshot

6 Related Work

In [4] the authors proposed a method to generate Petri net benchmarks. They start with a Petri net containing two places and two transitions with all nodes connecting to each other, and make use of the refinement rules given by Murata in [18]. Whereas in our approach we use not only the Murata rules, but other rules as well, this makes our rules more extensive and more general. We focus on generating workflow nets, and start with a much simpler net with only one single place. They use a weighted random selection of rules to control the number of transitions and places. Such a probability mechanism is also realized in our approach, and we investigate more characteristics of nets besides the number of nodes.

In [3] the authors developed a tool to generate process models in Petri nets and log of business processes. They use a set of workflow patterns in [17] and add probabilities to the patterns. In our approach we do this for graph grammars. Both can get a probability distribution on the set of graphs.

7 Conclusion

In this paper, we defined a graph grammar with weights on the construction rules, such that each graph of the graph language has a ceratin probability of occurring. We consider graphs in the form of Petri nets, and the generated Petri nets can be used as benchmarks. By extensive studies, we have a set of construction rules to generate all Petri nets with different starting nets using a stepwise refinement approach. Based upon structures, the nets can be classified into different classes. Each Petri net in a class has a certain probability to be selected in a benchmark. Moreover, we distribute weights to the rules and all elements in a net, this makes the generation random. We have determined a number of characteristics used in the random classes of Petri nets, and we are able to get probability distributions of each characteristic over the classes. We focused upon an interesting characteristic called coverage, and presented two possible applications of it. A tool has been developed in ProM to realize our methodology.

In future research, we would like to identify the probability parameters based upon a concrete set of process models in a statistical way. If we have the parameter we can do better benchmarking. Also we would like to distribute our tool over grid so that we can generate a large number of benchmarks.

References

1. W. M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 2001.
2. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16:2004, 2003.
3. Process Mining Group at University of Padua. Process Log Generator. http://www.processmining.it/sw/plg, 2010.
4. G. Bergmann, A. Horvath, I. Rath, and D. Varro. A Benchmark Evaluation of Incremental Pattern Matching in Graph Transformation. In *Proceedings of the 4th International Conference on Graph Transformation, ICGT'08*, pages 396–410, Leicester, UK, 2008.
5. G. Berthelot. Transformations and Decompositions of Nets. In *Lecture Notes in Computer Science: Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties . / W. Brauer, W. Reisig, and G. Rozenberg (Eds.)*, volume 254, pages 360–376. Springer Verlag, 1987.

6. I. Corro Ramos, A. Di Bucchianico, Hakobyan L., and K.M. van Hee. Synthesis and Reduction of State Machine Workflow Nets. Technical report CS 06-18, Edinhoven University of Technology, 2006.
7. I. Corro Ramos, A. Di Bucchianico, Hakobyan L., and K.M. van Hee. Model Driven Testing Based On Test History. In *Transactions on Petri Nets and Other Models of Concurrency I*, volume 1, pages 134–151, 2008.
8. J. Desel. Reduction and Design of Well-behaved Concurrent Systems. In *Proceedings on Theories of concurrency : unification and extension, CONCUR '90*, pages 166–181, New York, NY, USA, 1990. Springer-Verlag New York, Inc.
9. E. W. Dijkstra. Go To Statement Considered Harmful. *Communications of the ACM*, 11(3):147–148, March 1968.
10. B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst. The ProM Framework: A New Era in Process Mining Tool Support. In *Lecture Notes in Computer Science: Applications and Theory of Petri Nets 2005: 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005. / Gianfranco Ciardo, Philippe Darondeau (Eds.)*, volume 3536, pages 444–454. Springer Verlag, jun 2005.
11. C. Girault and R. Valk, editors. *Petri Nets for System Engineering: A Guide to Modelling, Verification, and Applications*. Springer-Verlag, 2002.
12. K. M. van Hee, J. Hidders, G. Houben, J. Paredaens, and P. Thiran. On the Relationship between Workflow Models and Document Types. *Information Systems*, 34(1):178–208, 2008.
13. K. M. van Hee and Zheng Liu. From Service-Oriented Architecture via Coloured Petri Nets to Java Code. In *Proceedings of the Fifth International Workshop on Modellig of Objects, Compomens and Agents, MOCA'09*, pages 129–149, Hamburg, Germany, 2009.
14. K. M. van Hee, N. Sidorova, and M. Voorhoeve. Generalised Soundness of Workflow Nets Is Decidable. In *Proceedings of the 25th International Conference, ICATPN'04*, Bologna, Italy, 2004.
15. A. Lim, W. C. Oon, and W. B. Zhu. Towards Definitive Benchmarking of Algorithm Performance. In *Proceedings of the 11th European Conference on Information Systems, ECIS 2003*, Naples, Italy, 2003.
16. Architecture of Information System Group at indhoven University of Technology. Prom Nightly Builds. http://prom.win.tue.nl/tools/prom/nightly/, 2010.
17. N. Russell, A. H. M. ter Hofstede, W. M. P. van der Aalst, and M. Mulyar. Workflow Controlflow Patterns: A Revised View. Technical report, 2006.
18. I. Suzuki and T. Murata. A Method for Stepwise Refinement and Abstraction of Petri Nets. *Journal of Computer and System Sciences*, 27(1):51–76, 1983.

Bounded Model Checking for Parametric Timed Automata[⋆]

Michał Knapik[1] and Wojciech Penczek[1,2]

[1] Institute of Computer Science, PAS, J.K. Ordona 21, 01-237 Warszawa, Poland
{Michal.Knapik}@ipipan.waw.pl

[2] Institute of Informatics, Podlasie Academy, Sienkiewicza 51, 08-110 Siedlce, Poland
penczek@ipipan.waw.pl

Abstract. The paper shows how bounded model checking can be applied to parameter synthesis for parametric timed automata with continuous time. While it is known that the general problem is undecidable even for reachability, we show how to synthesize a part of the set of all the parameter valuations under which the given property holds in a model. The results form a complete theory which can be easily applied to parametric verification of a wide range of temporal formulae – we present such an implementation for the existential part of $\mathrm{CTL}_{-\mathrm{X}}$.

1 Introduction and related work

The growing abundance of complex systems in real world, and their presence in critical areas fuels the research in formal specification and analysis. One of the established methods in systems verification is model checking, where the system is abstracted into the algebraic model (e.g. various versions of Kripke structures, Petri nets, timed automata), and then processed with respect to the given property (usually a formula of modal or temporal logic). Classical methods have their limits however – the model is supposed to be a complete abstraction of system behaviour, with all the timing constraints explicitely specified. This situation has several drawbacks, e.g. the need to perform a batch of tests to confirm the proper system design (or find errors) is often impossible to fullfill due to the high complexity of the problem. Introducing parameters into models changes the task of *property verification* to task of *parameter synthesis*, meaning that parametric model checking tool produces the set of parameter valuations under which the given property holds instead of simple *holds/does not hold* answer. Unfortunately, the problem of parameter synthesis is shown to be undecidable for some of widely used parametric models, e.g. parametric timed automata [3, 8] and bounded parametric time Petri nets [15].

Many of model checking tools acquired new capabilities of parametric verification, e.g. UPPAAL-PMC [11] – the parametric extension of UPPAAL, LPMC [14] – extending PMC. Some of the tools were built from scratch with parametric

⋆ Partly supported by the Polish Ministry of Science and Higher Education under the grant No. N N206 258035.

model checking in mind, e.g. TREX [1] and MOBY/DC [7]. Parametric analysis is also possible with HyTech [10] by means of hybrid automata. However, due to undecidability issues, algorithms implemented in these tools need not to stop and are very time and resource consuming. Another, very interesting approach is given in a recently developed IMITATOR tool [4] – having both the parametric timed automaton and the initial parameter valuation, IMITATOR synthesizes a set of parameter constraints. Substituting the parameters with a valuation satisfying these constraints is guaranteed to produce the timed automaton which is *time-abstract* equivalent to the one obtained from substituting the parameters with the initial valuation.

In this paper we present a new approach to parametric model checking, based on the observation that while we are not able to synthesize the *full* set of parameter constraints in general, there is no fundamental rule which forbids us from obtaining a *part* of this set. In Section 2 we introduce the parametric region graph – an extension of region graph used in theory of timed automata [2] and show (in Section 3) how the computation tree of a model can be unwinded up to some finite depth in order to apply bounded model checking (BMC) techniques [5]. To the best knowledge of the authors, this is the first application of BMC to parametric timed automata and seems to be a quite promising direction of research – firstly due to the unique BMC advantage which allows for verification of properties in limited part of the model, secondly due to the fact that it is quite easy to present BMC-based model checking algorithms for existential parts of many modal and temporal logics. In fact we describe how Parametric BMC can be implemented for the existential subset of CTL$_{-\mathrm{X}}$ logic in Section 3, including the analysis of a simplified parametric model of the 4-phase handshake protocol.

2 Theory of Parametric Timed Automata

In this paper we use two kinds of variables, namely *parameters* $P = \{p_1, \ldots, p_m\}$ and *clocks* $X = \{x_0, \ldots, x_n\}$. An expression of the form $\sum_{i=1}^{m} t_i \cdot p_i + t_0$, where $t_i \in \mathbb{Z}$ is called a *linear expression*. A *simple guard* is an expression of the form $x_i - x_j \prec e$, where $i \neq j$, $\prec \in \{\leq, <\}$ and e is a linear expression. A conjunction of simple guards is called a *guard* and the set of all guards is denoted by G. We valuate the clocks in nonnegative reals, and parameters in naturals (including 0) that is $\upsilon : P \to \mathbb{N}$ is a *parameter valuation* and $\omega : X \to \mathbb{R}^{\geq 0}$ is a *clock valuation* (both υ and ω can be thought of as points in, respectively, $\mathbb{N}^m$ and $\mathbb{R}^{\geq 0^n}$). Additionally, following [11] we assume that $\omega(x_0) = 0$ – the "false clock" x_0 is fixed on 0 for convenience only, for uniform presentation of guards. By $e[\upsilon]$ we denote the value obtained by substituting the parameters in a linear expression e according to parameter valuation υ. We denote $\omega \models_\upsilon x_i - x_j \prec e$ iff $\omega(x_i) - \omega(x_j) \prec e[\upsilon]$ holds, and naturally extend this notion to guards. We also need a notion of *reset* that is a set of expressions of the form $x_i := b_i$ where $b_i \in \mathbb{N}$, and $0 < i \leq n$. The set of all resets is denoted by R, and the action of resetting a clock valuation ω by reset $r \in R$ is defined as following: $\omega[r]$ is a clock valuation such that $\omega[r](x_i) = b_i$ if $x_i := b_i \in r$, and $\omega[r](x_i) = \omega(x_i)$

otherwise. If $\delta \in \mathbb{R}$ and ω is a clock valuation, then $\omega + \delta$ is a clock valuation such that $(\omega + \delta)(x_i) = \omega(x_i) + \delta$ for all $0 < i \leq n$, and $\omega(x_0) = 0$. An *initial clock valuation* ω_0 is the valuation satisfying $\omega(x_i) = 0$ for all $x_i \in X$.

We also adopt a convenient notation from [11], where the $\leq$ symbol is treated as *true* and the $<$ symbol is treated as $false$. The propositional formulae built from symbols $\leq$ and $<$ are evaluated in a standard way. As to give an example, $\leq \Rightarrow <$ evaluates to $<$, $< \Rightarrow \leq$ evaluates to $\leq$, and $\neg(\leq \vee <)$ evaluates to $<$.

2.1 Parametric Timed Automata

Let us recall some notions from the theory of parametric timed automata. Non-parametric timed automata [2] are state-transition graphs augmented with a finite number of clocks, and clock constraints guarding the transitions between states. Their parametric version [3] allows for using parameters (other than clocks) in guard expressions – which may be perceived as creating the general template for system behaviour under more abstract timed constraints.

Definition 1. *A tuple* $\mathcal{A} = \langle Q, q_0, A, X, P, \rightarrow, I\rangle$ *where:*

- Q *is a set of* locations*,*
- $q_0 \in Q$ *is the* initial location*,*
- A *is a set of* actions*,*
- X *and* P *are, respectively, sets of clocks and parameters,*
- $I : Q \rightarrow G$ *is an* invariant *function,*
- $\rightarrow \subseteq Q \times A \times G \times R \times Q$ *is a transition relation.*

is called a parametric timed automaton *(PTA). All the above sets are finite. We abbreviate* (q, a, g, r, q') *as* $q \overset{a,g,r}{\rightarrow} q'$*.*

The semantics of PTA is presented below, in form of a labeled transition system.

Definition 2 (Concrete semantics). *Let* $\mathcal{A} = \langle Q, q_0, A, X, P, \rightarrow, I\rangle$ *be a parametric timed automaton and* υ *be a parameter valuation. The labeled transition system of* $\mathcal{A}$ *under* υ *is defined as a tuple* $[\mathcal{A}]_\upsilon = \langle S, s_0, \overset{d}{\rightarrow}\rangle$ *where:*

- $S = \{(q, \omega) \mid q \in Q,$ *and* ω *is a clock valuation such that* $\omega \models_\upsilon I(q)\}$,
- $s_0 = (q_0, \omega_0)$ *(we assume that* $\omega_0 \models_\upsilon I(q_0)$*),*
- *let* $(q, \omega), (q', \omega') \in S$*. The transition relation* $\overset{d}{\rightarrow}$ *is defined as follows:*
  - *if* $d \in \mathbb{R}^{\geq 0}$, *then* $(q, \omega) \overset{d}{\rightarrow} (q', \omega')$ *iff* $q = q'$ *and* $\omega' = \omega + d$,
  - *if* $d \in A$, *then* $(q, \omega) \overset{d}{\rightarrow} (q', \omega')$ *iff* $q \overset{a,g,r}{\rightarrow} q'$, *and* $\omega \models_\upsilon g$, *and* $\omega' = \omega[r]$.

The elements of S *are called the* concrete states *of* $\mathcal{A}_\upsilon$*.*

The automaton obtained by substituting parameters in the guards and the invariants of $\mathcal{A}$ by appropriate values of the parameter valuation υ is denoted by $\mathcal{A}_\upsilon$. The concrete semantics of $\mathcal{A}_\upsilon$ is defined as $[\mathcal{A}_\upsilon] = [\mathcal{A}]_\upsilon$. Notice that $\mathcal{A}_\upsilon$ is a timed automaton and $[\mathcal{A}_\upsilon]$ – its concrete semantics [2].

Our definition of parametric timed automata slightly differs from the one presented in [11], namely, we do not allow nonnegative reals as parameter values. As it was shown in [3], the choice of the parameter valuation codomain does not change the fact that the emptiness problem is undecidable. We explain the origin of this restriction in the following subsection.

2.2 Parametric Region Graph

In non-parametric timed automata theory, the *region graph* [2] is used as a part of a convenient method of presenting the concrete state space in a uniform, finite way. The finiteness of the resulting structure is a result of presence of both the *bounded* and *unbounded* regions. Intuitively, the bounded regions are convex bounded sets in the space of clock valuations, while the unbounded regions are convex and unbounded. The latter ones are defined using the maximal values of clock constraints – this is not possible in the general case of parametric timed automata (see however the optimization techniques in [11]), therefore in this paper we consider only the bounded regions. We divide the space of all the clock valuations into the set of *regions* using the following equivalence relation.

Definition 3. *Let ω, ω' be valuations of clocks $X = \{x_0, \ldots, x_n\}$. Then, $\omega \approx \omega'$ iff the following conditions hold:*

- $\lfloor \omega(x_i) \rfloor = \lfloor \omega'(x_i) \rfloor$ *for all* $x_i \in X$,
- *and* $frac(\omega(x_i)) < frac(\omega(x_j)) \iff frac(\omega'(x_i)) < frac(\omega'(x_j))$ *for all* $i \neq j, 1 \leq i, j \leq n$,
- *and* $frac(\omega(x_i)) = 0 \iff frac(\omega'(x_i)) = 0$ *for all* $x_i \in X$,

where $frac(\omega(x_i))$ denotes the fractional part of $\omega(x_i)$. The equivalence classes of $\approx$ are called (detailed) regions.

To our aims it is convenient to describe regions as sets of valuations satisfying certain guard expressions.

Lemma 1. *Let $X = \{x_0, \ldots, x_n\}$ be a set of clocks, and Z – a region of valuations. There exists a guard $g_Z = \bigwedge_{i,j \in \{0,\ldots,n\}, i \neq j} x_i - x_j \prec_{ij} b_{ij}$, such that $\prec_{ij} \in \{\leq, <\}$ and $b_{ij} \in \mathbb{Z}$ satisfying:*

$$Z = \{\omega \mid \omega \models g_Z\}.$$

Proof. We need to specify the values of b_{ij} together with the accompanying relation $\prec_{ij}$. Let $Z = [\omega]_{\approx}$ (the following considerations are valid for any choice of ω from Z).

- If $frac(\omega(x_i)) = 0$, $frac(\omega(x_j)) = 0$, let $\prec_{ij} = \leq$ and $b_{ij} = \lfloor \omega(x_i) \rfloor - \lfloor \omega(x_j) \rfloor$,
- if $frac(\omega(x_i)) \neq 0$, $frac(\omega(x_j)) = 0$, let $\prec_{ij} = <$ and $b_{ij} = \lceil \omega(x_i) \rceil - \lfloor \omega(x_j) \rfloor$,
- if $frac(\omega(x_i)) = 0$, $frac(\omega(x_j)) \neq 0$, let $\prec_{ij} = <$ and $b_{ij} = \lfloor \omega(x_i) \rfloor - \lfloor \omega(x_j) \rfloor$,
- for $frac(\omega(x_i)) \neq 0$, $frac(\omega(x_j)) \neq 0$:
  - if $frac(\omega(x_i)) = frac(\omega(x_j))$, let $\prec_{ij} = \leq$, $b_{ij} = \lfloor \omega(x_i) \rfloor - \lfloor \omega(x_j) \rfloor$,

- if $frac(\omega(x_i)) < frac(\omega(x_j))$, put $\prec_{ij}=<$, $b_{ij} = \lfloor\omega(x_i)\rfloor - \lfloor\omega(x_j)\rfloor$,
- if $frac(\omega(x_i)) > frac(\omega(x_j))$, let $\prec_{ij}=<$, $b_{ij} = \lceil\omega(x_i)\rceil - \lfloor\omega(x_j)\rfloor$.

It is easy to see that if $\omega \approx \omega'$, then for any guard g we have $\omega \models g$ iff $\omega' \models g$. Therefore, as g_Z was constructed in such a way that $\omega \models g_Z$, we have also $\omega' \models g_Z$ for all $\omega' \in Z$. On the other hand, if $\omega' \models g_Z$, then satisfaction of the guards of form $x_i - x_0 \prec_{i0} b_{i0}$ and $x_0 - x_i \prec_{0i} b_{0i}$ (recall that x_0 is fixed) guarantees that $\lfloor\omega'(x_j)\rfloor = \lfloor\omega(x_j)\rfloor$ for all $x_j \in X$. Similarly, $\omega'(x_i)$ has nonzero fractional value iff $frac(\omega(x_i)) \neq 0$, as $\omega'(x_i) \in (\lfloor\omega(x_i)\rfloor, \lceil\omega(x_i)\rceil)$, provided that $frac(\omega(x_i)) \neq 0$. Let us assume that $0 < frac(\omega(x_i))$, and $frac(\omega(x_i)) < frac(\omega(x_j))$, then from $\omega(x_i) - \omega(x_j) < \lfloor\omega(x_i)\rfloor - \lfloor\omega(x_j)\rfloor$ we have $\omega'(x_i) - \omega'(x_j) < \lfloor\omega'(x_i)\rfloor - \lfloor\omega'(x_j)\rfloor$. Therefore $\omega'(x_i) - \lfloor\omega'(x_i)\rfloor < \omega'(x_j) - \lfloor\omega'(x_j)\rfloor$, thus $frac(\omega(x_i)) < frac(\omega(x_j))$.

The guard constructed in the proof of the above lemma is called the *characteristic guard* of Z. In the above proof we used the fact that if one representative of an equivalence class satisfies a guard g, then so do all the remaining members. This is not true if we allow nonnegative reals as parameter values – for example it is easy to see that only some of representatives of class $[(0, 0.3)]$ satisfy $x_1 - x_0 < p$ under parameter valuation υ such that $\upsilon(p) = 0.5$.

Definition 4. *Let $\mathcal{A} = \langle Q, q_0, A, X, P, \rightarrow, I\rangle$ be a parametric timed automaton, $X = \{x_0, \ldots, x_n\}$ and $P = \{p_1, \ldots, p_m\}$. We introduce a relation in the set of all the pairs (Z, C) where Z is a region, and $C \subseteq \mathbb{N}^m$ is a subset of the set of all the valuations of parameters (treated as natural vectors). Let $s = x_i - x_j \prec e$ be a simple guard, and $g_Z = \bigwedge_{i,j\in\{0,\ldots,n\}, i\neq j} x_i - x_j \prec_{ij} b_{ij}$ the characteristic guard of region Z. Then we define:*

$$(Z, C) \overset{s}{\rightsquigarrow} (Z', C') \text{ iff } Z = Z' \text{ and } C' = C \cap \{\upsilon \mid b_{ij}(\prec_{ij} \Rightarrow \prec)e[\upsilon]\}.$$

Let g be a guard and s a simple guard, then:

$$(Z, C) \overset{g\wedge s}{\rightsquigarrow} (Z', C') \text{ iff for some } (Z'', C'') \text{ we have } (Z, C) \overset{g}{\rightsquigarrow} (Z'', C'')$$

$$\text{and } (Z'', C'') \overset{s}{\rightsquigarrow} (Z', C').$$

There is a natural intuition behind the above definition – if $(Z, C) \overset{g}{\rightsquigarrow} (Z', C')$ then (Z', C') contains all the pairs $(\omega, \upsilon) \in Z \times C$ such that $\omega \models_\upsilon g$. Such an operation is a counterpart for guard addition from [11], notice however that we do not need a burden of costly canonicalization. Below we state some basic properties of $\overset{g}{\rightsquigarrow}$ relation.

Lemma 2. *Let $(Z, C) \overset{g}{\rightsquigarrow} (Z', C')$, where g is a guard. Then, the following conditions hold:*

1. *if $(\omega, \upsilon) \in (Z, C)$ and $\omega \models_\upsilon g$, then $(\omega, \upsilon) \in (Z', C')$,*
2. *if $(\omega, \upsilon) \in (Z', C')$, then $\omega \models_\upsilon g$.*

Proof. Let us start with the first part of the lemma. Let us assume that $\omega \models_\upsilon g$. By the induction on the complexity of g we prove that $\upsilon \in C'$.

The base case is when $g = x_i - x_j \prec e$ (g is a simple guard). Let us assume that g_Z contains a simple guard of the form $x_i - x_j \leq b_{ij}$ where $b_{ij} \in \mathbb{Z}$. Notice that in this case the characteristic guard contains also a simple guard of the form $x_j - x_i \leq -b_{ij}$, therefore $b_{ij} = \omega'(x_i) - \omega'(x_j)$ for each $\omega' \in Z$. As $\omega \models_\upsilon g$, then $b_{ij} = \omega'(x_i) - \omega'(x_j) \prec e[\upsilon]$. Therefore $b_{ij} \prec e[\upsilon]$, which in this case means that $b_{ij}(\prec_{ij} \Rightarrow \prec) e[\upsilon]$. Now let us assume that g_Z contains a simple guard of the form $x_i - x_j < b_{ij}$. In this case, for each $\omega' \in Z$ there exists $\delta \in (0,1)$ such that $\omega'(x_i) - \omega'(x_j) = (b_{ij} - 1) + \delta$. Let us notice that $e[\upsilon] \in \mathbb{Z}$, therefore from $(b_{ij} - 1) + \delta = \omega'(x_i) - \omega'(x_j) \prec e[\upsilon]$ we obtain $b_{ij} \leq e[\upsilon]$. The latter inequality means that in this case $b_{ij}(\prec_{ij} \Rightarrow \prec) e[\upsilon]$ holds.

For the induction step, notice that if $(Z, C) \overset{g' \wedge s}{\rightsquigarrow} (Z', C')$ (g' is a guard, and s a simple guard), then there exists (Z'', C'') such that $(Z, C) \overset{g'}{\rightsquigarrow} (Z'', C'')$ and $(Z'', C'') \overset{s}{\rightsquigarrow} (Z', C')$. From the inductive assumption we obtain that as $\omega \models_\upsilon g' \wedge s$ implies $\omega \models_\upsilon g'$, then $\upsilon \in C''$. Similarly, as $(\omega, \upsilon) \in (Z'', C'')$ and $\omega \models_\upsilon s$, we have $\upsilon \in C'$.

The proof of the second part of the lemma is also by the induction on the structure of g. Assume that $g = x_i - x_j \prec e$ and g_Z contains a simple guard of form $x_i - x_j \prec_{ij} b_{ij}$. If $(Z, C) \overset{g}{\rightsquigarrow} (Z', C')$, then $C' = C \cap \{\upsilon \mid b_{ij}(\prec_{ij} \Rightarrow \prec) e[\upsilon]\}$. As $\omega(x_i) - \omega(x_j) \prec_{ij} b_{ij}$ and $b_{ij}(\prec_{ij} \Rightarrow \prec) e[\upsilon]$ then $\omega(x_i) - \omega(x_j)(\prec_{ij} \wedge (\prec_{ij} \Rightarrow \prec)) e[\upsilon]$. Therefore we have $\omega(x_i) - \omega(x_j) \prec e[\upsilon]$, thus $\omega \models_\upsilon g$.

For the induction step, let us notice that if $(Z, C) \overset{g' \wedge s}{\rightsquigarrow} (Z', C')$, then there exists (Z'', C'') such that $(Z, C) \overset{g'}{\rightsquigarrow} (Z'', C'')$ and $(Z'', C'') \overset{s}{\rightsquigarrow} (Z', C')$. If $(\omega, \upsilon) \in (Z', C')$ then by the inductive assumption $\omega \models_\upsilon s$ holds. As $C' \subseteq C'' \subseteq C$, then $\upsilon \in C''$ and $(\omega, \upsilon) \in (Z'', C'')$. Therefore, from the inductive assumption we obtain $\omega \models_\upsilon g'$ and, finally, $\omega \models_\upsilon g' \wedge s$.

From the above lemma we immediately obtain the following corollary.

Corollary 1. *Let Z be a region, and C a subset of set of all the parameter valuations. Then, the following conditions hold:*

1. *if $(Z, C) \overset{g}{\rightsquigarrow} (Z', C')$, then $Z' \times C' = Z \times C \cap \{(\omega, \upsilon) \mid \omega \models_\upsilon g\}$,*
2. *if $\omega \in Z$, $\upsilon \in C$, and $\omega \models_\upsilon g$, then $(Z, C) \overset{g}{\rightsquigarrow} (Z', C')$ for some Z', C' such that $(\omega, \upsilon) \in Z' \times C'$.*

In order to develop our theory further, we need to define two additional operations on regions.

Definition 5. *Let $Z = [\omega]_\approx$ be a region and $r \in R$ be a reset. Then,* resetting *of Z by r is defined as: $Z[r] = [\omega[r]]_\approx$.*

Clearly, resetting of a region does not depend on the choice of a representative.

Definition 6. *Let Z and Z' be two different regions. Region Z' is called a* time successor *of Z (denoted by $\tau(Z)$) iff for all $\omega \in Z$ there exists $\delta \in \mathbb{R}$ such that $\omega + \delta \in Z'$ and $\omega + \delta' \in Z \cup Z'$ for all $\delta' \leq \delta$.*

Now, we are in the position to present the notion of a *parametric region graph*, being an extension of *region graph* used in theory of timed automata [2]. The main idea is to augment regions with sets of parameter valuations under which the given concrete state (its equivalence class) is reachable from the initial state, and to mimick the transitions in the concrete semantics by their counterparts in parametric region graph.

Definition 7. *Let $\mathcal{A} = \langle Q, q_0, A, X, P, \to, I\rangle$ be a parametric timed automaton. Define the* parametric region graph *of $\mathcal{A}$ as the tuple $PREG(\mathcal{A}) = \langle S, s_0, \overset{d}{\to}\rangle$ where:*

- *$S = \{(q, Z, C) \mid q \in Q, Z$ is a region, $C \subseteq \mathbb{N}^m$ and $\forall_{\upsilon \in C} \exists_{\omega \in Z}\ \omega \models_\upsilon I(q)\}$,*
- *$s_0 = (q_0, Z_0, C_0)$ where $Z_0 = [\omega_0]_\approx$ and $C_0 = \{\upsilon \mid \omega_0 \models_\upsilon I(q_0)\}$,*
- *$(q, Z, C) \overset{d}{\to} (q', Z', C')$ is defined as follows:*
  - *if $d = \tau$ (time transition), then $q = q'$, $Z' = \tau(Z)$, and C' is such that $(Z', C) \overset{I(q)}{\leadsto} (Z', C')$,*
  - *if $d \in A$ (action transition), then there exists a transition $q \overset{d,g,r}{\to} q'$ in $\mathcal{A}$ and C'' such that $(Z, C) \overset{g}{\leadsto} (Z, C'')$ and $(Z[r], C'') \overset{I(q')}{\leadsto} (Z', C')$.*

Additionally, we call nodes of type $(q, Z, \emptyset)$ dead*, and assume that they have no outgoing transitions.*

Notice that in the above definition we could replace $\exists$ with $\forall$, due to the fact that for any guard g, fixed parameter valuation υ, and clock valuations ω, ω' such that $\omega \approx \omega'$ we have $\omega \models_\upsilon g$ iff $\omega' \models_\upsilon g$.

Both the concrete semantics of (parametric) timed automaton, and (parametric) region graph are *labelled transition systems.* We define finite and infinite runs in a labelled transition system in a usual way.

Lemma 3. *Let A be a parametric timed automaton, and $\rho_n = s_0, s_1, \ldots s_n$ a finite run in $PREG(\mathcal{A})$, where $s_i = (q_i, Z_i, C_i)$, and $C_n \neq \emptyset$. For any $(\omega, \upsilon) \in Z_n \times C_n$ there exists a finite run $\mu_n = t_0, t_1, \ldots t_n$ in $\mathcal{A}_\upsilon$, such that $t_i = (q_i, \omega_i)$, $\omega_i \in Z_i$ for $i \in \{0, \ldots, n\}$, and $\omega_n = \omega$.*

Proof. The base case of $n = 0$ is straightforward – as from the definition of $PREG(\mathcal{A})$ we have $\omega \models_\upsilon I(q_0)$ for any $(\omega, \upsilon) \in Z_0 \times C_0$.

Recall that $C_n \subseteq C_{n-1}$. If $s_{n-1} \overset{d}{\to} s_n$ is a time transition (with $d = \tau$), then $\tau(Z_{n-1}) = Z_n$. Therefore for each $\omega_n \in Z_n$ there exist $\omega_{n-1} \in Z_{n-1}$, and $l \in \mathbb{R}$, such that $\omega_n = \omega_{n-1} + l$. We conclude the case by noticing that $(\omega_{n-1}, \upsilon) \in Z_{n-1} \times C_{n-1}$, $\omega_n \models_\upsilon I(q_n)$, and using the inductive assumption.

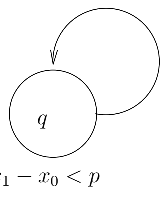
Now, if $s_{n-1} \overset{d}{\to} s_n$ is an action transition ($d \in A$), then there exists a transition $q_{n-1} \overset{d,g,r}{\to} q_n$ in $\mathcal{A}$, and a subset C' of $\mathbb{N}^m$, such that $(Z_{n-1}, C_{n-1}) \overset{g}{\leadsto}$

(Z_{n-1}, C'), and $(Z_{n-1}[r], C') \overset{I(q_n)}{\rightsquigarrow} (Z_{n-1}[r], C_n)$. Therefore for each $\omega_n \in Z_n$ we have $\omega_n \models_v I(q_n)$, and there exists $\omega_{n-1} \in Z_{n-1}$ such that $\omega_n = \omega_{n-1}[r]$, $\omega_{n-1} \models_v I(q_{n-1})$, and $\omega_{n-1} \models_v g$ (notice that $v \in C_n \cap C' \cap C_{n-1}$). We conclude the case by assuming $t_{n-1} = (q_{n-1}, \omega_{n-1})$, $t_n = (q_n, \omega_n)$ and using the inductive assumption.

Notice that the definition of the transition relation in $PREG(\mathcal{A})$ implies that in ρ_n we have $C_{i+1} \subseteq C_i$ for all $0 \le i < n$. In particular $C_n \subseteq C_i$ for all $0 \le i \le n$.

The above lemma does not extend to infinite runs, as shown in the following example.

Example 1. Consider the simple parametric timed automaton:



The following infinite run in $PREG(\mathcal{A})$ does not have a counterpart in $\mathcal{A}_v$ due to the fact that p is unbounded.

$$(q, [(0,0)], \{p \mid p > 0\}) \xrightarrow{\tau} (q, [(0,0.1)], \{p \mid p \ge 1\}) \xrightarrow{\tau}$$

$$(q, [(0,1)], \{p \mid p > 1\}) \xrightarrow{\tau} (q, [(0,1.1)], \{p \mid p \ge 2\}) \xrightarrow{\tau} \dots$$

Consider a transition $(q, Z, C) \xrightarrow{d} (q', Z', C')$ in $PREG(\mathcal{A})$. Notice that if $\omega \in Z$, $v \in C \cap C'$, then $(q, \omega) \xrightarrow{d'} (q', \omega')$ in $[\mathcal{A}_v]$, where $d' = d$ if d is an action, and d' is some real number if $d = \tau$. From this observation and Lemma 3 we obtain the following corollary.

Corollary 2. *Let $\rho = s_0, s_1, \dots$ be an infinite run in $PREG(\mathcal{A})$, such that $s_i = (q_i, Z_i, C_i)$ for some Z_i, C_i, and let $v \in C_i$ for all $i \ge 0$. Then, there exists an infinite run $\mu = t_0, t_1, \dots$ in the concrete semantics of $\mathcal{A}_v$, such that $t_i = (q_i, \omega_i)$, and $\omega_i \in Z_i$.*

The counterpart of Lemma 3 holds without the restriction on finiteness of runs.

Lemma 4. *Let $\mathcal{A}$ be a parametric timed automaton, and $\mu = t_0, t_1, \dots t_n \dots$ an infinite (finite) run in $\mathcal{A}_v$, where $t_i = (q_i, \omega_i)$, and such that if $t_i \xrightarrow{d} t_{i+1}$ is a time transition, then $[\omega_{i+1}] = \tau([\omega_i])$. Then, there exists an infinite (finite, resp.) run $\rho = s_0, s_1, \dots s_n \dots$ in $PREG(\mathcal{A})$ such that $s_i = (q_i, Z_i, C_i)$, and $(\omega_i, v) \in Z_i \times C_i$ for each $i \ge 0$ ($0 \le i \le n$, resp.).*

Proof. Let us start with the finite run case, and let $Z_i = [\omega_i]$. The base case is straightforward – just assume $C_0 = \{u \mid \omega_0 \models_u I(q_0)\}$ and notice that $\upsilon \in C_0$.

Assume that we have already constructed a finite run $\rho_n = s_0, s_1, \dots s_{n-1}$.

If $t_{n-1} \overset{d}{\to} t_n$ is a time transition, then $\tau(Z_{n-1}) = Z_n$, $\omega_n \in Z_n$, $\upsilon \in C_{n-1}$, and $\omega_n \models_\upsilon I(q_n)$. Therefore, from Corollary 1 we obtain that there exists C' such that $(Z_n, C_{n-1}) \overset{I(q_n)}{\rightsquigarrow} (Z_n, C')$, $\upsilon \in C'$, and conclude the case by placing $C_n = C'$, and the inductive assumption.

If $t_{n-1} \overset{d}{\to} t_n$ is an action transition, then there exists a transition in $\mathcal{A}$ such that for some guard g and reset r we have $q_{n-1} \overset{d,g,r}{\to} q_n$. Notice that as $(\omega_{n-1}, \upsilon) \in Z_{n-1} \times C_{n-1}$, $\omega_{n-1} \models_\upsilon g$, $\omega_{n-1}[r] = \omega_n$, and $\omega_n \models_\upsilon I(q_n)$, from Corollary 1 we have that there exist sets C', C'' satisfying $(Z_{n-1}, C_{n-1}) \overset{g}{\rightsquigarrow} (Z_{n-1}, C')$, $\upsilon \in C'$, and $(Z_{n-1}, C') \overset{I(q_n)}{\rightsquigarrow} (Z_n, C'')$. We conclude the case by assuming $C_n = C''$.

Let $\mu = t_0, t_1, \dots$ be an infinite run in $\mathcal{A}_\upsilon$. We have already shown that for each finite prefix $\mu_n = t_0, t_1, \dots t_n$ we can construct its counterpart $\rho_n = s^n_0, s^n_1, \dots s^n_n$ in $PREG(\mathcal{A})$, where $s^i_n = (q_i, Z_i, C^n_i)$. Notice that $C^n_i = C^{n+1}_i$, so the infinite sequence $\rho = s_0, s_1, \dots$, where $s_i = (q_i, Z_i, C^i_i)$ is a valid infinite run in $PREG(\mathcal{A})$ satisfying $(\omega_i, \upsilon) \in Z_i \times C^i_i$ for all $i \geq 0$.

The following definition formalizes the connection between parametric region graph, and region graphs. In what follows, by a subgraph of $PREG(\mathcal{A}) = \langle S, s_0, \overset{d}{\to}\rangle$ we mean a tuple $\langle S', s_0, \overset{d}{\hookrightarrow}\rangle$, where S' is a subset of S, and $\overset{d}{\hookrightarrow}$ is the restriction of $\overset{d}{\to}$ to S'.

Definition 8. *Let $\mathcal{A}$ be a parametric timed automaton, υ – a parameter valuation, and F – a subgraph of $PREG(\mathcal{A})$. By $proj(F, \upsilon)$ we define a subgraph of F whose states are tuples (q, Z, C) such that $\upsilon \in C$.*

Observe that $proj(PREG(\mathcal{A}), \upsilon)$ is in fact isomorphic with the region graph of $\mathcal{A}_\upsilon$ – by a forgetful functor stripping C from tuple (q, Z, C).

3 Bounded Model Checking for $\mathbf{ECTL_{-X}}$

The central idea of bounded model checking is to unfold the computation tree of a considered model up to some depth, and then perform the analysis of such a finite structure [5]. Such an approach limits us to verification (and in our case – parameter synthesis) of existential properties only, it should be noted however that implicit model checking methods often fail in case of large and complex systems. Bounded model checking seems to be especially effective in searching for counterexamples, i.e. in proving that some undesirable property holds in a model. This allows for detection of serious design flaws of concurrent and reactive systems.

The non-parametric model checking tool verifies a model (system specification) against a given property (usually in form of a temporal logic formula), producing the answer of simple *holds/does not hold* type. Its parametric counterpart is supposed to work slightly differently – having a parametric model we expect the answer in form of a set of parameter values under which a given property is satisfied. The automated synthesis of a complete set of desired parameter valuations is not possible in case of timed automata due to general undecidability of the problem, however obtaining a part of this set still seems to be a worthy goal. Our approach allows for incremental synthesis of parameters, i.e. if the valuations obtained by analysis of a part of a computation tree are not sufficient, then the tree can be unfolded up to a greater depth for further analysis. Combined with an expert supervision, the synthesized parameter valuations can give rise to hypotheses specifying the whole space of desired parameters.

We propose the following general flow of property verification/parameter synthesis.

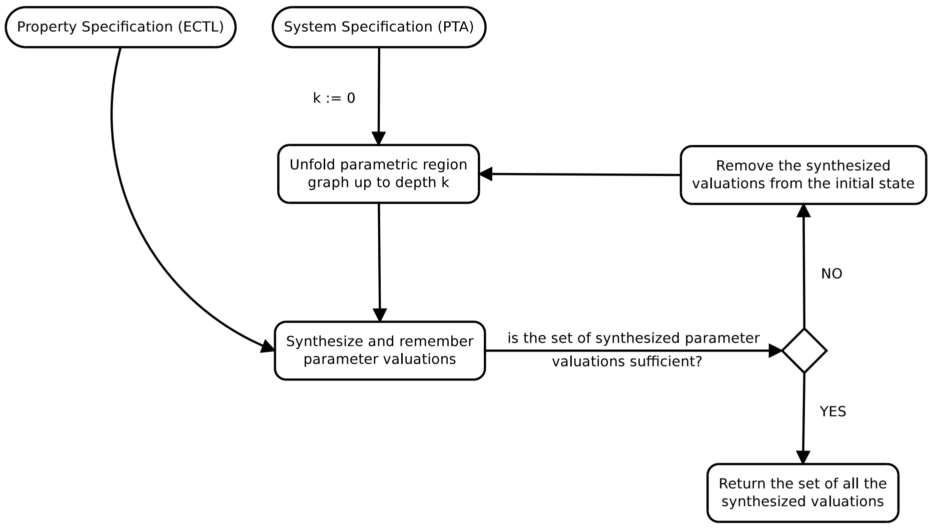

Property Specification (ECTL)
System Specification (PTA)
k := 0
Unfold parametric region graph up to depth k
Remove the synthesized valuations from the initial state
NO
Synthesize and remember parameter valuations
is the set of synthesized parameter valuations sufficient?
YES
Return the set of all the synthesized valuations

Fig. 1. Parametric Bounded Model Checking schema

The above diagram is very general. One of the approaches in the current applications of bounded model checking to verification of system properties is to encode the limited part of the computation tree together with a property in question as a propositional formula [6, 13]. The result can be checked using an efficient SAT-solver.

3.1 From Parametric Region Graph to concrete semantics

The $PREG(\mathcal{A})$ structure is infinite. In order to represent the infinite runs in a finite substructure we need a notion of *loop*.

Definition 9. *Let $\rho_n = s_0, s_1, \ldots s_n$ be a finite run in $PREG(\mathcal{A})$, and $s_i \xrightarrow{d} s_{i+1}$ for all $0 \leq i < n$. If $s_n = (q_n, Z_n, C_n)$ and there exists $s_i = (q_i, Z_i, C_i)$, where $0 \leq i < n$ such that $s_n \xrightarrow{d} s_i$ and $q_n = q_i$, $Z_n = Z_i$, then ρ_n is called a* loop.

Let $\rho_n = s_0, s_1, \ldots s_n$ be a loop in $PREG(\mathcal{A})$, such that $s_i = (q_i, Z_i, C_i)$, and $(q_n, Z_n) = (q_j, Z_j)$ for some $j < n$. We can create an infinite run $\hat{\rho} = \hat{s_0}, \hat{s_1}, \ldots$ by unwinding the ρ_n loop as follows:

$$\hat{s_i} = \begin{cases} (q_i, Z_i, C_n) & \text{for } i < n \\ (q_{j+(n-i)mod(n-j)}, Z_{j+(n-i)mod(n-j)}, C_n) & \text{for } i \geq n. \end{cases}$$

The validity of such a construction is based on the observation that $C_n \subseteq C_i$ for all $0 \leq i \leq n$ and the fact that transitions in $PREG(\mathcal{A})$ are defined in terms of g_Z and guards only. Applying Corollary 2 to such an unwinding we obtain the following corollary.

Corollary 3. *Let $\rho = s_0, s_1, \ldots, s_n$ be a loop in $PREG(\mathcal{A})$, where $s_i = (q_i, Z_i, C_i)$, and $\upsilon \in C_n$ – a parameter valuation. There exists an infinite run $\mu_t = t_0, t_1, \ldots$ in the concrete semantics of $\mathcal{A}_\upsilon$, where $t_i = (\hat{q_i}, \omega_i)$, $\omega_i \in Z_i$ for $i < n$, $\omega_i \in Z_{j+(n-i)mod(n-j)}$ for $i \geq n$, and:*

$$\hat{q_i} = \begin{cases} q_i & \textit{for } i < n \\ q_{j+(n-i)mod(n-j)} & \textit{for } i \geq n. \end{cases}$$

3.2 Parametric Bounded Model Checking for ECTL$_{-\mathrm{X}}$

The presented method can be applied to the verification of a variety of properties. As the example, in this subsection we present the application of introduced theory to verification of properties specified in the existential part of Computation Tree Logic (CTL$_{-\mathrm{X}}$) without the *next* operator [9] – namely ECTL$_{-\mathrm{X}}$. Intuitively, CTL$_{-\mathrm{X}}$ uses a branching time model, where many possible paths in the future exist. The whole CTL$_{-\mathrm{X}}$ contains both the universal ("for all the possible paths") and existential modalities ("there exists a path in the future") while ECTL$_{-\mathrm{X}}$ contains only the latter ones – see [13] for more thorough treatment.

Definition 10 (CTL$_{-\mathrm{X}}$ **and** ECTL$_{-\mathrm{X}}$ **syntax).** *Let $\mathcal{PV}$ be a set of propositions containing the* true *symbol, and $p \in \mathcal{PV}$. The set of well-formed* CTL$_{-\mathrm{X}}$ *formulae is given by the following grammar:*

$$\Phi ::= p \mid \neg\Phi \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid EG\Phi \mid E\Phi U\Phi.$$

The existential subset of CTL$_{-\mathrm{X}}$*, i.e.* ECTL$_{-\mathrm{X}}$ *is defined as a restriction of* CTL$_{-\mathrm{X}}$ *such that the negation can be applied to the propositions only.*

Additionally we use the derived modalities: $EF\alpha \overset{def}{=} E(trueU\alpha)$, $AF\alpha \overset{def}{=} \neg EG\neg\alpha$, $AG\alpha \overset{def}{=} \neg EF\neg\alpha$. Each modality of CTL$_{-\mathrm{X}}$ has an intuitive meaning. The path quantifier A stands for "on every path" and E means "there exists a path". G stands for "in all the states", F means "in some state", and U has a meaning of "until".

We augment the given parametric timed automaton $\mathcal{A} = \langle Q, q_0, A, X, P, \to, I\rangle$ with a labelling function $\mathcal{L} : Q \to 2^{\mathcal{PV}}$. Let us present an intepretation of ECTL$_{-\mathrm{X}}$ formulae for a parametric region graph.

Definition 11 (ECTL$_{-\mathrm{X}}$ semantics for parametric region graph). *Let $\mathcal{A} = \langle Q, q_0, A, X, P, \to, I\rangle$ be a parametric timed automaton, and F – a subgraph of its parametric region graph, such that (q_0, Z_0, C_0'), where $C_0' \subseteq C_0$, is a state of F. Let s be a state of F, $p \in \mathcal{PV}$, and α, β be* ECTL$_{-\mathrm{X}}$ *formulae. We treat F as a model for* ECTL$_{-\mathrm{X}}$ *formulae, defining the $\models$ relation as follows.*

1. *$F, (q, Z, C) \models p$ iff $p \in \mathcal{L}(q)$,*
2. *$F, s \models \neg p$ iff $F, s \not\models p$,*
3. *$F, s \models \alpha \vee \beta$ iff $F, s \models \alpha$ or $F, s \models \beta$,*
4. *$F, s \models E\alpha U\beta$ iff there exists a run $\rho_n = s_0, s_1, \ldots$, where $s_0 = s$, s_i are states of F for $i \geq 0$, $F, s_j \models \beta$ for some $j \geq 0$, and $F, s_i \models \beta$ for all $i < j$,*
5. *$F, s \models EG\alpha$ iff there exists a run $\rho_n = s_0, s_1, \ldots$, such that $F, s_i \models \alpha$ for all $i \geq 0$.*

We abbreviate $F, (q_0, Z_0, C_0) \models \alpha$ as $F \models \alpha$.

The counterpart of the above definition for the timed automaton $\mathcal{A}_\upsilon = \langle S, s_0, \xrightarrow{d}\rangle$ obtained from the parametric timed automaton $\mathcal{A}$ under the parameter valuation υ is similar – except for that it is defined over the concrete semantics ($s \in S$). Therefore the only difference is in the first clause which takes the following form:

1. *$\mathcal{A}_\upsilon, (q, \omega) \models p$ iff $p \in \mathcal{L}(q)$*

As previously, we abbreviate $\mathcal{A}_\upsilon, (q_0, \omega_0) \models \alpha$ as $\mathcal{A}_\upsilon \models \alpha$.

In order to apply bounded model checking to verification of temporal properties in $PREG(\mathcal{A})$ we need to specify the version of the above semantics for finite subgraphs of $PREG(\mathcal{A})$. The only difference concerns clauses 4 and 5 which take the following form:

4. *$F, s \models E\alpha U\beta$ iff there exists a finite run $\rho_n = s_0, s_1, \ldots s_n$, where $s_0 = s$, s_i are states of F for $0 \leq i \leq n$, $F, s_j \models \beta$ for some $0 \leq j \leq n$, and $F, s_i \models \beta$ for all $i < j$,*
5. *$F, s \models EG\alpha$ iff there exists a loop $\rho_n = s_0, s_1, \ldots, s_n$, such that $F, s_i \models \alpha$ for all $0 \leq i \leq n$.*

Recall that timed automaton $\mathcal{A}_\upsilon$ is *strongly non-zeno* (see [16]) iff for each sequence of states $q_1, \ldots, q_n$ such that $q_i \xrightarrow{a_i, g_i, r_i} q_{i+1}$ for all $0 \leq i < n$, and $q_n \xrightarrow{a_n, g_n, r_n} q_1$ (we call such a sequence a *structural loop*) there exists a clock x satisfying the following conditions:

– for some $1 \leq i \leq n$ the x clock is reset in step i (i.e. $x := 0 \in r_i$),

– there exists $1 \leq j \leq n$ such that for any clock valuation ω if $\omega \models_{\upsilon} g_j$, then $\omega(x) \geq 1$.

Intuitively, if an automaton is strongly non-zeno, then in each its loop at least one unit of time elapses ([16]). Notice that checking if the automaton is strongly non-zeno does not require any representation of the state space.

Theorem 1. *Let $\mathcal{A}$ be a parametric timed automaton, F – a finite subgraph of $PREG(A)$ containing state (q_0, Z_0, C'_0), where $C'_0 \subseteq C_0$, and $P = \bigcap\{C \mid (q, Z, C)$ is a state of $F\}$. If P is nonempty, and $\mathcal{A}_{\upsilon}$ is strongly non-zeno for each $\upsilon \in P$, then for each formula $\alpha \in \mathrm{ECTL}_{-\mathrm{X}}$ if $F \models \alpha$, then $\mathcal{A}_{\upsilon} \models \alpha$ for all $\upsilon \in P$.*

Proof. Let $\upsilon \in P$ be a parameter valuation. Denote by $\hat{F}$ a (possibly infinite) subgraph of $PREG(\mathcal{A})$ created in two steps:

– firstly, by adding to F the new states created by unwinding of each loop along the lines presented above – obtaining F',
– secondly, by replacing all the states (q, Z, C) in F' by (q, Z, P) – obtaining $\hat{F}$.

It is easy to see that $F \models \alpha$ iff $\hat{F} \models \alpha$. Recall that $proj(\hat{F}, \upsilon)$ is isomorphic to some subgraph of the region graph of $\mathcal{A}_{\upsilon}$. As satisfiability of $\mathrm{ECTL}_{-\mathrm{X}}$ formulae in a subgraph of the region graph implies satisfiability in the region graph, and satisfiability in region graph is equivalent to satisfiability in the concrete model (see [16]) we obtain the thesis of the theorem.

3.3 Example – four phase handshake protocol

In this section we perform a first step in parametric analysis of a simplified version of four phase handshake protocol. The protocol is extensively used in practice and widely studied, having both the software and hardware implementations [**?**,**?**]. The considered system consists of two communicating entities – the Producer and the Consumer. The Producer creates data packages and sends them to the Consumer. Both the components communicate using two shared boolean variables, that is: **req** (request) governed by the Producer and used to signal the Consumer that the data is prepared and ready to be read, and **ack** (acknowledge) governed by the Consumer and used to signal the Producer that the data has been read successfully and the Consumer is ready. The initial value of both the variables is *false*.

The running system goes through the following sequence of signals (req, ack):

$$(false, false) \rightarrow (true, false) \rightarrow (true, true) \rightarrow (false, true) \rightarrow (false, false).$$

As we have no tool for automated analysis at our disposal yet, we analyze the simplified version of the system behaviour. We introduce two parameters, omitting the signal propagation time, namely: $minIO$, and $maxIO$ being, respectively, the lower and the upper bound on read/write time.

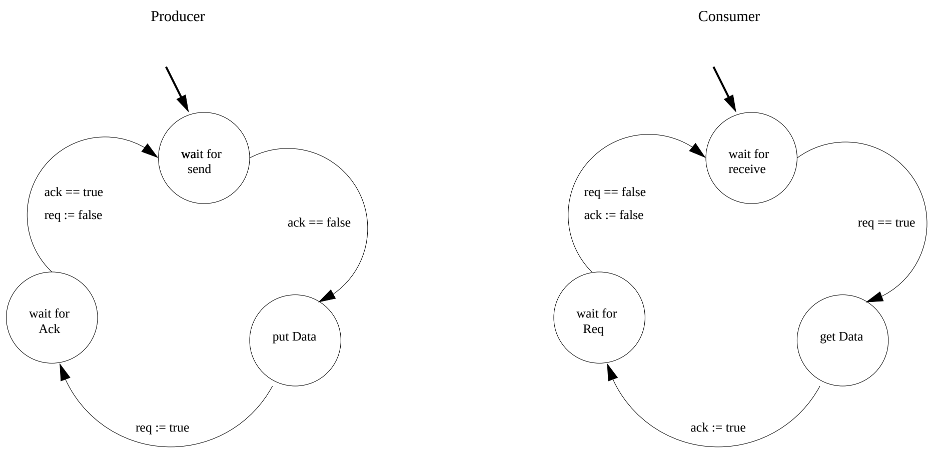


Fig. 2. 4–phase handshake protocol

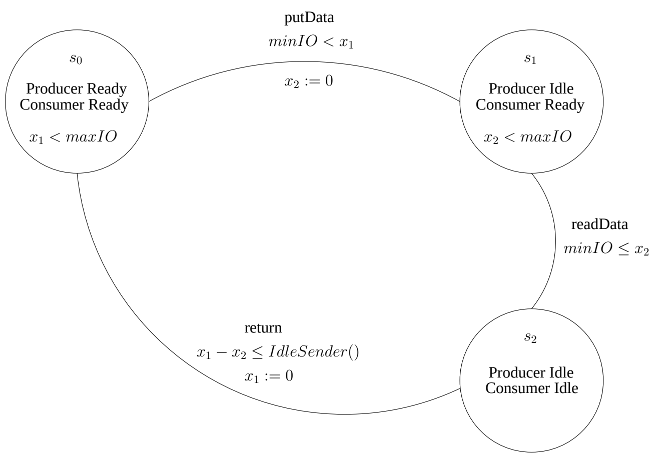


Fig. 3. 4–phase handshake protocol, behaviour diagram

The $IdleSender$ function guards the time that the Producer is allowed to be idle after putting data into some shared transmission vehicle (e.g. a bus). Let us put $IdleSender() := maxIO - minIO$ and unwind the Parametric Region Graph of Figure 3 (we omit the dummy clock x_0).

Notice that the above graph contains a loop, introduced by the sequence of actions: $\tau, \tau, putData, readData, return$. This loop can be unwinded as presented in Subsection 4.1 into an infinite path in the Parametric Region Graph, and into

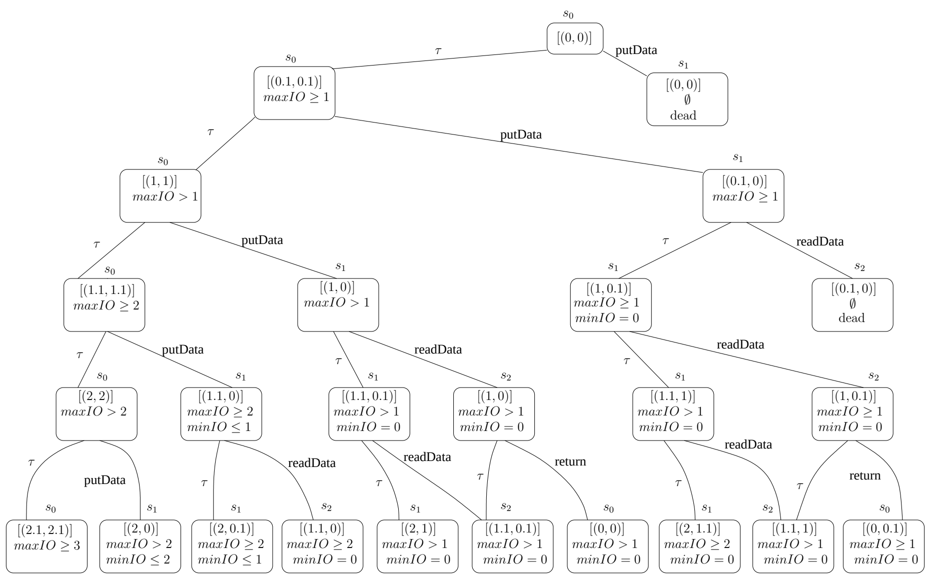

s_0 $[(0,0)]$
τ
putData
s_1 $[(0,0)]$ $\emptyset$ dead
s_0 $[(0.1,0.1)]$ $maxIO \geq 1$
τ
putData
s_0 $[(1,1)]$ $maxIO > 1$
s_1 $[(0.1,0)]$ $maxIO \geq 1$
τ
putData
τ
readData
s_0 $[(1.1,1.1)]$ $maxIO \geq 2$
s_1 $[(1,0)]$ $maxIO > 1$
s_1 $[(1,0.1)]$ $maxIO \geq 1$ $minIO = 0$
s_2 $[(0.1,0)]$ $\emptyset$ dead
τ
putData
τ
readData
τ
readData
s_0 $[(2,2)]$ $maxIO > 2$
s_1 $[(1.1,0)]$ $maxIO \geq 2$ $minIO \leq 1$
s_1 $[(1.1,0.1)]$ $maxIO > 1$ $minIO = 0$
s_2 $[(1,0)]$ $maxIO > 1$ $minIO = 0$
s_1 $[(1.1,1)]$ $maxIO > 1$ $minIO = 0$
s_2 $[(1,0.1)]$ $maxIO \geq 1$ $minIO = 0$
τ
putData
τ
readData
τ
readData
τ
return
τ
readData
τ
return
s_0 $[(2.1,2.1)]$ $maxIO \geq 3$
s_1 $[(2,0)]$ $maxIO > 2$ $minIO \leq 2$
s_1 $[(2,0.1)]$ $maxIO \geq 2$ $minIO \leq 1$
s_2 $[(1.1,0)]$ $maxIO \geq 2$ $minIO = 0$
s_1 $[(2,1)]$ $maxIO > 1$ $minIO = 0$
s_2 $[(1.1,0.1)]$ $maxIO > 1$ $minIO = 0$
s_0 $[(0,0)]$ $maxIO > 1$ $minIO = 0$
s_1 $[(2,1.1)]$ $maxIO \geq 2$ $minIO = 0$
s_2 $[(1.1,1)]$ $maxIO > 1$ $minIO = 0$
s_0 $[(0,0.1)]$ $maxIO \geq 1$ $minIO = 0$

Fig. 4. The 4–phase handshake protocol, Parametric Region Graph of depth 5

loops in concrete semantics of non-parametric timed automata with $minIO = 0$, and $maxIO$ instantiated by any value greater that 1.

The graph of Figure 4, treated as a subgraph of the Parametric Region Graph of Figure 3 allows us to observe that in the considered system the property $EGEF(ProducerIdle \wedge ConsumerReady)$ holds for $minIO = 0$, and $maxIO > 1$, with the previously mentioned loop as a witness. The intuition behind the considered formula is that the Producer will put data into the transmission infinitely often in the running system.

Of course, this is only the first, hand-made, step of synthesis of the parameter valuations under which the considered property is satisfied. The complete analysis of non-simplified versions with more parameters and components has to wait until we develop the planned tool.

4 Future work

The theory presented in this paper is to be implemented in Verics model checker [12]. There is a growing evidence [14, **?**] of success of model checking in verification of safety critical industrial applications, and the idea of parameter synthesis for a complex model or protocol seems to be promising in analysis and design of real-world systems. Also, as the method is quite general, we expect that it may be applied to many known temporal, modal and epistemic logics.

References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. R. Alur, T. Henzinger, and M. Vardi. Parametric real-time reasoning. In *Proc. of the 25th Ann. Symp. on Theory of Computing (STOC'93)*, pages 592–601. ACM, 1993.
3. É. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, October 2009.
4. A. Annichini, A. Bouajjani, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems. In *Proc. of CAV*, pages 368–372, 2001.
5. A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. In *Highly Dependable Software*, volume 58 of *Advances in Computers*. Academic Press, 2003. Pre-print.
6. I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou. Handshake protocols for de-synchronization. In *International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 149–158. IEEE Computer Society Press, 2004.
7. E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.
8. H. Dierks and J. Tapken. Moby/DC - a tool for model-checking parametric real-time specifications. In *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 271–277. Springer-Verlag, 2003.
9. L. Doyen. Robust parametric reachability for timed automata. *Information Processing Letters*, 102:208–213, 2007.
10. E. A. Emerson and E. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.
11. S.B. Furber and P. Day. Four-phase micropipeline latch control circuits. In *IEEE Transactions on VLSI Systems*, volume 4, pages 247–253, 1996.
12. T. Henzinger, P. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. In *Proc. of the 9th Int. Conf. on Computer Aided Verification (CAV'97)*, volume 1254 of *LNCS*, pages 460–463. Springer-Verlag, 1997.
13. T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. In *Proc. of the 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, volume 2031 of *LNCS*, pages 189–203. Springer-Verlag, 2001.
14. M. Kacprzak, W. Nabiałek, A. Niewiadomski, W. Penczek, A. Półrola, M. Szreter, B. Woźna, and A. Zbrzezny. VerICS 2008 - a model checker for time Petri nets and high-level languages. In *Proc. of Int. Workshop on Petri Nets and Software Engineering (PNSE'09)*, pages 119–132. University of Hamburg, 2009.
15. Spelberg R. L., De Rooij R. C. H., , and Toetenel W. J. Application of parametric model checking – the root contention protocol using lpmc. In *Proc. of the 7th ASCI Conference*, pages 73–85, February 2000.
16. W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.
17. M. Stoelinga. Fun with firewire: A comparative study of formal verification methods applied to the ieee 1394 root contention protocol. In *Formal Asp. Comput.*, volume 14, pages 328–337, 2003.

18. L-M. Tranouez, D. Lime, and O. H. Roux. Parametric model checking of time Petri nets with stopwatches using the state-class graph. In *Proc. of the 6th Int. Workshop on Formal Analysis and Modeling of Timed Systems (FORMATS'08)*, volume 5215 of *LNCS*, pages 280–294. Springer-Verlag, 2008.
19. S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001.

Specification of Decision Diagram Operations

Alexandre Hamez[1], Steve Hostettler[2], Alban Linard[2], Alexis Marechal[2], Emmanuel Paviot-Adet[3], and Matteo Risoldi[2]

[1] CNRS - LAAS, 7, avenue du Colonel Roche, 31077 Toulouse, France
`FirstName.LastName@laas.fr`

[2] Université de Genève, 7 route de Drize, 1227 Carouge, Switzerland
`FirstName.LastName@unige.ch`

[3] Université Pierre & Marie Curie, LIP6 - CNRS UMR 7606,
4 place Jussieu, 75252 Paris Cedex 05, France
`FirstName.LastName@lip6.fr`

Abstract. Decision Diagrams (DDs) are a well populated family of data structures, used for efficient representation and manipulation of huge data sets. Typically a given application requires choosing one particular category of DDs, like Binary Decision Diagrams (BDDs) or Data Decision Diagrams (DDDs), and sticking with it.

Each category provides a language to specify its operations. For instance, the operation language of BDDs provides `if-then-else`, `apply`, *etc.* We focus on two main kinds of operation languages: BDD-like and DDD-like. They overlap: some operations can be expressed in both kinds of languages, while others are only available in one kind.

We propose in this article a critical comparison of BDD-like and DDD-like languages. From the identified problems, we also propose a unified language for DD operations. It covers both BDD-like and DDD-like languages, and even some operations that cannot be expressed in either.

1 Introduction

Decision Diagrams (DDs) are now widely used in model checking as an extremely compact representations of state spaces [1]. Numerous DD categories have been developed over the past twenty years based on the same principles. Each category is adapted to a particular application domain and comes with a manipulation language, that is used to create and modify the DDs.

Typically a given application requires choosing one particular category of DDs, like Binary Decision Diagrams (BDDs) or Data Decision Diagrams (DDDs), and sticking with it. Then, the user has to learn its operations, which might be a non-trivial task. DDs are used for both efficient memory representation *and* efficient computation time. But knowing which operation leads to better efficiency sometimes requires deep knowledge and understanding of the DD category.

Two solutions try to circumvent this problem. The first one is to use high-level Domain Specific Languages (DSLs), specific to an application domain. For instance, CrocoPat[4] completely hides the DDs and provides to the user a language for manipulating relational expressions. We do *not* consider high-level languages here. The second solution is automatic optimization of low-level operations [2]. We are interested in this article only in low-level languages provided with the DDs. DSLs can then be translated to low-level programs.

Section 2 first does a brief presentation of DDs. It presents the terminology we are using in the remainder of the article, and the categories of DDs we cover.

We compare two kinds of languages, those used in BDD-like structures (BDDs, Algebraic Decision Diagrams (ADDs), Multi-valued Decision Diagrams (MDDs), for instance) in Section 3, and those used in DDD-like structures (DDDs, Set Decision Diagrams (SDDs), Σ Decision Diagrams (ΣDDs)) in Section 4. They offer very different operations to their users. Moreover, their expressiveness differ. We think they are therefore good candidates for comparison.

We then propose in Section 5 low-level operations that generalize both BDD-like and DDD-like operations. They only apply to DDs where the function's results are stored in terminal vertices, so Edge-Valued Decision Diagrams (EVDDs) are *not* covered in this article. Moreover, hierarchy as in SDDs and continuous input domains are not handled for simplicity.

2 Decision Diagram Principles

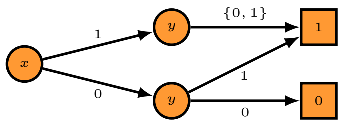
Decision Diagrams are representations of functions using Directed Acyclic Graphs (DAGs) with maximal sharing from the leaves and roots. DD categories differ on the signatures of represented functions. We note $\mathbb{B}$ the Boolean domain and $\mathbb{N}$ the natural integers. Function signatures are for instance $\mathbb{B}^n \to \mathbb{B}$ for BDDs [3], $\mathbb{B}^n \to \mathbb{N}$ for ADDs [4], and $\mathbb{N}^* \to \mathbb{B}$ (unbounded sequences of integers as inputs) for DDDs [5]. Categories also differ in the graph representation of functions. In most cases, the result is stored in terminal vertices, but Edge-Valued Binary Decision Diagrams (EVBDDs) [6] and Edge-Valued Multi-Valued Decision Diagrams (EVMDDs) [7] store them along the paths.

Represented functions are total. They return a value in their ouput domain for each value of their input domain. These domains can be infinite, either because they are discrete but not bounded as in DDDs or ADDs [4], or because they are continuous as in Interval Decision Diagrams (IDDs) [8]. As the DD is a finite graph, not all functions have a DD counterpart. This excludes functions that effectively return an infinite number of different values, as they cannot be represented by a finite number of terminals in the DD categories presented here.

A DD represents a function, which can also be seen as an association of each input element with an output one. We consider categories where every input element is a path of the graph, ending with a terminal vertex labeled by the function's corresponding output value. Each vertex on the path is a variable of the function, and edges are labeled by their possible values. An order over

[4] http://www.sosy-lab.org/~dbeyer/CrocoPat/

variables specifies the allowed succession of vertices in the graph. For instance, the Boolean function $x \vee y$ is represented by the BDD in Figure 1.



The function is described by the following set of paths:

$$\left\{ \begin{matrix} x \xrightarrow{0} y \xrightarrow{0} 0, x \xrightarrow{0} y \xrightarrow{1} 1 \\ x \xrightarrow{1} y \xrightarrow{0} 1, x \xrightarrow{1} y \xrightarrow{1} 1 \end{matrix} \right\}$$

Fig. 1. BDD for $x \vee y$ with variable order $x < y$, and its corresponding paths

We compare operations for two categories of DDs that have deep differences. First, operations on BDDs and other categories with a fixed number of variables are presented in Sec. 3. Then, Sec. 4 presents operations on DDDs where paths can use different variables, and even have different lengths. Sec.5 makes a synthesis of these two kinds, by proposing a new language for operations.

3 BDD-like Operations

BDD-like operations are used on DDs representing functions with a finite and homogeneous input domain. Their signatures are thus of the form $\mathbb{X}^n \rightarrow \mathbb{O}$, where $\mathbb{X}$ is a finite domain. This includes categories such as BDDs [3], ADDs [4], MDDs [9,10], *etc.* Their paths are therefore of the form $v_1 \xrightarrow{x_1} \dots v_n \xrightarrow{x_n} t$.

Several operations are defined for BDDs in [3]. We give their extension to finite edges [10] and terminal [4] domains. In practice, they are used in BDD libraries, such as CUDD[5].

The following operations form the BDD-like manipulation language. It is composed of functions, which take DDs (noted d_i), variables (v_i), input values (x_i), output values (t_i) or operators ($\odot$) as parameters. We give – informally – the result of each operation as its set of returned paths.

`constant(`t`)` creates a DD where all paths lead to the terminal value t. It represents the constant function that returns t whatever its inputs are.
`constant(`t`)`$= \left\{ v_1 \xrightarrow{x_1} \dots v_n \xrightarrow{x_n} t \right\}$[6]

`make(`v`,` x`)` creates a DD where only the variable v is relevant to the function's result. The paths where v has value x lead to terminal value 1, and others to terminal 0, even for DD categories with unbounded terminals such as ADDs. We explain this restriction in Problem 1.
`make(`v`,` x`)`$= \left\{ v_1 \xrightarrow{x_1} \dots v \xrightarrow{\mathbb{X}\backslash\{x\}} \dots v_n \xrightarrow{x_n} 0 \right\} \cup \left\{ v_1 \xrightarrow{x_1} \dots v \xrightarrow{x} \dots v_n \xrightarrow{x_n} 1 \right\}$

`apply(`$\odot$`,` d_l`,` d_r`)` computes $d_l \odot d_r$. It takes two DDs d_l and d_r representing functions of signature $\mathbb{X}^n \rightarrow \mathbb{O}$, and a binary operator $\odot : \mathbb{O} \times \mathbb{O} \rightarrow \mathbb{O}$ on their output domain. It returns the DD of the function computed by:
`apply(`$\odot$`,` d_l`,` d_r`)`$= \left\{ v_1 \xrightarrow{x_1} \dots v_n \xrightarrow{x_n} t_1 \odot t_2 \;\middle|\; \begin{matrix} v_1 \xrightarrow{x_1} \dots v_n \xrightarrow{x_n} t_1 \in d_l \\ v_1 \xrightarrow{x_1} \dots v_n \xrightarrow{x_n} t_2 \in d_r \end{matrix} \wedge \right\}$

[5] http://vlsi.colorado.edu/~fabio/CUDD/
[6] We omit universal quantification on x_i for readability, as in other operations.

`restrict(`v`, `x`, `d`)` restricts d to the value x of the variable v, and then ignores this variable's values. The returned function's result is similar to the original one, when v has value x, so this operation computes $f|_{v=x}$. The resulting DD still has all variables, including v, which has no effect on its result.
`restrict(`v`, `x`, `d`)`$= \left\{ v_1 \xrightarrow{x_1} \dots v \xrightarrow{\mathbb{X}} \dots v_n \xrightarrow{x_n} t \middle| v_1 \xrightarrow{x_1} \dots v \xrightarrow{x} \dots v_n \xrightarrow{x_n} t \in d \right\}$

`compose(`v`, `d_l`, `d_r`)` substitutes the variable v in d_l with the DD d_r. Both DDs d_l and d_r must represent functions of type $\mathbb{X}^n \to \mathbb{X}$. This operation can be expressed using `apply` and `restrict`, and does not appear in articles on BDDs other than [3].

`satisfy-all(`t`, `d`)` returns the set of paths with terminal value t. The result of this operation is *not* a DD but an iterable enumeration of paths.
`satisfy-all(`t`, `d`)`$= \left\{ v_1 \xrightarrow{x_1} \dots v_n \xrightarrow{x_n} t \in d \right\}$

`satisfy-one(`t`, `d`)` chooses one element from those returned by `satisfy-all`. Choice may be deterministic or not, depending on the implementation.

`satisfy-count(`t`, `d`)` Instead of returning paths, this operation counts them. Again, the result is *not* a DD, it is an integer.
`satisfy-count(`t`, `d`)`$= \left| \left\{ v_1 \xrightarrow{x_1} \dots v_n \xrightarrow{x_n} t \in d \right\} \right|$

The language for BDD-like operations is widely used, in many libraries and applications. The example given below[7] builds a BDD for $\overline{x} \wedge \overline{y} \wedge \overline{z}$. It then restricts the function for variable $x = 0$ and counts the paths leading to terminal 1. Results of operations are given as comments, after `//`. Notice that all operations, except `satisfy-count`, return a DD containing *all* variables x, y, z. Their declaration is required before executing the operation. As the operation is composed of nested function calls, it should be read from bottom to top. The call to `constant` is common in BDD programming, but not useful here.

```
satisfy-count // $\Longrightarrow 2$
( 1
, restrict // $\Longrightarrow x \xrightarrow{0,1} y \xrightarrow{0} z \xrightarrow{0} 1 \cup \cdots \to 0$
  ( $x$, 0
  , apply // $\Longrightarrow x \xrightarrow{0} y \xrightarrow{0} z \xrightarrow{0} 1 \cup \cdots \to 0$
    ( ∧
    , make($z$, 0) // $\Longrightarrow x \xrightarrow{0,1} y \xrightarrow{0,1} z \xrightarrow{0} 1 \cup x \xrightarrow{0,1} y \xrightarrow{0,1} z \xrightarrow{1} 0$
    , apply // $\Longrightarrow x \xrightarrow{0} y \xrightarrow{0} z \xrightarrow{0,1} 1 \cup \cdots \to 0$
      ( ∧
      , make($y$, 0) // $\Longrightarrow x \xrightarrow{0,1} y \xrightarrow{0} z \xrightarrow{0,1} 1 \cup x \xrightarrow{0,1} y \xrightarrow{1} z \xrightarrow{0,1} 0$
      , apply // $\Longrightarrow x \xrightarrow{0} y \xrightarrow{0,1} z \xrightarrow{0,1} 1 \cup x \xrightarrow{1} y \xrightarrow{0,1} z \xrightarrow{0,1} 0$
        ( ∧
        , make($x$, 0) // $\Longrightarrow x \xrightarrow{0} y \xrightarrow{0,1} z \xrightarrow{0,1} 1 \cup x \xrightarrow{1} y \xrightarrow{0,1} z \xrightarrow{0,1} 0$
        , constant(1) // $\Longrightarrow x \xrightarrow{0,1} y \xrightarrow{0,1} z \xrightarrow{0,1} 1$
) ) ) ) )
```

[7] Inspired by the documentation of CUDD.

This language for BDD-like operations has several drawbacks. We describe them in the remainder of this section. Note that these drawbacks do not prevent wide use of this language.

Problem 1 (`make` restricts terminal values). Creating a DD with this operation requires a default value for other paths. For instance, `make(`x`, 0)` creates a DD returning 1 whenever $x = 0$, but implicitly also 0 for the other cases.

Boolean functions have a trivial implicit result, as only two terminal values are available. But it is not the case for ADDs, which represent functions $\mathbb{B}^n \to \mathbb{N}$. Their implicit result is chosen as the identity element 0 of addition. With addition, all terminal values can be obtained, as in `apply(`$+$`, make(`x`, 0), make(`x`, 0))` which returns $x \xrightarrow{0} y \xrightarrow{0,1} z \xrightarrow{0,1} 2 \cup x \xrightarrow{1} y \xrightarrow{0,1} z \xrightarrow{0,1} 0$.

So, the set of terminal values should be a monoid for BDD-like operations. It requires defining a $+$ operation to generate all possible terminal values. Note that `constant` can then be also restricted to the terminals' generator only.

Problem 2 (The language is not minimalist). The `compose` operation is redundant. It can be expressed using two other operations, `apply` and `restrict`. Usually, this is not a problem, as redundant operations can be safely removed from the language to get a minimal one. But the operation is specifically defined because its algorithm is more efficient. So, this BDD-like language is not minimalist, but removing the redundant operation has an efficiency cost.

The same task can be defined in several ways, that are not equivalent regarding efficiency. Writing a complex operation therefore requires knowledge from the user on how its evaluation internally works.

Problem 3 (The language requires embedding). A usual operation in model checking is to compute the fixed point of next states computation. But the BDD-like language has no operations for iteration or recursion. So, it requires to be *embedded* into a general-purpose language, that can compute the fixed point.

The BDD-like languages are thus truly DSLs. They describe an operation in a concise and readable manner, but are not as expressive as a general-purpose programming language. They can express only very simple computations.

Problem 4 (Operations cannot be optimized). In [11] it has been shown that rewriting the operations can lead to huge performance improvements. This has been partially automated in DDD-like languages [2]. Because of embedding (see Problem 3), a library with BDD-like operations cannot see the whole operation to perform. It only processes it in small operations. Only few optimizations are available for these, whereas most improvements require access to the full syntax tree of the operation.

This problem is very similar to Problem 2, which occurs because the `compose` operation cannot be expressed efficiently enough using other operations. The language is not low-level enough to enable optimizations that could bring the same efficiency. Its operations take whole DDs as parameters and return a whole DD. So, the language lacks more intrusive operations, that apply to parts of the DDs. DDD-like operations presented in Section 4 define them, and have shown great optimizations thanks to lower-level operations.

4 DDD-like Operations

The main difference between BDD-like and DDD-like structures is the input domain of represented functions. In [5], paths are defined as "sequences of assignments", because in the same DD, paths can be of various lengths, and the same variable can appear several times along a path. We cannot describe the domain of functions represented by DDD-like structures with a cartesian product notation. We propose in [12] a specification of their domain.

These categories include DDDs [5] and SDDs [13] which represent sets and have Boolean terminals, and MultiSet Decision Diagrams (MSDDs) [14] which represent multisets with natural integer terminals. We propose in [12] a unification of BDD-like and DDD-like structures, where they represent functions on complex data types rather than functions with multiple arities. We do not give importance to these subtleties in this article, so readers only need to be aware of paths with different lengths and variable repetition.

DDD-like operations are partitioned in two languages: a language of binary set operations (Sec. 4.1) and a language of unary set transformation (Sec. 4.2). They are not totally disjoint, as bridges exist between them.

4.1 Binary Set Operations

The language of set expressions is composed of the usual binary set operations $\cup, \cap, \setminus$ – or their multiset counterparts – applied to two DDs. These three operations are distinct, whereas they are all covered by `apply` in BDD-like operations. A general definition is difficult, because domains of variables can be unbounded. To preserve the finite graph representation of the function, an infinite number of paths must lead to terminal 0, whereas a finite number lead to other terminals. So, the operation applied to terminals should always return 0 when both its operands are 0 to ensure termination. A general `apply` could exist, given operator on terminals respects this constraint.

These operations are a small subset of BDD-like operations. For instance, they are not sufficient to compute a state space. So, they are completed with a language for set transformations.

4.2 Unary Set Transformations

The unary set transformations, initially proposed in [5], have nothing in common with BDD-like operations. They are close to the `map` functions found in most functional languages: for a DD category representing sets, an operation is applied to each path. But instead of returning a transformed path, application on each path returns a set of paths, each set represented by a DD.

These transformations are homomorphisms. They must therefore enforce a constraint: all must be linear for a union $\cup$ operator defined in the DD category: $h(d_1 \cup d_2) = h(d_1) \cup h(d_2)$. The linearity property means that every input DD can be split in several DDs, then the operation applied to each, and the result obtained by union of their results. For simplicity, operations given below do not mention particular cases required by linearity. These cases are common to all operations, and are discussed in Problems 8 and 9. Note that these operations are applied on quasi-reduced DDs, *i.e.* without level skipping.

Unary transformations are parametric: they take parameters that define their behavior in addition to the DD to transform. To describe the language of such operations, we show each one with both kinds of parameters. The first one, between brackets, contains all parameters except the DD to transform. The second one, between parentheses, is this DD. We thus clearly distinguish between parameters that define the operation, and the parameter to which it is applied.

`terminal[`t`]` is the only operation that allows to create a DD outside unary set transformations. It is therefore crucial, as the basis for each operation. This nullary operation creates a DD terminal valued by t. It is close to the `constant` BDD-like operation. But they differ because the DD returned by `terminal` represents the nullary function that returns t, whereas the DD returned by `constant` represents a n-ary function.
`terminal[`t`]()`$= \{\mapsto t\}$

`constant[`d_c`]` returns the constant DD d_c, whatever its input DD is. This operation is parameterized by the DD to return, and is applied to a DD that it ignores (almost, see Problem 8). The name of this operation can be misleading. The returned DD represents a function that can be constant or not: all its paths do not necessarily lead to the same terminal. But the `constant` operation returns this DD independently of its input. It thus differs from `constant` of BDD-like operations, which returns a constant function.
`constant[`d_c`](`d`)` $= d_c$

`identity[]` returns its input with no modification. This operation can be expressed using other operations, but is introduced in DDDs for efficiency.
`identity[](`d`)` $= d$;

`make[`$v \xrightarrow{x}$`]` adds a prefix $v \xrightarrow{x}$ to a DD. It differs from the BDD-like `make` operations, because it adds a variable to the represented function given as parameter, whereas the BDD-like creates an n-ary function from scratch.

$$\texttt{make[}v \xrightarrow{x}\texttt{](}d\texttt{)}= \left\{ \begin{array}{c|c} v \xrightarrow{x} v_1 \xrightarrow{x_1} \dots v_n \xrightarrow{x_n} t & \\ v \xrightarrow{\{y \neq x\}} v_1 \xrightarrow{x_1} \dots v_n \xrightarrow{x_n} 0 & v_1 \xrightarrow{x_1} \dots v_n \xrightarrow{x_n} t \in d \end{array} \right\}$$

`match[`$v \xrightarrow{x}$`, `h`]` is the inverse operation of `make`. It selects paths where $v \xrightarrow{x}$ is a prefix of the DD, and then applies another operation h to the subDD that prefix leads to. Other paths return the identity DD.
`match[`$v \xrightarrow{x}$`, `h`](`d`)`$= h\left(\left\{v_1 \xrightarrow{x_1} \dots v_n \xrightarrow{x_n} t \;\middle|\; v \xrightarrow{x} v_1 \xrightarrow{x_1} \dots v_n \xrightarrow{x_n} t \in d\right\}\right)$
This operation is overloaded, as it also exists to match a terminal.
`match[`t`, `h`](`d`)`$= h\left(\{t \mid t \in d\}\right)$

`composition[`h_1`, `h_2`]` computes the composition $h_1 \circ h_2$, and applies it to its input DD. It has no relation with `compose` of BDD-like operations, which substitutes a variable with the result of a DD.
`composition[`h_1`, `h_2`](`d`)` $= (h_1 \circ h_2)(d) = h_1(h_2(d))$

`union[`h_1`, `h_2`]` (respectively `intersection`) computes the union (resp. intersection) of results of h_1 and h_2 applied to the input DD. This operation wraps some of the set operations presented in Sec. 4.1. Set difference $\setminus$ is not wrapped because it is not a linear operation. Note that these two operations can be easily extended to n-ary versions as they are associative and commutative.
`union[`h_1`, `h_2`]`$(d) = h_1(d) \cup h_2(d)$ `intersection[`h_1`, `h_2`]`$(d) = h_1(d) \cap h_2(d)$

`fixpoint[`h`]` has been introduced for model checking applications. Problem 3 shows that BDD-like languages lack operations for iteration. `fixpoint` provides such an operation. It computes the fixed point of h, applied to the input DD.
`fixpoint[`h`]`$(d) = h^\infty(d) = h(\ldots h(d) \ldots)$

The DDD-like unary set transformations are much more expressive than the BDD-like operations. Figure 2 shows a Petri net and an operation that computes its state space, which will be improved later. The operation uses the `fixpoint` operator where BDD-like operations require to be embedded in a general purpose language. To encode an least fixed point, `identity` is used to keep previously computed states.

We show how the operation works by applying the operation for transition t to the DDD path $q \xrightarrow{1} p \xrightarrow{2} r \xrightarrow{0} 1$ representing the initial state of the Petri net.

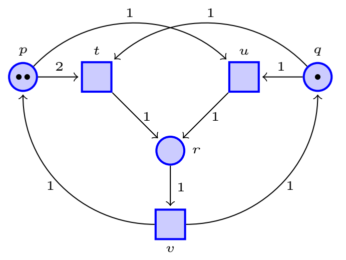
$$\begin{array}{rr}
 & \texttt{match}[q \xrightarrow{1}, \ldots] \; (q \xrightarrow{1} p \xrightarrow{2} r \xrightarrow{0} 1) \\
\Longrightarrow & \texttt{make}[q \xrightarrow{0}, \ldots] \; (p \xrightarrow{2} r \xrightarrow{0} 1) \\
\Longrightarrow & q \xrightarrow{0} \texttt{match}[p \xrightarrow{2}, \ldots] \; (p \xrightarrow{2} r \xrightarrow{0} 1) \\
\Longrightarrow & q \xrightarrow{0} \texttt{make}[p \xrightarrow{0}, \ldots] \; (r \xrightarrow{0} 1) \\
\Longrightarrow & q \xrightarrow{0} p \xrightarrow{0} \texttt{match}[r \xrightarrow{0}, \ldots] \; (r \xrightarrow{0} 1) \\
\Longrightarrow & q \xrightarrow{0} p \xrightarrow{0} \texttt{make}[r \xrightarrow{1}, \ldots] \; (1) \\
\Longrightarrow & q \xrightarrow{0} p \xrightarrow{0} r \xrightarrow{1} \texttt{match}[1, \ldots] \; (1) \\
\Longrightarrow & q \xrightarrow{0} p \xrightarrow{0} r \xrightarrow{1} \texttt{identity}[] \; (1) \\
\Longrightarrow & q \xrightarrow{0} p \xrightarrow{0} r \xrightarrow{1} 1
\end{array}$$

Problem 5 (Operations usually must be lazily defined). The operation given in Figure 2 has a bug, because the `match` operation for transition u requires exactly one token in p. So, it cannot be applied from the initial state. We have to enumerate all possible cases, by adding :

, `match [`$q \xrightarrow{1}$`, make [`$q \xrightarrow{0}$`, match [`$\boldsymbol{p} \xrightarrow{1}$`, make [`$p \xrightarrow{0}$`,`
`match [`$r \xrightarrow{0}$`, make [`$r \xrightarrow{1}$`, match [1, identity []...] // u`

We cannot express "add n tokens" or "remove n tokens", only "set to n tokens". In practice, an operation usually has an *a priori* unbounded number of `match` calls. They will be discovered during computation. Description of the operation would be infinite, so it is in practice rather defined lazily. Then `match` uses a function returning the operation associated with each encountered value.

Problem 6 (Lazy operations cannot be optimized). The operations that compute the state space of Petri net in Figure 2 can be optimized. The operation



```
fixpoint[
  union[
    identity
  , match[$q \xrightarrow{1}$, make[$q \xrightarrow{0}$, match[$p \xrightarrow{2}$, make[$p \xrightarrow{0}$,
    match[$r \xrightarrow{0}$, make[$r \xrightarrow{1}$, match[1, identity[]...] // t
  , match[$q \xrightarrow{1}$, make[$q \xrightarrow{0}$, match[$p \xrightarrow{1}$, make[$p \xrightarrow{0}$,
    match[$r \xrightarrow{0}$, make[$r \xrightarrow{1}$, match[1, identity[]...] // u
  , match[$q \xrightarrow{0}$, make[$q \xrightarrow{1}$, match[$p \xrightarrow{0}$, make[$p \xrightarrow{1}$,
    match[$r \xrightarrow{1}$, make[$r \xrightarrow{0}$, match[1, identity[]...] // v
] ]
```

Fig. 2. A Petri net and the operation encoding its firing rule. The variable order used by operations is $q < p < r$. Transitions' names are given as comments after their encoding. We use `]...]` instead of numerous closing brackets, so indentation is meaningful.

is rewritten to an equivalent but more efficient one, given below, by merging identical `match` calls. It is more efficient because some parts of the operation are merged, and thus require fewer application of the operations. The currently most advanced optimization technique, called "automatic saturation" has been introduced in [2]. It follows the same principle of rewriting an operation into a more efficient one.

```
fixpoint[
  union[
    identity
  , match[$q \xrightarrow{1}$, make[$q \xrightarrow{0}$, union[
      match[$p \xrightarrow{1}$, make[$p \xrightarrow{0}$, match[$r \xrightarrow{0}$, make[$r \xrightarrow{1}$,
      match[1, identity[]...]
    , match[$p \xrightarrow{2}$, union[
        make[$p \xrightarrow{0}$, match[$r \xrightarrow{0}$, make[$r \xrightarrow{1}$, match[1, identity[]...]
        make[$p \xrightarrow{1}$, match[$r \xrightarrow{0}$, make[$r \xrightarrow{1}$, match[1, identity[]...]
    ]...]
  , match[$q \xrightarrow{0}$, make[$q \xrightarrow{1}$, match[$p \xrightarrow{0}$, make[$p \xrightarrow{1}$,
    match[$r \xrightarrow{1}$, make[$r \xrightarrow{0}$, identity[]...]
] ]
```

Non-optimized and optimized ones are expressed in the same language. So, the language is suitable for optimization by program transformation. It solves the Problem 4 of BDD-like languages.

Lazy operations seen in Problem 5 are useful to deal with *a priori* unbounded structures. But they have a drawback: they cannot be optimized, because their suboperations are not known until execution. To circumvent the problem, new operations have been added in DDD-like languages. They are redundant with existing ones, but provide the missing information. However, as more optimizations are added, more specific operations like these must also be added. This approach creates several redundant ways to define the same operations. Users have to know which one is better to enable most optimizations.

Problem 7 (Weak typing). Each operation is designed to work on DDs of a given type, *i.e.* a variable order and a category that defines the variables' domains and output domain. For instance, the following operation can only be applied to DDDs of variable order $x < y$. It detects the path $x \xrightarrow{1} y \xrightarrow{1} 1$.

```
match[x -1->, match[y -1->, match[1, constant[terminal[1]]]]]
```

Operations' operands which are DDs are not explicitly typed. Only the user knows their type. So, an operation can discover at runtime that it is not applied to an intended operand. The operation then fails, but how this failure is reported is unclear in the DDD-like languages: return of a special terminal $\top$, assertion to stop the program, exception that can be caught. . . or even return of terminal 0 which is in fact silent.

Instead of giving a DD of the wrong type to an operation, typing errors can also appear because of DDs returned by inner operations. This leads to errors (known in DDDs as the "top" $\top$ terminal) that are hard to debug[8]. Such a typing error occurs when operands of `union` or `intersection` do not return DDs of the same type. For instance, the following operation is erroneous because the first part of `union` returns a DDD with variable order $x < y$ whereas the second has variable order $y < x$.

```
union[ make[x -1->, make[y -0->, constant[terminal[1]]]
     , make[y -1->, make[x -0->, constant[terminal[1]]]]]
```

It seems easy here to check the typing problems. But lazy operations, required for Problem 5, prevent checking in most practical cases.

Problem 8 (An identity DD is only transformed to an identity DD). The linearity constraint of homomorphisms requires that every DD can be split in several other DDs, operations applied to them, and their results merged.

Each DD can be decomposed to itself and the identity DD (noted d_0), where all paths lead to terminal 0. From the definition of linearity, a DD can be decomposed in itself and the identity, as in $h(d) = h(d \cup d_0) = h(d) \cup h(d_0)$. So, every operation must return the identity DD when applied to the identity d_0.

[8] Users that have used at least once a DDD library know the pain to look at thousands of lines of DD paths to search where something went wrong.

For instance, all operations applied to the DDD representing the empty set return the empty set. Even `constant`[d_c](d_0) cannot return its DD parameter d_c when applied to d_0. In this particular case, it must return the identity d_0. This constraint does not exist in BDD-like operations. They can return any DD when applied to the identity one.

This requirement prevents the user of DDD-like operations to start its application with the identity DD, and then filling it through operations. In model checking applications using DDDs, we have to start therefore with an initially non-empty set of states, which is usually the initial state.

Problem 9 (Operations on terminals must also be linear). This is another consequence of homomorphisms, as in Problem 8. Consider a DD representing multisets. Each path represents an element of the multiset, its terminal represents its cardinality. The multiset union sums the terminals of identical paths. Because each DD can be decomposed, the result computed for a path leading to n must be the same as the sum of results for the same path ending with n_1 and n_2, where $n = n_1 + n_2$.

For instance, Fig. 3 shows an operation that tries to count paths leading to terminal 1. Its result is computed by adding all subresults, given as destinations of red bold arrows. The final result is shown for a DD (left part) and for one arbitrary decomposition of the DD (right part). They differ, because the operation on terminals is not linear.

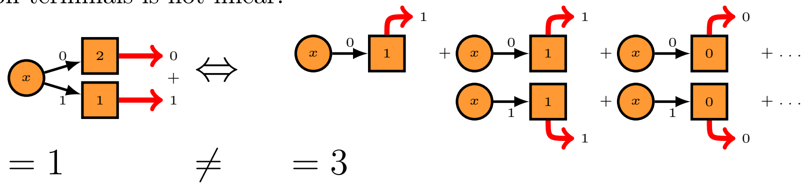


Fig. 3. Counting paths leading to terminal 1 is not a linear operation

Users specify operations only for paths leading to 1 in DDDs. This category has Boolean terminals, and homomorphisms always return the identity DD d_0 when paths end with 0. For DDs representing multisets (MSDDs in [14]), letting the user specify the operation for any non-zero terminal value is dangerous, as it might break linearity. To ensure this property, user should still define the operation only for terminal 1.

Problem 10 (Goal of restriction to linear operations is unclear). Even if we allow operations to return not only DDs, but also other data types such as integers, the linearity constraint is too restrictive for many operations. For instance, even a simple operation like counting the number of paths leading to a particular terminal value is *not* a linear operation (see Fig. 3 of Problem 9). DDD-like languages have an operation to count paths, but it cannot be used within linear operations. So, this special operation is doomed to appear only at the end of a computation. The reason why DDDs define their operations as homomorphisms is not explicitly given in [5]. They are well adapted for set transformation, but they restrict drastically the expressiveness of their language. We believe that the main justification for such constraints is efficiency.

Linearity, $h(d_1 \cup d_2) = h(d_1) \cup h(d_2)$, ensures not only that a DD can be decomposed to its paths to compute the operation's result, but also that the subresults can always be combined *before* the next computation. Consider the operation $(f \circ g)(d_1 \cup d_2)$. It can be rewritten as the union of two compositions $(f \circ g)(d_1) \cup (f \circ g)(d_2)$, or as $f(g(d_1) \cup f(g(d_2))$. In both cases, the result if the same, whether f is applied to two subresults $g(d_1)$ and $g(d_2)$, or to their union $g(d_1) \cup g(d_2)$. The second case is likely to do less computations[9] than the first one, thus saving time and memory.

This optimization is possible because of the linearity constraint. We propose in Sec. 5 to move this constraint where it should be: as an optimization enabled only when operations can be proven linear.

Problem 11 (Unary operations are not enough). Some operations, such as set union $\cup$, intersection $\cap$ or difference $\setminus$ for DDDs are not unary. As DDD-like set transformations are unary only, there is no way for a user to describe them inside the homomorphisms framework.

Restriction to unary functions is useful to define operations as combinators. For instance, `composition[match[`$x \xrightarrow{1}$`, identity], make[`$y \xrightarrow{0}$`, identity]]` is a unary operation, because each suboperation is also unary and returns exactly one result. The input DD argument of composed functions is implicit.

Composition of n-ary operations does not allow such shortcuts, because several links can exist between the inputs and outputs of composed operations. But n-ary operations are the general case, that is missing in DDD-like operations.

5 Generalized Operations

We propose in this section a language for generalized operations, that solves the problems identified in Sections 3 and 4. This language covers both BDD-like and DDD-like languages. Throughout the section, a simple program given in Fig. 4 is used as example. It counts the similar paths in two DDs. Note that the proposed operations are applied to quasi-reduced DDs, as in DDD-like unary operations. Missing parts must be rebuilt on-the-fly when the operations are applied.

Program A program is a set of *named* operations, like *check*, $check_y$, $check_{x<y}$ and *count* in Fig. 4. Naming enables recursion. There is therefore no need for a particular `fixpoint` operation, nor embedding loops in a general-purpose language (Problem 3). One of these named operations is called by the user.

Typed inputs and outputs Each named operation has *typed* inputs and outputs (Problem 7). For instance, *count*`(`l`: in` t`,` r`: in` t`,` o`: out` *int*`)` has two inputs l and r of the same type t, and one output o of type *int*. To handle n-ary operations (Problem 11), several – or no – inputs and outputs are possible. All of them are DDs, for homogeneity, even values (*int*) are considered as terminals. Operand types are defined by a DD category (BDD, DDD, *etc.*) and a variable order. We

[9] But in DDs, who knows?

```
check(l: in v, r: in v, o: out int) =
  match | l: |t|, r: |t| ⇒ o ←make |1|
        | l: |t|, r: |u| ⇒ o ←make |0|
check_y(l: in u, r: in u, o: out int) =
  match | $l : y \xrightarrow{v} d_l, r : y \xrightarrow{v} d_r \Rightarrow check(l \leftarrow d_l, r \leftarrow d_r, o \rightarrow o)$
        | $l : y \xrightarrow{u} d_l, r : y \xrightarrow{v} d_r \Rightarrow o$ ←make |0|
check_{x<y}(l: in t, r: in t, o: out int) =
  match | $l : x \xrightarrow{v} d_l, r : x \xrightarrow{v} d_r \Rightarrow check_y(l \leftarrow d_l, r \leftarrow d_r, o \rightarrow o)$
        | $l : x \xrightarrow{u} d_l, r : x \xrightarrow{v} d_r \Rightarrow o$ ←make |0|
count(l: in t, r: in t, o: out int) =
  merge o from $check_{x<y}(l \leftarrow l, r \leftarrow r, o)$ with +
```

Fig. 4. Program that counts the similar paths in two DDs with variables $x < y$

propose in [12] a way to define types with an automaton describing the domains of variables and their order.

We use four types of DDs in our example. Three types correspond to DDs with Boolean terminals and unbounded variables: no variable (type v), variable y (type u) and variables $x < y$ (type t). The fourth type corresponds to DDs with natural terminals and no variables (type *int*), used to count the results.

Predefined operation templates Operations can be of three kinds, that we call "operation templates". They are inspired from DDD-like operations, but are minimalist as each one has a very specific role. They thus solve Problem 2.

`match | ` $prefix^n \Rightarrow call$ `|` . . . takes as parameter a finite mapping from prefix patterns to operation calls. As we deal with n-ary operations, a pattern is composed of n prefixes, one for each input operand. For each set of DD paths, one taken from each operand, this operation finds the first matching pattern, and applies the associated operation. A prefix can be:

– a terminal constant or placeholder[10] for instance |1| or |t|,
– a DD placeholder (which also covers a terminal), for instance d,
– a prefix with constants or placeholders as variable or/and edge values, and another prefix for the successor, for instance $1 \xrightarrow{x} d$, $x \xrightarrow{0} d$, or $x \xrightarrow{y}$|t|.

As in functional languages with pattern-matching, all the patterns of a `match` must cover all possible prefixes of the input DDs. They also must bind all the outputs of the operation. The following binary operation, which uses the value of x as discriminant, is only valid when x has a Boolean domain.

`match |` $d_1 : x \xrightarrow{1} d, d_2 : x \xrightarrow{y} d' \Rightarrow \ldots$ `|` $d_1 : x \xrightarrow{0} d, d_2 : x \xrightarrow{y} d' \Rightarrow \ldots$

d_i are the names of input parameters that are matched against a prefix, and "..." is here the next operation to perform (not given). Scope of placeholders is in their pattern, so the two ys are distinct in the previous example, but the two xs in a pattern $x \xrightarrow{x}$ have the same value. The following unary operation is valid even for unbounded domains, as placeholder y can take any value.

[10] We name terminal placeholder a language variable that can store a terminal value. The term "variable" is already used in DDs, so we do not use it to avoid ambiguities.

`match` | $d_1 : x \xrightarrow{x} d \Rightarrow \ldots$ | $d_1 : x \xrightarrow{y} d \Rightarrow \ldots$

Note that patterns overlap in the previous examples. We chose the usual policy of functional languages: only the first matching pattern is applied. This is useful to define default cases.

Operation calls bind inputs and outputs of caller and callee. We can wrap a placeholder with an expression, to compute a value from its content. However, we do not describe expressions here, by lack of space. So, placeholders are simply bound to inputs and outputs of operations.

`make` | *pattern* | … creates a DD from patterns using constants and placeholders. Patterns are expressed in the same way as in `match`. They must also cover all possible prefixes of created DD. In Fig.4, we only create terminals, but this operation can also create DD nodes. For instance, the code below creates the DD $x \xrightarrow{0} 0 \cup x \xrightarrow{1} 1$, when d is a placeholder holding the terminal 0, x is a variable and y is a free placeholder of Boolean domain. Placeholders used in the patterns can be bound to a value or be free. The latter case is useful to fill a whole unbounded domain.

`make` | $x \xrightarrow{0} d$ | $x \xrightarrow{y}$ |1|

As patterns may overlap, we chose a policy consistent with `match`: when a prefix can be created using several patterns, the first one prevails. Note that this operation template does not require a default value, and thus solves Problem 1.

`merge` *placeholder* `from` *call* `with` *operator* is a new operation. It is required because there is no more a global $\cup$ operator as in DDD-like languages. Each `match` operation does not return a single DD, but a multiset of DDs, because it generates one or several DDs for each matching prefix. `merge` maps operations to how their results should be merged. The operator used to merge results (+ in Fig 4) is specified only for terminals, following the BDD-like `apply` mechanism. It is applied on terminals of paths that differ only by their terminal values.

This operator must be associative and commutative, because there is no order on generated DDs. Moreover, when dealing with unbounded domains, it must have an identity element. Homomorphisms are a special case, where the merge operator respects linearity.

6 Conclusion

This article does a critical comparison of BDD-like and DDD-like manipulation languages. It also proposes a new language for manipulation of DDs, that takes inspiration from both BDD-like languages and DDD-like languages, and covers them. This language also enables more operations. We give the example of counting paths that are similar in two DDs. This operation cannot be defined in the BDD-like and DDD-like languages, but is available in our proposal. The proposed language solves the problems we identify, such as minimalism of the language or expressiveness. It provides a small number of operations that have been carefully designed to allow future optimizations and are general enough to handle n-ary typed operations.

We plan to extend the proposed language to hierarchical DDs, as they are useful DD categories. Moreover, we have to redefine the optimizations that already exist for SDDs. Adding some kind of genericity to the language is also required, because the proposed specification of types can prevent code reuse.

References

1. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic Model Checking: 10^{20} States and Beyond. In: LICS '90: 5th Annual IEEE Symposium on Logic in Computer Science. (1990) 428–439
2. Hamez, A., Thierry-Mieg, Y., Kordon, F.: Hierarchical Set Decision Diagrams and Automatic Saturation. In: Petri Nets '08: 29th International Conference on Applications and Theory of Petri Nets. (2008) 211–230
3. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers **35**(8) (1986) 677–691
4. Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., Somenzi, F.: Algebraic Decision Diagrams and Their Applications. Formal Methods in System Design **10**(2/3) (1993) 188–191
5. Couvreur, J.M., Encrenaz, E., Paviot-Adet, E., Poitrenaud, D., Wacrenier, P.A.: Data Decision Diagrams for Petri Net analysis. In: ICATPN '02: 23rd International Conference on Applications and Theory of Petri Nets. (2002) 101–120
6. Lai, Y., Sastry, S.: Edge-Valued Binary Decision Diagrams for Multi-Level Hierarchical Verification. In: DAC '92: 29th ACM/IEEE conference on Design automation. (1992) 608–613
7. Ciardo, G., Siminiceanu, R.: Using Edge-Valued Decision Diagrams for Symbolic Generation of Shortest Paths. In: FMCAD '02 : 4th International Conference on Formal Methods in Computer-Aided Design. (2002) 256–273
8. Strehl, K., Thiele, L.: Symbolic Model Checking of Process Networks Using Interval Diagram Techniques. In: ICCAD '98: 1998 IEEE/ACM international conference on Computer-aided design. (1998) 686–692
9. Srinivasan, A., Kam, T., Malik, S., Brayton, R.K.: Algorithms for Discrete Function Manipulation. In: ICCAD '90: International Conference on CAD. (1990) 92–95
10. Corella, F., Zhou, Z., Song, X., Langevin, M., Eduard, C.: Multiway Decision Graphs for Automated Hardware Verification. Formal Methods in System Design **10**(1) (1997) 7–46
11. Ciardo, G., Lüttgen, G., Siminiceanu, R.: Saturation: An Efficient Iteration Strategy for Symbolic State-Space Generation. In: TACAS '01: Tools and Algorithms for the Construction and Analysis of Systems. (2001) 328–342
12. Linard, A., Paviot-Adet, E., Kordon, F., Buchs, D., Charron, S.: polyDD: Towards a Framework Generalizing Decision Diagrams. In: ACSD'10: 10th International Conference on Application of Concurrency to System Design. (2010)
13. Couvreur, J.M., Thierry-Mieg, Y.: Hierarchical Decision Diagrams to Exploit Model Structure. In: FORTE '05 : Formal Techniques for Networked and Distributed Systems. (2005) 443–457
14. Lucio, L., Hostettler, S.: Multi-set decision diagrams. Technical Report 205, Centre Universitaire D'Informatique, Université de Genève (2009) Available as http://smv.unige.ch/technical-reports/TR205-MSDD.pdf.

Towards Automatic Composition of Web Services: A SAT-Based Phase [⋆]

Wojciech Penczek[1,2], Agata Półrola[3], and Andrzej Zbrzezny[4]

[1] Polish Academy of Sciences, ICS, Ordona 21, 01-237 Warsaw, Poland
[2] University of Podlasie, ICS, Sienkiewicza 51, 08-110 Siedlce, Poland
penczek@ipipan.waw.pl
[3] University of Łódź, FMCS, Banacha 22, 90-238 Łódź, Poland
polrola@wmi.uni.lodz.pl
[4] Jan Długosz University, IMCS, Armii Krajowej 13/15, 42-200 Częstochowa, Poland
a.zbrzezny@ajd.czest.pl

Abstract. Automating the composition of web services is an object of a growing interest nowadays. In our former paper [3] we proposed a method for converting the problem of the composition to the problem of building a graph of worlds consisting of formally defined objects, and presented the first phase of this composition resulting in building a graph of types of services (an *abstract graph*). In this work we propose a method of replacing abstract flows of this graph by sequences of concrete services able to satisfy the user's request. The method is based on SAT-based reachability checking for (timed) automata with discrete data and parametric assignments.

1 Introduction

In recent years there has been a growing interest in automating the composition of web services. The number of more and more complex Internet services is still growing nowadays; several standards describe how services can be invoked (WSDL [17]), how they exchange information (SOAP [13]), how they synchronise the executions in complex flows (BPEL [16]), and finally how they can be discovered (UDDI [15]). However, still there is a lack of automatic methods for arranging and executing their flows. One of the problems to deal with is the size of the environment - most existing composition methods work with concrete instances of web services, so even a simple query requires taking all the instances of all the types of services into account. Another problem follows from incompatibilities in inputs/outputs of services, and difficulties in comparing their capabilities and qualities - two services can offer the same functionality, but this fact cannot be detected automatically without unification of their interfaces made by the providers.

In our work [3] we proposed an approach to automatic composition of services which can potentially solve the above problems. The problem of automatic composition of web services is converted to the problem of building a graph of worlds consisting of

[⋆] Supported by the Polish National Science Center under the decision DEC-2011/01/B/ST6/01477

formally defined objects, which are transformed by services. We introduce a uniform semantic description of service types. In order to adapt a possibly wide class of existing services, specific interfaces of concrete services are to be translated to the common one by adapters (called *proxies*), built in the process of service registration. The process is to be based on descriptions of interfaces of services, specified both in WSDL and in the languages containing a semantic information (like OWL-S or Entish [1]). The client's goal is expressed in a fully declarative intention language. The user describes two worlds: the initial and the final one, using the notions coming from an ontology, and not knowing any relations between them or between the services. The task of the composition system consists in finding a way of transforming the initial world into the final one. The composition is three-phase. In the first phase, called *abstract planning* or *planning in types*, we create an *abstract plan*, which shows sequences of service types whose executions possibly allow to accomplish the goal. The second phase makes these scenarios "concrete", which means replacing the types of services by their concrete instances. This can also involve choosing a plan which is optimal from the user's point of view. Finally, the last phase consists in supervising the execution of the optimal run, with a possibility of correcting it in the case of a service failure.

Our previous paper [3] described a method of generating an abstract graph of services. In the current work we deal with the second phase of composition: concretising abstract flows, i.e., with searching for sequences of concrete services which can lead to satisfying user's request. We apply model checking techniques to this aim. The substage aimed at choosing an optimal scenario is not considered in this version of the approach.

The rest of the paper is organised as follows. In Sec. 2 we present the related work. Sec. 3 introduces worlds and services transforming them. Sec. 4 describes briefly the abstract planning phase and its result. Next, in Sec. 5 we present our approach to SAT-based concretising abstract scenarios. Sec. 6 show experimental results and concluding remarks.

2 Related Work

There are many papers dealing with the topic of web services composition [4, 7–10, 14]. Some of these works consider static approaches, where flows are given as a part of the input, while the others deal with dynamically created flows. One of the most active research areas is a group of methods referred to as AI Planning [4]. Several approaches use Planning Domain Definition Language (PDDL [5]). Another group of methods is built around the so-called rule-based planning, where composite services are generated from high-level declarative descriptions, and compositionality rules describe the conditions under which two services are composable. The information obtained is then processed by some designated tools. The project SWORD [6] uses an entity-relation formalism to specify web services. The services are specified using pre- and postconditions; a service is represented as a Horn rule denoting that the postcondition is achieved when the preconditions are true. A rule-based expert system generates a plan. Another methodology is the logic-based program synthesis [8]. Definitions of web services and user requirements, specified in DAML-S [2], are translated to formulas of

Linear Logic (LL): the descriptions of web services are encoded as LL axioms, while a requirement is specified as a sequent to be proven by the axioms. Then, a theorem prover determines if such a proof exists.

Besides the automatic approaches mentioned above, there exist also half-automatic methods assuming human assistance at certain stages [12]. Some approaches are based on specifying a general plan of composition manually; the plan is then refined and updated in an automatic way.

Inspired by the Entish project [1], our approach enables to model automated composition based on matching input and output types of services. We adapt also the idea of three-phase composition, but introduce original definitions of services and composition techniques.

3 Worlds and Services

In our approach we introduce a unified semantics for functionalities offered by services, which is done by defining a dictionary of notions/types describing their inputs and outputs. A service is then understood as a function which transforms a set of data into another set of data (or as a transition between them). The sets of data are called *worlds*. The worlds can be described by the use of an *ontology*, i.e., a formal representation of a knowledge about them.

Definition 1 (World and objects). *The* universum *is the set of all the objects. The objects have the following features:*

- *each object is either a* concrete object *or an* abstract object*,*
- *each object contains named attributes whose values are either other objects or:*
  - *values of simple types (numbers, strings, boolean values; called* simple attributes*) or* `NULL` *(*empty value*) for concrete objects,*
  - *values from the set* {`NULL`, `SET`, `ANY`} *for abstract objects.*

  If an attribute `A` *of the object* O *is an object itself, then* O *is extended by all the attributes of* `A` *(of the names obtained by adding* `A`*'s name as a prefix). Moreover, when an object having an object attribute is created, its subobject is created as well, with all the attributes set to* `NULL`*.*
- *each simple attribute has a boolean-valued flag* `const`*.*

A world *is a set of objects chosen from the universum. Each object in a world is identified by a unique name.*

By default each `const` flag is set to `false`. If the flag of an attribute is `true`, then performing on the object any operation (service) which sets this attribute (including services initialising it) is not allowed (the value of the attribute is considered to be final). The attributes are referred to by `ObjectName.AttributeName`.

Definition 2 (Object state, world state). *A* state of an object `O` *is a function* V_o *assigning values to all the attributes of* `O` *(i.e., is the set of pairs* (`AttributeName`, `AttributeValue`)*, where* `AttributeName` *ranges over all the attributes of* `O`*). A* state of a world *is defined as the state of all the objects of this world.*

In order to reason about worlds and their states we define the following two-argument functions (the second default argument of these functions is the world we are reasoning about):

- `Exists` - a function whose first argument is an object, and which says whether the object exists in the world,
- `isSet` - a function whose first argument is an attribute of an object, and which says whether the attribute is set (has a nonempty value),
- `isConst` - a function whose first argument can be either an attribute or an object. When called for an attribute, the function returns the value of its `const` flag; when called for an object it returns the conjunction of the `const` flags of all the attributes of this object.

The ontologies collect the knowledge not only about the structure of worlds, but also about the ways they can be transformed, i.e., about services. The services are organised in a hierarchy of classes, and described both on the level of classes (by specifying what all the services of a given class do - such a pattern of behaviour is referred to as an *abstract service* or a *metaservice*), and on the level of objects (*concrete services*). The description of a service includes, besides specifying input and output data types, also declaration of introducing certain changes to a world, i.e., of creating, removing and modifying objects. The definition of a service is as follows:

Definition 3 (Service). *A* service *is an object of a non-abstract subclass*[1] *of the abstract class* `Service`*. A service contains (initialised) attributes, inherited from the base class* `Service`*. The attributes can be grouped into*

- processing lists *(the attributes* `produces`*,* `consumes`*,* `requires`*),*
- modification lists *(the attributes* `mustSet`*,* `maySet`*,* `mustSetConst`*,* `maySetConst`*), and*
- validation formulas *(the attributes* `preCondition` *and* `postCondition`*).*

Moreover, a service can contain a set of quality attributes.

A service modifies (transforms) a world, as well as the world's state. The world to be transformed by a service is called its *pre-world* (*input world*), while the result of the execution is called a *post-world* (*output world*). Modifying a world consists in modifying a subset of its objects. The objects being transformed by one service cannot be modified by another one at the same time (i.e., transforming objects is an atomic activity). A world consisting of a number of objects can be transformed into a new state in two ways[2]: by a service which operates on a subset of its elements, or by many services which operate concurrently on disjoint subsets of its elements.

The groups of attributes are presented in the definitions below.

[1] We use the standard terminology of object-oriented programming. The term "subclass" is related to inheritance. A class is called abstract if instantiating it (i.e., creating objects following the class definition) is useless, in the sense that the objects obtained this way do not correspond to any real-world entity.

[2] Services which create new objects are not taken into account.

Definition 4 (Processing lists). *The* processing lists *are as follows:*

– `produces` - *a list of named objects of classes whose instances are created by the service in the post-world,*
– `consumes` - *a list of named objects of classes whose objects are taken from the input world, and do not exist in the world resulting from the service execution (the service removes them from the world),*
– `requires` - *a list of named objects of classes whose instances are required to exist in the current world to invoke the service and are still present in the output world.*

The formal parameters from the above lists define an alphabet for modification lists and validation formulas.

Definition 5 (Modification lists). *The* modification lists *are as follows:*

– `mustSet` - *a list of attributes of objects occurring in the lists* `produces` *and* `requires` *of a service, which are obligatorily set (assigned a nonempty value) by this service,*
– `maySet` - *a list of attributes of objects occurring in the lists* `produces` *and* `requires` *of a service, which may (but not must) be set by this service,*
– `mustSetConst` - *a list of attributes of the objects which occur in the lists* `produces` *and* `requires` *of a service, which are obligatorily set as being constant in the worlds after executing this service,*
– `maySetConst` - *a list as above, but of the attributes which may be set as constant.*

A grammar for the above lists can be found in [3]. The attributes of the objects appearing in processing lists which do not belong to the union of lists `mustSet` and `maySet` are not changed when the service is called.

Definition 6 (Validation formulas). *The* validation formulas *are as follows:*

– `preCondition` - *a formula which describes the condition under which the service can be invoked. It consists of atomic predicates over the names of objects from the lists* `consumes` *and* `requires` *of the service and over their attributes, and is written in the language of the first order calculus without quantification (atomic predicates with conjunction, disjunction and negation connectives). The language of atomic predicates contains comparisons of expressions over attributes with constants, and functions calls with object names and attributes as arguments. In particular, it contains calls of the functions* `isSet`, `isConst` *and* `Exists`[3].
– `postCondition` - *a formula which specifies conditions satisfied by the world resulting from invoking the service. The formula consists of atomic predicates over the names of objects from the lists* `consumes`, `produces` *and* `requires` *of the service and over their attributes. To the objects and attributes one can apply pseudofunctions* `pre` *and* `post` *which refer to the state of an object or an attribute in the*

[3] Using `Exists` in `preCondition` is redundant w.r.t. using an appropriate object in the list `consumes` or `requires`. However, the future directions of developing the service description language mentioned in the final part of the paper, include moving modification lists to validation formulas.

input and the output world of this service, respectively. By default, the attributes of objects listed in `consumes` *refer to the state of the pre-world, whereas these in* `produces` *and* `requires` *- to the state of the post-world.*

Definition 7. *A service* `u` *is* enabled *(*executable*) in the current state of a world* `s` *if:*

- *each object* `o` *from the lists* `consumes` *and* `requires` *of* `u` *can be mapped onto an object in* `s`*, of the class of* `o` *or of its subclass; the mapping is such that each object in* `s` *corresponds to at most one object from the above lists;*
- *for the objects in* `s` *which, according to the above mapping, are actual values of the parameters in* `consumes` *and* `requires` *the formula* `preCondition` *of* `u` *holds,*
- *the list* `mustSet` *of* `u` *contains no attributes for which, in objects which are actual values of the parameters, the flag* `const` *is set.*

Definition 8. *A service* `u` *executable at the current world* `s` *produces a new world* `s'` *in which:*

- *there are all the objects from* `s`*, besides these which in the mapping done for executing* `u` *were actual values for the parameters in* `consumes`*,*
- *there is a one-to-one mapping between all the other objects in* `s'` *and the objects in the list* `produces` *of* `u`*, such that each object* `o` *from the list* `produces` *corresponds to an object in* `s'` *which is of a (sub)class of* `o`*;*
- *for the objects which, according to the above mappings, are actual values of the parameters in the processing lists the formula* `postCondition` *holds,*
- *in the objects which are actual values of the appropriate parameters the flags* `const` *of the attributes listed in* `mustSetConst` *of* `u` *are set, and the attributes listed in* `mustSet` *of* `u` *have nonempty values,*
- *assuming the actual values of the parameters as above, all the attributes of all the objects existing both in* `s` *and in* `s'` *which do not occur neither in* `mustSet` *nor in* `maySet` *have the same values as in the world* `s`*; the same holds for the flags* `const` *of the attributes which do not occur neither in* `mustsetConst` *nor in* `maySetConst`*. Moreover, all the attributes listed in* `mustSet` *or* `maySet` *which are of nonempty values in* `s`*, in* `s'` *are of nonempty values as well.*

3.1 Concrete Services

We assume here that concrete services present their offers in their pre- and postconditions. and that their `maySetConst` and `maySet` lists are empty (i.e., setting an attribute or the `Const` flag optionally is not allowed in this case).

A grammar for validation formulas is as follows:

```
<objectName>        ::=  <objectName from consumes> |
                         pre(<objectName from requires>) |
                         post(objectName from requires>) |
                         <objectName from produces>
<objectAttribute>   ::=  <objectName>.<attributeName> |
                         <objectName>.<objectAttribute>
<expressionElement> ::=  "integer value" | "real value" |
                         <objectAttribute> "of a numeric type"
<arithmOp>          ::=   + | - | * | /
<expression>        ::=  <expressionElement> |
                         <expression> <arithmOp> <expression>
<compOp>            ::=  = | < | <= | > | >=
<atomicPredicate>   ::=  Exists(<objectName>) |
                         isSet(<objectAttribute>) |
                         isConst(<objectAttribute>) |
                         not <atomicPredicate> |
                         <objectAttribute> <compOp> <expression>|
                         <objectAttribute> <compOp>  "value"
<conjunction>       ::=  <atomicPredicate> |
                         <atomicPredicate> and <conjunction>
<validationFormula> ::=  <conjunction> |
                         <conjunction> or <validationFormula>
```

It should be noticed that `pre()` and `post()` are allowed in `postCondition` only. Moreover, in this paper we assume that the expressions involve only attributes which refer either to names of objects from `consumes`, or to the names of objects which are of the form `pre(objectName from requires)` (i.e., that the expressions involve only values of attributes in the input world of a service). We assume also that all the elements of an expression are of the same type, the result is of this type as well, and so is the attribute this result is compared with. The expressions involve attributes of numeric types only, while the atomic predicates allow comparing an attribute of an arbitrary type with a value of the same type[4].

The values of the attributes and the values occurring in comparisons in the atomic predicates above are as follows:

- boolean values,
- integer values of a certain range[5],
- characters,
- real values of a certain range, with the precision limited to a number of decimal places (usually two),
- values of certain enumeration types.

Enumeration types are used instead of strings. In fact, such an approach seems sufficient to represent the values necessary: in most cases the names of items offered or processed by services come from a certain set of known names (e.g. names of countries, cities, names of washing machines types etc), or can be derived from the repository (e.g. names of shops which registered their offers). Similarly, restricting the precision of real values seems reasonable (usually two decimal places are sufficient to express the amount of a ware we buy, a price, a capacity etc). Consequently, all the values considered can be treated as discrete. It should be noticed also that we assume an ordering on the elements of enumeration types and the boolean values[6].

[4] The grammar for validation formulas is given in a semi-formal way. The “quoted” items should be understood as notions from the natural language. By `"value"` we mean a value of an arbitrary (also non-numeric) type.

[5] A natural restriction when using programming languages.

[6] Similarly as in the Ada programming language.

4 Abstract Planning

The aim of the composition process is to find a sequence of services whose execution can satisfy a user's goal. The user describes its goal in a declarative language defined by the ontology. He specifies (possibly partially) an initial and a final (desired) world, possibly giving also some evaluation criteria. The query is defined in the following way:

Definition 9 (Query). *A* query *consists of the following elements:*

- *an* initial domain - *a list of named objects which are elements of the initial world. The form of the list is analogous to the form of the list* `produces` *in the description of a service;*
- *an* initial clause *specifying a condition which is to be satisfied by the initial world. The clause is a formula over the names of objects and their attributes, taken from the initial domain. The grammar of the clause is analogous to the grammar of the* `preCondition`*;*
- *an* effect domain - *a list of named objects which have to be present in a final world (i.e., a subset the final world must contain);*
- *an* effect clause *specifying a condition which is to be satisfied by the final world. The clause is a formula over the names of objects and their attributes from both the domains defined above; references to the initial state of an object, if ambiguous, are specified using the notations* `pre(objectName)` *and* `post(objectName)`*, analogously as in the language used in the formulas* `postCondition` *of services. The grammar of the effect clause is analogous to the grammar of the* `postCondition`*;*
- *an* execution condition - *a formula built over services (unknown to the user when specifying the query) from a potential run performing the required transformation of the initial world into a target world. While construction of this formula simple methods of quantification and aggregation are used;*
- *a* quality function - *a real-valued function over the initial world, the final world and services in a run, which specifies a user's criterion of valuating the quality of runs. The run of the smallest value of this function is considered to be the best one.*

The last two parts of a query are used after finishing both the abstract planning phase and the first part of concrete planning, which adjusts types and analyses pre- and post-conditions of concrete services.

The aim of a composition process is to find a path in the graph of all the possible transitions between worlds which leads from a given initial world to a given final world, specified (possibly partially) in a user's query, using no other knowledge than that contained in the ontology. The composition is three-phase; the first phase (described in [3]) consists in finding all the sequences of service types (abstract services) which can potentially lead to satisfying the user's goal. The result of the abstract planning phase is an *abstract graph*.

The abstract graph is a directed multigraph. The nodes of the graph are worlds in certain states, while its edges are labelled by services. Notice that such a labelling carries an information which part of a input world (node) is transformed by a given service (that is specified by actual values of the parameters in `consumes` and `requires` of the service), and which part of the output world (node) it affects (the lists `produces` and `requires` of this service). We distinguish some nodes of the graph - these which have no input edges represent alternative initial worlds, while these with no output edges are alternative final worlds. A formal definition of the abstract graph is as follows:

Definition 10. *An* abstract graph *is a tuple $GA = (V, V_p, V_k, E, L)$, where V is a subset of the set S of all the worlds, $V_p \subseteq V$ is a set of initial nodes, $V_k \subseteq V$ is a set of final nodes, and $E \subseteq V \times V$ is a transition relation s.t. $e = (v, v') \in E$ iff $L(e)$ transforms the world v into v', where $L : E \longrightarrow U$ is a function labelling the edges with services.*

5 Main Idea

From the phase of abstract composition [3] we get a graph showing the sequences of service types which can potentially lead to satisfying user's request. The next step towards obtaining a flow to be run is to find concrete services of the appropriate types whose offers enable satisfying the query. We use SAT-based bounded model checking to this aim. In the paper [19] we have shown how to test reachability for timed automata with discrete data using BMC and the model checker VerICS. We adapt the above approach.

The main idea of our solution consists in translating each path of the abstract graph to a timed automaton with discrete data and parametric assignments (TADDPA). The automaton represents concrete services of appropriate types (corresponding to the types of services in the scenario we are working on) which can potentially be executed to reach the goal. The variables of the automaton store the values of the attributes of the objects occurring along the path, while the parameters are assigned to variables when the exact value assigned by a service is unknown. Next, we test reachability of a state satisfying the user's query. If such a state is reachable, we get a reachability witness, containing both an information about a sequence of concrete services to be executed to reach the goal and the values of parameters for which this sequence is executable.

In spite of using timed automata we currently do not make use of the timing part of this formalism, but the reason for using them is twofold. Firstly, doing this allowed us to adapt the existing implementation for timed automata with discrete data (modified to handle their extension - TADDPA). Secondly, in the future we are going to use the clocks to represent the declared times of services executions, which should enable us searching for scenarios of an appropriate timed length.

Below, we introduce all the elements of our approach.

5.1 Timed Automata with Discrete Data and Parametric Assignments

Given a set of discrete types $\mathcal{T} = \bigcup_{i=1,\ldots,n} T_i$ ($n \in \mathbb{N}$), including an integer type, a character type, user-defined enumeration types, a real type of a precision given etc., such that for any $T_i \in \mathcal{T}$ there is an ordering[7] on the values of T_i. By $\mathcal{T}_N \subset \mathcal{T}$ we denote the subset of $\mathcal{T}$ containing all the numeric types $T \in \mathcal{T}$. Let DV be a finite set of variables whose types belong to $\mathcal{T}$, and let DP be a finite set of parameters whose

[7] Similarly as in some programming languages, e.g. the Ada language.

types belong to $\mathcal{T}$. Let $type(a)$, for $a \in DV \cup DP$, denote the type of a. The sets of *arithmetic expressions* over T for $T \in \mathcal{T}_N$, denoted $Expr(T)$, are defined by

$$expr ::= c \mid v \mid expr \otimes expr,$$

where $c \in T$, $v \in DV$ with $type(v) = T$, and $\otimes \in \{+, -, *, /\}$. By $type(expr)$ we denote the type of all the components of the expression and therefore the type of the result[8]. Moreover, we define $Expr(\mathcal{T}) = \bigcup_{T \in \mathcal{T}_N} Expr(T)$.

The set of *boolean expressions* over DV, denoted $BoE(DV)$, is defined by

$$\beta ::= true \mid v \sim c \mid v \sim v' \mid expr \sim expr' \mid \beta \wedge \beta \mid \beta \vee \beta \mid \neg\beta,$$

where $v, v' \in DV$, $c \in type(v)$, $type(v') = type(v)$, $expr, expr' \in Expr(\mathcal{T})$, $type(expr) = type(expr')$, and $\sim \in \{=, \neq, <, \leq, \geq, >\}$.

The set of *instructions* over DV and DP, denoted $Ins(DV, DP)$, is given by

$$\alpha ::= \epsilon \mid v := c \mid v := p \mid v := v' \mid v := expr \mid \alpha\alpha,$$

where ϵ denotes the empty sequence, $v, v' \in DV$, $c \in type(v)$, $p \in DP$ and $type(p) = type(v)$, $type(v') = type(v)$, $expr \in Expr(\mathcal{T})$, and $type(expr) = type(v)$[9]. Thus, an instruction over DV is either an *atomic instruction* over DV which can be either *non-parametric* ($v := c$, $v := v'$, $v := expr$) or *parametric* ($v := p$), or a (possibly empty) *sequence* of atomic instructions. Moreover, by $Ins^{\diamond}(DV, DP)$ we denote the set consisting of all these $\alpha \in Ins(DV, DP)$ in which any $v \in DV$ appears on the left-hand side of ":=" (i.e. is assigned a new value, possibly taken from a parameter) at most once. By a *variables valuation* we mean a total mapping $\mathbf{v} : DV \longrightarrow \mathcal{T}$ satisfying $\mathbf{v}(v) \in type(v)$ for each $v \in DV$. We extend this mapping to expressions of $Expr(\mathcal{T})$ in the usual way. Similarly, by a *parameters valuation* we mean a total mapping $\mathbf{p} : DP \rightarrow \mathcal{T}$ satisfying $\mathbf{p}(p) \in type(p)$ for each $p \in DP$. Moreover, we assume that the domain of values for each variable and each parameter is finite.

The satisfaction relation ($\models$) for a boolean expression $\beta \in BoE(DV)$ and a valuation $\mathbf{v}$ is defined as: $\mathbf{v} \models true$, $\mathbf{v} \models \beta_1 \wedge \beta_2$ iff $\mathbf{v} \models \beta_1$ and $\mathbf{v} \models \beta_2$, $\mathbf{v} \models \beta_1 \vee \beta_2$ iff $\mathbf{v} \models \beta_1$ or $\mathbf{v} \models \beta_2$, $\mathbf{v} \models \neg\beta$ iff $\mathbf{v} \not\models \beta$, $\mathbf{v} \models v \sim c$ iff $\mathbf{v}(v) \sim c$, $\mathbf{v} \models v \sim v'$ iff $\mathbf{v}(v) \sim \mathbf{v}(v')$, and $\mathbf{v} \models expr \sim expr'$ iff $\mathbf{v}(expr) \sim \mathbf{v}(expr')$. Given a variables valuation $\mathbf{v}$, a parameter valuation $\mathbf{p}$ and an instruction $\alpha \in Ins(DV, DP)$, we denote by $\mathbf{v}(\alpha, \mathbf{p})$ a valuation $\mathbf{v}'$ such that

- if $\alpha = \epsilon$ then $\mathbf{v}' = \mathbf{v}$,
- if $\alpha = (v := c)$ then for all $v' \in DV$ it holds $\mathbf{v}'(v') = c$ if $v' = v$, and $\mathbf{v}'(v') = \mathbf{v}(v')$ otherwise,
- if $\alpha = (v := v_1)$ then for all $v' \in DV$ it holds $\mathbf{v}'(v') = v_1$ if $v' = v$, and $\mathbf{v}'(v') = \mathbf{v}(v')$ otherwise,

[8] Using different numeric types in the same expression is not allowed. The "/" operator denotes either the integer division or the "ordinary" division, depending on the context.

[9] Distinguishing between assigning an arithmetic expression, and separately assigning a parameter, a constant or a variable follows from the fact that arithmetic expressions are defined for numeric types only. The same applies to the definition of boolean expressions.

– if $\alpha = (v := expr)$ then for all $v' \in DV$ it holds $\mathbf{v}'(v') = expr$ if $v' = v$, and $\mathbf{v}'(v') = \mathbf{v}(v')$ otherwise,
– if $\alpha = (v := p)$ then for all $v' \in DV$ it holds $\mathbf{v}'(v') = \mathbf{p}(p)$, and $\mathbf{v}'(v') = \mathbf{v}(v')$ otherwise,
– if $\alpha = \alpha_1\alpha_2$ then $\mathbf{v}' = (\mathbf{v}(\alpha_1, \mathbf{p}))(\alpha_2, \mathbf{p})$.

Let $\mathcal{X} = \{x_1, \ldots, x_{n_\mathcal{X}}\}$ be a finite set of real-valued variables, called *clocks*. The set of *clock constraints* over $\mathcal{X}$, denoted $\mathcal{C}_\mathcal{X}(\mathcal{X})$, is defined by the grammar:

$$\mathfrak{cc} ::= true \mid x_i \sim c \mid x_i - x_j \sim c \mid \mathfrak{cc} \wedge \mathfrak{cc},$$

where $x_i, x_j \in \mathcal{X}$, $c \in \mathbb{N}$, and $\sim \in \{\leq, <, =, >, \geq\}$. Let $\mathcal{X}^+$ denote the set $\mathcal{X} \cup \{x_0\}$, where $x_0 \notin \mathcal{X}$ is a fictitious clock representing the constant 0. An *assignment* over $\mathcal{X}$ is a function $\mathfrak{a} : \mathcal{X} \longrightarrow \mathcal{X}^+$. $Asg(\mathcal{X})$ denotes the set of all the assignments over $\mathcal{X}$.

By a *clock valuation* we mean a total mapping $\mathbf{c} : \mathcal{X} \longrightarrow \mathbb{R}_+$. The satisfaction relation ($\models$) for a clock constraint $\mathfrak{cc} \in \mathcal{C}_\mathcal{X}(\mathcal{X})$ and a clock valuation $\mathbf{c}$ is defined as $\mathbf{c} \models true$, $\mathbf{c} \models (x_i \sim c)$ iff $\mathbf{c}(x_i) \sim c$, $\mathbf{c} \models (x_i - x_j \sim c)$ iff $\mathbf{c}(x_i) - \mathbf{c}(x_j) \sim c$, and $\mathbf{c} \models \mathfrak{cc}_1 \wedge \mathfrak{cc}_2$ iff $\mathbf{c} \models \mathfrak{cc}_1$ and $\mathbf{c} \models \mathfrak{cc}_2$. In what follows, the set of all the clock valuations satisfying a clock constraint $\mathfrak{cc}$ is denoted by $[\![\mathfrak{cc}]\!]$. Given a clock valuation $\mathbf{c}$ and $\delta \in \mathbb{R}_+$, by $\mathbf{c} + \delta$ we denote a clock valuation $\mathbf{c}'$ such that $\mathbf{c}'(x) = \mathbf{c}(x) + \delta$ for all $x \in \mathcal{X}$. Moreover, for a clock valuation $\mathbf{c}$ and an assignment $\mathfrak{a} \in Asg(\mathcal{X})$, by $\mathbf{c}(\mathfrak{a})$ we denote a clock valuation $\mathbf{c}'$ such that for all $x \in \mathcal{X}$ it holds $\mathbf{c}'(x) = \mathbf{c}(\mathfrak{a}(x))$ if $\mathfrak{a}(x) \in \mathcal{X}$, and $\mathbf{c}'(x) = 0$ otherwise (i.e., if $\mathfrak{a}(x) = x_0$). Finally, by $\mathbf{c}^0$ we denote the *initial* clock valuation, i.e., the valuation such that $\mathbf{c}^0(x) = 0$ for all $x \in \mathcal{X}$.

Definition 11. *A* timed automaton with discrete data and parametric assignments *(TADDPA) is a tuple* $\mathcal{A} = (\mathcal{L}, L, l^0, DV, DP, \mathcal{X}, \mathcal{E}, \mathcal{I}_c, \mathcal{I}_v, \mathbf{v}^0)$*, where* $\mathcal{L}$ *is a finite set of* labels (actions)*,* L *is a finite set of* locations*,* $l^0 \in L$ *is the* initial location*,* DV *is a finite set of variables (of the types in* $\mathcal{T}$*),* DP *is a finite set of parameters (of the types in* $\mathcal{T}$*),* $\mathcal{X}$ *is a finite set of clocks,* $\mathcal{E} \subseteq L \times \mathcal{L} \times BoE(DV) \times \mathcal{C}_\mathcal{X}(\mathcal{X}) \times Ins^\diamond(DV, DP) \times Asg(\mathcal{X}) \times L$ *is a* transition relation*,* $\mathcal{I}_c : L \longrightarrow \mathcal{C}_\mathcal{X}(\mathcal{X})$ *and* $\mathcal{I}_v : L \longrightarrow BoE(DV)$ *are, respectively* a clocks' *and* a variables' invariant functions*, and* $\mathbf{v}^0 : DV \longrightarrow \mathcal{T}$ *s.t.* $\mathbf{v}^0 \models \mathcal{I}_v(l^0)$ *is an initial variables valuation.*

The invariant functions assign to each location a clock constraint and a boolean expression specifying the conditions under which $\mathcal{A}$ can stay in this location. Each element $t = (l, \mathfrak{l}, \beta, \mathfrak{cc}, \alpha, \mathfrak{a}, l') \in \mathcal{E}$ denotes a transition from the location l to the location l', where $\mathfrak{l}$ is the label of the transition t, β and $\mathfrak{cc}$ define the enabling conditions for t, α is the instruction to be performed, and $\mathfrak{a}$ is the clock assignment. Moreover, for a transition $t = (l, \mathfrak{l}, \beta, \mathfrak{cc}, \alpha, \mathfrak{a}, l') \in \mathcal{E}$ we write $source(t)$, $label(t)$, $vguard(t)$, $cguard(t)$, $instr(t)$, $asgn(t)$ and $target(t)$ for l, $\mathfrak{l}$, β, $\mathfrak{cc}$, α, $\mathfrak{a}$ and l' respectively.

Semantics of the above automata is given as follows:

Definition 12. Semantics *of a TADDPA* $\mathcal{A} = (\mathcal{L}, L, l^0, DV, DP, \mathcal{X}, \mathcal{E}, \mathcal{I}_c, \mathcal{I}_v, \mathbf{v}^0)$ *for a parameter valuation* $\mathbf{p} : DP \longrightarrow \mathcal{T}$ *is a labelled transition system*[10] $\mathcal{S}(\mathcal{A}, \mathbf{p}) = (Q, q^0, \mathcal{L}_\mathcal{S}, \rightarrow)$*, where:*

[10] By a labelled transition system we mean a tuple $\mathcal{S} = (S, s^0, \Lambda, \rightarrow)$, where S is a set of states, $s^0 \in S$ is the initial state, Λ is a set of labels, and $\rightarrow \subseteq S \times \Lambda \times S$ is a (labelled) transition relation.

- $Q = \{(l, \mathbf{v}, \mathbf{c}) \mid l \in L \wedge \forall_{v \in DV} \mathbf{v}(v) \in \mathit{type}(v) \wedge \mathbf{c} \in \mathbb{R}_+^{|\mathcal{X}|} \wedge \mathbf{c} \models \mathcal{I}_c(l) \wedge \mathbf{v} \models \mathcal{I}_v(l)\}$ *is the set of states,*
- $q^0 = (l^0, \mathbf{v}^0, \mathbf{c}^0)$ *is the initial state,*
- $\mathcal{L}_\mathcal{S} = \mathcal{L} \cup \mathbb{R}_+$ *is the set of labels,*
- $\rightarrow \subseteq Q \times \mathcal{L}_\mathcal{S} \times Q$ *is the smallest transition relation defined by the rules:*
  - *for* $\mathfrak{l} \in \mathcal{L}$, $(l, \mathbf{v}, \mathbf{c}) \xrightarrow{\mathfrak{l}} (l', \mathbf{v}', \mathbf{c}')$ *iff there exists a transition* $t = (l, \mathfrak{l}, \beta, \mathfrak{cc}, \alpha, \mathfrak{a}, l') \in \mathcal{E}$ *such that* $\mathbf{v} \models \mathcal{I}_v(l)$, $\mathbf{c} \models \mathcal{I}_c(l)$, $\mathbf{v} \models \beta$, $\mathbf{c} \models \mathfrak{cc}$, $\mathbf{v}' = \mathbf{v}(\alpha, \mathbf{p}) \models \mathcal{I}_v(l')$, *and* $\mathbf{c}' = \mathbf{c}(\mathfrak{a}) \models \mathcal{I}_c(l')$ (action transition),
  - *for* $\delta \in \mathbb{R}_+$, $(l, \mathbf{v}, \mathbf{c}) \xrightarrow{\delta} (l, \mathbf{v}, \mathbf{c} + \delta)$ *iff* $\mathbf{c}, \mathbf{c} + \delta \models \mathcal{I}_c(l)$ (time transition).

A transition $t \in \mathcal{E}$ is *enabled* at a state $(l, \mathbf{v}, \mathbf{c})$ for a given parameter valuation $\mathbf{p}$ if $\mathbf{v} \models \mathit{vguard}(t)$, $\mathbf{c} \models \mathit{cguard}(t)$, $\mathbf{c}(\mathit{asgn}(t)) \models \mathcal{I}_c(\mathit{target}(t))$, and $\mathbf{v}(\mathit{instr}(t), \mathbf{p}) \models \mathcal{I}_v(\mathit{target}(t))$. Intuitively, in the initial state all the variables are set to their initial values, and all the clocks are set to zero. Then, being in a state $q = (l, \mathbf{v}, \mathbf{c})$ the system can either execute an enabled transition t and move to the state $q' = (l', \mathbf{v}', \mathbf{c}')$ where $l' = \mathit{target}(t)$, the valuation of variables is changed according to $\mathit{instr}(t)$ and the parameter valuation $\mathbf{p}$, and the clock valuation is changed according to $\mathit{asgn}(t)$, or move to the state $q' = (l, \mathbf{v}, \mathbf{c} + \delta)$ which results from passing some time $\delta \in \mathbb{R}_+$ such that $\mathbf{c} + \delta \models \mathit{inv}(l)$.

We say that a location l (a variables valuation $\mathbf{v}$, respectively) is *reachable* if some state $(l, \cdot, \cdot)$ ($(\cdot, \mathbf{v}, \cdot)$, respectively) is reachable in $\mathcal{S}(\mathcal{A}, \mathbf{p})$. Given $D \subseteq DV$, a partial variables valuation $\mathbf{v}_D : D \longrightarrow \mathcal{T}$ is reachable if some state $(\cdot, \mathbf{v}, \cdot)$ s.t. $\mathbf{v}|_D = \mathbf{v}_D$ is reachable in $\mathcal{S}(\mathcal{A}, \mathbf{p})$.

5.2 SAT-Based Reachability Checking

In the paper [19] we showed how to test reachability for timed automata with discrete data (TADD) using SAT-based bounded model checking to this aim. The main idea consisted in discretising the set of clock valuation of the automaton considered, in order to obtain a countable state space. Next, the transition relation of the transition system obtained was unfolded up to some depth k, and the unfolding was encoded as a propositional formula. The property to be tested was encoded as a propositional formula as well, and satisfiablity of the conjunction of these two formulas was checked using a SAT-solver. Satisfiability of the conjunction allowed to conclude that a path from the initial state to a state satisfying the property was found.

Comparing with the automata considered in [19], the automata used in this paper are extended in the following way:

- values of discrete variables are not only integers, but are of several discrete types,
- arithmetic expressions used can be of a more involved form,
- the invariant function involves not only clock comparisons, but also boolean expression over values of discrete variables,
- the definition of the automaton contains additionally a set of parameters, and the instructions can be assignments of the form $a\_variable := a\_parameter$.

As it is easy to see, discretisation of the set of clock valuations for TADDPA can be done analogously as in [19]. The way of extending arithmetic operations on integers was described in [18]. New data types can be handled by conversions to integers; introducing extended invariants is straightforward. The only problem whose solution cannot be easily adapted from the previous approach is that SAT-based reachability testing for TADDPA involves also searching for a parameter valuation for which a state satisfying a given property can be reached. However, the idea of doing this can be derived from the idea of SAT-solvers: a SAT-solver searches for a valuation of propositional variables for which a formula holds. Thus, we represent the values of parameters by sets of propositional variables; finding a valuation for which a formula γ which encodes that a state satisfying a given property is reachable along a path of a length k implies also finding an appropriate valuation of parameters occurring along the path considered.

5.3 SAT-Based Service Composition

In order to apply the above verification method to automatic searching for sequences of concrete services able to satisfy the user's request we translate paths of the abstract graph to timed automata with discrete data and parametric assignments. The translation uses the descriptions of concrete services, as well as the user's query.

Consider a path $\pi = w_0 \to w_1 \to \ldots w_n$ ($n \in \mathbb{N}$) in the abstract graph, such that $w_0 \in V_p$ (i.e., is an initial world) and $w_n \in V_k$ (i.e., is a final world) - i.e., a sequence of worlds and abstract services which transform them. Let O_π be the set of all the objects which occur in all the worlds along this path (i.e., $O_\pi = \{o \in w_i \mid i = 0, \ldots, n\}$). Then, we define $V(\pi) = \{objectName.attributeName \mid objectName \in O_\pi\}$, $V_{pre}(\pi) = \{objectName.attributeName.pre \mid objectName \in O_\pi \wedge \exists_{i \in \{0,\ldots,n-1\}}$ $objectName \in w_i \cap w_{i+1}\}$ and $V'(\pi) = V(\pi) \cup \{v.isConst \mid v \in V(\pi)\} \cup \{v.isSet \mid v \in V(\pi)\} \cup \{v.isAny \mid v \in V(\pi)\} \cup V_{pre}(\pi)$. The set of discrete variables of the automaton $\mathcal{A}(\pi)$ corresponding to π is equal to $V'(\pi)$. The intuition behind this construction is that for each attribute of each objects occuring along the path we define a variable aimed at storing the value of the attribute ($objectName.attributeName$). Moreover, for each such variable we introduce three new boolean variables: the one saying whether the flag `isConst` for the attribute has been set ($objectName.attributeName.isConst$), the second one to express that the attribute has been set (has a nonempty value; $objectName.attributeName.isSet$, and the third one to specify that the value of the attribute is nonempty but its exact value is not given ($objectName.attributeName.isAny$). The variables in $V_{pre}(\pi)$ (of the form $objectName.attributeName.pre$) are aimed at storing values of attributes from a pre-world of a service.

The initial values of variables are taken from the initial world w_0 resulting from the user's query:

- for each attribute `x.y` which according to the query has a concrete value γ in w_0, we set $x.y := \gamma$, $x.y.isAny := false$ and $x.y.isSet := true$; concerning $x.y.isConst$ we set it $true$ if such a condition occurs in the query, otherwise it is set to $false$,
- for each attribute `x.y` which according to the query is set, but its value is not given directly, we set $x.y.isSet := true$, and $x.y.isAny = true$; $x.y.isConst$ is set

according to the query as above; $x.y$ can obtain any value of the appropriate type (we can assume it gets a "zero" value of $type(x.y)$),

- for each attribute `x.y` which does not occur in the query or is specified there as having the empty value we set $x.y.isSet = false$, $x.y.isAny = true$, $x.y.isConst = false$, the value of $x.y$ is set to an arbitrary value as above,
- each variable of the form $x.y.pre$ is assumed to have a "zero" value of $type(x, y)$.

Define for each w_i, $i = 0, \ldots, n$, a new location of $\mathcal{A}(\pi)$, denoted for simplicity w_i as well, and consider an edge $w_i \to w_{i+1}$ of π ($i \in \{0, \ldots, n-1\}$), corresponding to an abstract service sa_i. For each concrete service s of the type of sa_i we introduce a new location w_i^s and the transitions $w_i \xrightarrow{s} w_i^s$ and $w_i^s \xrightarrow{\varepsilon} w_{i+1}$ (where ε is an "empty" label)[11]. Then, we make use of the description of s as follows:

- the precondition of s becomes the guard of the transition $w_i \xrightarrow{s} w_i^s$ (notice that a disjunctive form is here allowed);
- the list `requires` of s is used to construct the instruction α "decorating" $w_i \xrightarrow{s} w_i^s$: initially α is set to ϵ, then, for each attribute `y` of an object `x` occurring in `requires` for which it holds $x.y.isSet = true$, α is extended by concatenating $x.y.pre := x.y$,
- the lists `mustSet` and `mustSetConst` of s are used to construct the instruction α as well: for each attribute `x.y` occuring in the list `mustSet` α is extended by concatenating $x.y.isSet := true$, and for each attribute `x.y` occuring in the list `mustSetConst` of s α is extended by concatenating $x.y.isConst := true$,
- the postcondition is used as follows (`x.y` denotes an attribute):
  - the predicates `Exists` are ignored,
  - the (possibly negated) predicates of the form `isSet(x.y)` or `isConst(x.y)` result in extending the instruction α "decorating" $w_i \xrightarrow{s} w_i^s$ by concatenating respectively $x.y.isSet := true$ or $x.y.isConst := true$ if such an instruction has not been added to α before (or respectively $x.y.isSet := false$ or $x.y.isConst := false$ if the predicates are negated)[12],
  - each predicate of the form `x.y = z` or `post(x.y)=z` (where `z` can be either a concrete value or an expression[13]) results in extending the instruction α by concatenating $x.y := z$, $x.y.isSet := true$ (if it has not been added before) and $x.y.isAny := false$,
  - for each predicate of the form `x.y # z` or `post(x.y) # z` with $\# \in \{<, >, \leq, \geq\}$ (where `z` is either a concrete value or an expression) we introduce a new parameter p, extend α by concatenating $x.y := p$, $x.y.isSet := true$ (if it has not been added before) and $x.y.isAny := false$, and conjunct the invariant of w_i^s (initially $true$) with the above predicate.

[11] We assume here that the postcondition of s contains no disjunctions; otherwise we treat s a number of concrete services each of which has the postcondition corresponding to one part of the DNF in the original postcondition of s.

[12] Possible inconsistencies, i.e. an occurence of `x.y` in `mustSet` and the predicate `not isSet(x.y)` in `postCondition`, are treated as ontology errors.

[13] Recall that the expressions can refer only to values the variables have in the pre-world of a service.

– moreover, for each attribute x.y which occurs either in mustSet or in the post-conidition in a predicate isSet(x.y), but does not have in the postCondtion any "corresponding" predicate which allows to set its value, we introduce a new parameter $p^s_{x.y}$, and extend α by adding $x.y := p^s_{x.y}$ and $x.y.isAny := false$.

The invariants of w_i and w_{i+1}, as well as the guard of the transition labelled with ε are set to $true$. The set of instructions of the latter transition is empty. The set of clocks of $\mathcal{A}(\pi)$ is empty as well. The intuition behind the above construction is as follows: initially, only the variables of the form $x.y$ corresponding to attributes specified by the user's query as having concrete values are set, while the rest stores random values (which is expressed by $x.y.isAny = true$). Next, concrete services modify values of the variables. If the description of a service specifies that an attribute is set and specifies the exact value assigned, then the transition corresponding to execution of this service sets the corresponding variable in an appropriate way. If the exact value of the attribute set is not given, a parameter for the value assigned is introduced, and possible conditions on this parameter (specified in the postcondition) are assigned to the target location as a part of its invariant. Moreover, before introducing any changes to the values of the variables corresponding to the attributes of the objects in requires their previous values are stored.

The above construction can be optimised in several ways. Firstly, one can add a new "intermediate" location w^s_i only in the case when no location, corresponding to a service of the same type as s and having the appropriate invariant, has been added before; otherwise, the transition outgoing w_i can be redirected to the existing location. Secondly, the variables of the form $x.y.pre$ can be introduced only for these attributes for which there is a postcondition of a service which refers both to pre(x).y and post(x).y. Finally, if we have several concrete services of a given type t occuring as the i-th service along the abstract path, and - according to the above construction - need to introduce for each of them a parameter to be assigned to a variable $x.y$, then we can reduce the number of parameters: instead of introducing a new parameter for each concrete service we can introduce one parameter $p^{t,i}_{x.y}$. This follows from the fact that only one concrete service of this type is exectuted as the i-th, and therefore only one assignment a new value to x.y is performed.

6 Experimental Results and Concluding Remarks

The method described above has been implemented. The preliminary implementation was tested on a Getting Juice example considered in [3], by running it to generate a sequence of concrete services corresponding to the abstract path SelectWare, then FruitSelling and then MakingJuice. The sequence has been found; a detailed description of the example together with the result can be found in the appendix.

Currently, the automaton is generated by hand, since a repository of concrete services is still under construction. In the future we are going to automate the method completely, including dynamic translation of a service, dynamic creation of enumeration types based on the query and on the contents of the repository, and building the automaton step by step. This will enable us to test efficiency of the approach.

References

1. S. Ambroszkiewicz. *enTish: An Approach to service Description and Composition*. ICS PAS, Ordona 21, 01-237 Warsaw, 2003.
2. DAML-S (and OWL-S) 0.9 draft release. http://www.daml.org/services/daml-s/0.9/, 2003.
3. M. Jarocki, A. Niewiadomski, W. Penczek, A. Półrola, and M. Szreter. Towards automatic composition of web services: Abstract planning phase. Technical Report 1017, ICS PAS, Ordona 21, 01-237 Warsaw, February 2010.
4. M. Klusch, A. Geber, and M. Schmidt. Semantic web service composition planning with OWLS-XPlan. In *Proc. of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web*. AAAI Press, 2005.
5. D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the Planning Domain Definition Language - version 1.2. Technical Report TR-98-003, Yale Center for Computational Vision and Control, 1998.
6. S. R. Ponnekanti and A. Fox. SWORD: A developer toolkit for web service composition. In *Proc. of the 11st Int. World Wide Web Conference (WWW'02)*, 2002.
7. J. Rao. *Semantic Web Service Composition via Logic-Based Program Synthesis*. PhD thesis, Dept. of Comp. and Inf. Sci., Norwegian University of Science and Technology, 2004.
8. J. Rao, P. Küngas, and M. Matskin. Logic-based web services composition: From service description to process model. In *Proc. of the IEEE Int. Conf. on Web Services (ICWS'04)*. IEEE Computer Society, 2004.
9. J. Rao and X. Su. A survey of automated web service composition methods. In *Proc. of the 1st Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, pages 43–54, 2004.
10. D. Redavid, L. Iannone, and T. Payne. OWL-S atomic services composition with SWRL rules. In *Proc. of the 4th Italian Semantic Web Workshop: Semantic Web Applications and Perspectives (SWAP 2007)*, 2007.
11. RSat. http://reasoning.cs.ucla.edu/rsat, 2006.
12. E. Sirin, J. Hendler, and B. Parsia. Semi-automatic compositions of web services using semantic description. In *Proc. of the Int. Workshop 'Web Services: Modeling, Architecture and Infrastructure' (at ICEIS 2003)*, 2003.
13. SOAP version 1.2. http://www.w3.org/TR/soap, 2007.
14. B. Srivastava and J. Koehler. Web service composition - current solutions and open problems. In *Proc. of Int. Workshop on Planning for Web Services (at ICAPS 2003)*, 2003.
15. Universal Description, Discovery and Integration v3.0.2 (UDDI). http://www.oasis-open.org/committees/uddi-spec /doc/spec/v3/uddi-v3.0.2-20041019.htm, 2005.
16. Web Services Business Process Execution Language v2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 2007.
17. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001.
18. A. Zbrzezny. A boolean encoding of arithmetic operations. Technical Report 999, ICS PAS, 2007.
19. A. Zbrzezny and A. Półrola. SAT-based reachability checking for timed automata with discrete data. *Fundamenta Informaticae*, 70(1-2):579–593, 2007.

A Experimental Results - A Detailed Description

Below we present the Getting Juice example considered in [3]. Assume we have the following classes:

```
Ware       id       integer
Ware       name     string
Ware       owner    string
Measurable capacity float
Juice      extends  Ware, Measurable
Fruits     extends  Ware, Measurable
```

and the following types of services:

```
SelectWare  produces    w:Ware
SelectWare  consumes    null
SelectWare  requires    null
SelectWare  mustSet     w.name; w.owner

Selling  produces       null
Selling  consumes       null
Selling  requires       w:Ware
Selling  mustSet        w.id; w.owner
Selling  preCondition   not isSet(w.id) and isSet(w.name)
                        and isSet(w.owner)
Selling  postCondition  w.owner!=pre(w).owner

FruitSelling  extends        Selling
FruitSelling  requires       w:Fruits
FruitSelling  mustSet        w.capacity
FruitSelling  postCondition  w.capacity>0

JuiceSelling  extends        Selling
JuiceSelling  requires       w:Juice
JuiceSelling  mustSet        w.capacity
JuiceSelling  postCondition  w.capacity>0

MakingJuice  produces       j:Juice
MakingJuice  consumes       f:Fruits
MakingJuice  mustSet        j.id; j.name; j.capacity
MakingJuice  preCondition   isSet(f.id) and isSet(f.name) and
                            isSet(f.owner) and f.capacity>0
MakingJuice  postCondition  isSet(j.id) and isSet(j.name) and
                            j.capacity>0
```

The user's query is specified as follows:

```
InitWorld    null
InitClause   true
EffectWorld  j:Juice
EffectClause j.id>0 and j.capacity=10 and j.owner="Me"
```

One of the sequences of services which possibly can lead to satisfying the query is `SelectWare`, then `FruitSelling` and then `MakingJuice` [3]. Below we consider concretising the above path of the abstract graph.

Assume the concrete instances of the `SelectWare` specify the following offers[14]:

[14] The current version of our implementation does not deal with inheritance of classes.

```
FruitNetMarket mustSet       w.name, w.owner
FruitNetMarket preCondition  -
FruitNetMarket postCondition (w.name=strawberry and
                              w.owner=shop1)  or
                             (w.name=blueberry and w.owner=shop1)

FruitNetOffers mustSet       w.name, w.owner
FruitNetOffers preCondition   -
FruitNetOffers postCondition (w.name=plum and w.owner=shop2) or
                             (w.name=apple and w.owner=shop2) or
                             (w.name=apple and w.owner=shop3)
```

Next, the fruitselling services specify:

```
Shop1 mustSet         w.id, w.owner, w.capacity
Shop1 precondition    not isSet(w.id) and isSet(w.name)
                      and isSet(w.owner) and w.owner=shop1
Shop1 postcondition   w.owner!=pre(w).owner and w.id>0 and
                      w.capacity>0 and w.capacity<=10

Shop2 mustSet         w.id, w.owner, w.capacity
Shop2 precondition    not isSet(w.id) and isSet(w.name)
                      and isSet(w.owner) and w.owner=shop2
Shop2 postcondition   w.owner!=pre(w).owner and w.id>0 and
                      w.capacity>0

Shop3 mustSet         w.id, w.owner, w.capacity
Shop3 precondition    not isSet(w.id) and isSet(w.name)
                      and isSet(w.owner) and w.owner=shop3
Shop3 postcondition   w.owner!=pre(w).owner and w.id>0
                      and w.capacity>=100
```

which means that Shop1 is able to sell at most 10 units of fruits, Shop2 - at least 100 units, while Shop3 is able to sell any amount. Finally, we have the following services which make juice:

```
HomeJuiceMaking mustSet       j.id, j.name, j.capacity
HomeJuiceMaking preCondition  isSet(f.id) and isSet(f.name) and
                              isSet(f.capacity) and isSet(f.owner)
                              and f.capacity>0
                              and f.capacity<=10 and
                              f.name!=plum and f.name!=apple
HomeJuiceMaking postCondition isSet(j.id) and isSet(j.name) and
                              j.capacity>0 and j.name=f.name
                              and j.capacity=f.capacity
                              and j.owner=f.owner

GrandmaKitchen mustSet      j.id, j.name, j.capacity
GrandmaKitchen preCondition isSet(f.id) and isSet(f.name) and
                            isSet(f.capacity) and isSet(f.owner)
                            and f.capacity>0 and f.capacity<=5
```

```
GrandmaKitchen postCondition isSet(j.id) and isSet(j.name) and
                             j.capacity>0 and j.name=f.name
                             and j.capacity=f.capacity
                             and j.owner=f.owner

JuiceTex      mustSet       j.id, j.name, j.capacity
JuiceTex      preCondition  isSet(f.id) and isSet(f.name) and
                            isSet(f.capacity) and isSet(f.owner)
                            and f.capacity>0
JuiceTex      postCondition isSet(j.id) and isSet(j.name) and
                            j.capacity>0 and j.name=f.name
                            and j.capacity=2*f.capacity
                            and j.owner=f.owner
```

Thus, assume that we have the following types: integer, float (we can assume that the precision is up to two decimal places), `FruitTypes = (strawberry, blueberry, apple, plum)`, and `OwnerNames = (Me, Shop1, Shop2, Shop3, Shop4)` (the ranges of enumeration types can be deduced from the offers and from the user's query). The variables and their types are: $f.id : integer, f.name : FruitTypes, f.owner : OwnerNames, f.capacity : float, j.id : integer, j.name : FruitTypes, j.owner : OwnerNames, j.capacity : float$ plus the corresponding boolean variables of the form $x.y.isSet$, $x.y.isAny$ and $x.y.isConst$. Moreover, we introduce one additional variable $f.owner.pre : OwnerNames$ to store the previous value of $f.owner$[15].

All the variables of the form x, y are initialised to zero values of the appropriate types, each $x.y.isSet$ and $x.y.isConst$ is initialised with $false$, and each $x.y.isAny$ is initialised with $true$.

– The `FruitNetMarket` generates two transitions)together with the intermediate locations and the "ε-transitions" outgoing them). The first one is decorated with $f.name.isSet := true; f.owner.isSet := true; f.name := strawberry; f.owner := shop1; f.owner.isAny := false; f.name.isAny := false$, the second one is decorated in a similar way, but with $f.name := blueberry$, the edges for the three offers of `FruitNetOffers` look similarily,
– the fruitselling services correspond to the following edges and intermediate locations:
  - for Shop1:
    * the guard of the first edge is $f.id.isSet = false \wedge f.name.isSet = true \wedge f.owner.isSet = true \wedge f.owner = shop1$,
    * the instruction is $f.id.isSet := true; f.owner.isSet := true; f.capacity.isSet := true; f.id.isAny := false; f.owner.isAny := false; f.capacity.isAny := false; f.owner.pre := f.owner;\ f.owner := p^{FS}_{f.own}; f.id := p^{FS}_{f.id}; f.capacity := p^{FS}_{f.cap}$ (where $p^{FS}_{\cdot}$ are parameters),
    * the invariant of the intermediate location is $\neg(f.owner.pre = f.owner) \wedge f.id > 0 \wedge f.capacity > 0 \wedge f.capacity \leq 10$;

[15] We apply the optimisation allowing to add one variable of the form $x.y.pre$ only, as well as the one consisting in reducing the number of parameters.

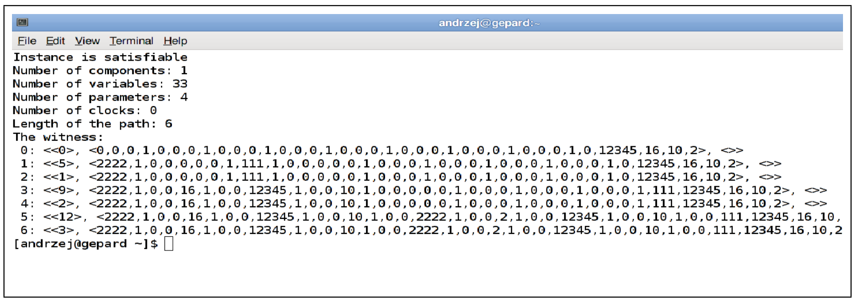
- for Shop2:
  - the guard of the first edge is $f.id.isSet = false \wedge f.name.isSet = true \wedge f.owner.isSet = true \wedge f.owner = shop2$,
  - the instruction is $f.id.isSet := true; f.owner.isSet := true; f.capacity.isSet := true; f.id.isAny := false; f.owner.isAny := false; f.capacity.isAny := false; f.owner.pre := f.owner;\ f.owner := p^{FS}_{f.own}; f.id := p^{FS}_{f.id}; f.capacity := p^{FS}_{f.cap}$,
  - the invariant of the intermediate location is $\neg(f.owner.pre = f.owner) \wedge f.id > 0 \wedge f.capacity > 0$,
- for Shop3:
  - the guard of the first edge is $f.id.isSet = false \wedge f.name.isSet = true \wedge f.owner.isSet = true \wedge f.owner = shop3$,
  - the instruction is $f.id.isSet := true; f.owner.isSet := true; f.capacity.isSet := true; f.id.isAny := false; f.owner.isAny := false; f.capacity.isAny := false; f.owner.pre := f.owner;\ f.owner := p^{FS}_{f.own}; f.id := p^{FS}_{f.id}; f.capacity := p^{FS}_{f.cap}$,
  - the invariant of the intermediate location is $\neg(f.owner.pre = f.owner) \wedge f.id > 0 \wedge f.capacity \geq 0$,

– for the services making juice from fruits:
- for `HomeJuiceMaking`:
  - the guard of the first edge is $f.id.isSet = true \wedge f.name.isSet = true \wedge f.owner.isSet = true \wedge f.capacity.isSet = true \wedge f.capacity > 0 \wedge f.capacity \leq 10 \wedge \neg(f.name = plum) \wedge \neg(f.name = apple)$
  - the instruction is $j.id.isSet := true; j.id.isAny := false; j.name.isSet := true; j.name.isAny := false; j.capacity.isSet := true; j.capacity.isAny := false; j.owner.isSet := true; j.owner.isAny := false; j.id := p^{MJ}_{j.id}; j.capacity := f.capacity; j.name := f.name, j.owner := f.owner$
  - the invariant of the intermediate location is $j.capacity > 0$
- for `GrandmaKitchen`
  - the guard of the first edge is $f.id.isSet = true \wedge f.name.isSet = true \wedge f.owner.isSet = true \wedge f.capacity.isSet = true \wedge f.capacity > 0 \wedge f.capacity \leq 5$,
  - the instruction is $j.id.isSet := true; j.id.isAny := false; j.name.isSet := true; j.name.isAny := false; j.capacity.isSet := true; j.capacity.isAny := false; j.owner.isSet := true; j.owner.isAny := false; j.id := p^{MJ}_{j.id}; j.capacity := f.capacity; j.name := f.name, j.owner := f.owner$
  - the invariant of the intermediate location is $j.capacity > 0$
- for `JuiceTex`
  - the guard of the first edge is $f.id.isSet = true \wedge f.name.isSet = true \wedge f.owner.isSet = true \wedge f.capacity.isSet = true \wedge f.capacity > 0$
  - the instruction is $j.id.isSet := true; j.id.isAny := false; j.name.isSet := true; j.name.isAny := false; j.capacity.isSet := true; j.capacity.isAny := false; j.owner.isSet := true; j.owner.isAny := false; j.id := p^{MJ}_{j.id}; j.capacity := 2 * f.capacity; j.name := f.name, j.owner := f.owner$

* $*$ the invariant of the intermediate location is $j.capacity > 0$

The condition to be tested is $j.id.isSet \wedge \neg j.id.isAny \wedge j.capacity.isSet \wedge \neg j.capacity.isAny \wedge j.owner.isSet \wedge \neg j.owner.isAny \wedge j.id > 0 \wedge j.capacity = 10 \wedge j.owner = me$. In practice, we extend it by adding an additional proposition which is true in the locations $w_0, \ldots, w_3$ to avoid obtaining paths which finish before both the transitions corresponding to a concrete service are executed.

After running our preliminary implementation, we have obtained the path corresponding to selling 10 units of blueberries by `Shop1` and processing them by `HomeJuiceMaking`. A screenshot displaying the witness is presented in Fig. 1. To find the



```
Instance is satisfiable
Number of components: 1
Number of variables: 33
Number of parameters: 4
Number of clocks: 0
Length of the path: 6
The witness:
 0: <<0>, <0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,12345,16,10,2>, <>>
 1: <<5>, <2222,1,0,0,0,0,0,1,111,1,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,12345,16,10,2>, <>>
 2: <<1>, <2222,1,0,0,0,0,0,1,111,1,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,12345,16,10,2>, <>>
 3: <<9>, <2222,1,0,0,16,1,0,0,12345,1,0,0,10,1,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,111,12345,16,10,2>, <>>
 4: <<2>, <2222,1,0,0,16,1,0,0,12345,1,0,0,10,1,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,111,12345,16,10,2>, <>>
 5: <<12>, <2222,1,0,0,16,1,0,0,12345,1,0,0,10,1,0,0,2222,1,0,0,2,1,0,0,12345,1,0,0,10,1,0,0,111,12345,16,10,
 6: <<3>, <2222,1,0,0,16,1,0,0,12345,1,0,0,10,1,0,0,2222,1,0,0,2,1,0,0,12345,1,0,0,10,1,0,0,111,12345,16,10,2
[andrzej@gepard ~]$
```

Fig. 1. A witness for concretisation of an abstract path for the Getting Juice example

witness, we checked satisfaction of the boolean formula encoding the translation of the tested condition. The formula in question consisted of 20152 variables and 52885 clauses; our implementation needed 0.65 second and 6.2 MB memory to produce it. Its satisfiability was checked by RSAT[11], a mainstream SAT solver; to checking it 0.1 seconds and 5.6 MB of memory were needed.

Improving the significance of benchmarks for Petri nets model checkers

Steve Hostettler, Alban Linard, Alexis Marechal, and Matteo Risoldi

Université de Genève, Route de Drize 7, 1227 Carouge - Switzerland
{FirstName.LastName}@unige.ch

Abstract. Benchmarking is a fundamental activity to rigorously quantify the improvements of a new approach or tool with respect to the state of the art. Generally, it consists in comparing results of a given technique with more or less similar approaches. In the Petri nets community, the comparison is often centered on model checking and/or state space calculation performance. However, there is sometimes little justification for the choice of the techniques to compare to. Also, benchmarks often lack context information, such as the exact model used, or how to reproduce the results. This makes it difficult to draw conclusions from the comparisons.
We conducted a survey among the Petri nets community in which we gathered information about the used formalisms and techniques. This revealed an unanimous interest for a common repository of benchmarks. The survey shows that existing efforts in this direction suffer from limitations that prevent their effectiveness.
In this article we report the results of the survey and we outline perspectives for improving Petri nets benchmark repositories.

1 Introduction

One of the goals of developing modern model checkers is often improving the performance of existing tools, whether in terms of time, memory consumption or scalability. A rigorous scientific method requires these improvements to be quantified in order to communicate them to the community. Usually this is done by running the model checker against a set of known benchmarks. A benchmark is

> [...] *a standard or point of reference against which things may be compared or assessed.*[1]

Benchmarks can range from academic examples which focus on a very specific aspect (*e.g.,* the dining philosophers for testing scalability) to more general, real-life examples (*e.g.,* an avionics system with real-time, concurrency and dependability requirements). Benchmark results should be useful when comparing to the state of the art. However, a meaningful comparison of benchmarks is often difficult for a number of reasons. One of them is that the typical article has no space to give enough details about features of the used model that could have greatly influenced the benchmark performance. For example, using a low-level modeling formalism with ad-hoc optimizations for the tool under test can bias the results. Also, even when one tries to use the same model for benchmarking as has been used for another tool, the translation from one formalism to another can be less than trivial, if not a daunting task. Formalisms differ in semantics, in the way of expressing information, and in expressive capabilities. Some are more abstract and can tackle several semantic aspects (*e.g.,* time, concurrency, algebras...) while others are more restricted. Translating, for example, a Coloured Petri Net (CPN) to a P/T net (PTN) requires some simplifications that have an impact on the interpretation of performance results.

We ran a survey in the Petri nets community, asking what formalisms are being used, what type (if any) of model checking is performed and what are the opinions on the hypothesis of standardizing benchmarks. This article is based on the answers of the community. In it we propose a way to improve the current situation by helping standardizing and classifying models used for benchmarks. We review previous efforts in creating repositories, analyze their limitations and how they can evolve to offer better functionality. The goal of the article is fostering a discussion towards a better, more usable repository of models that the community can benefit from.

The structure of the article is as follows. Sec. 2 reports the answers to the survey and the requirements that have been identified as a consequence. Sec. 3 analyzes existing related work. Sec. 4 illustrates our proposal. Sec. 5 discusses some critical points and open questions of the proposal. Sec. 6 draws conclusions, followed by references and by an appendix with complete diagrams and survey graphs.

2 Requirements

To identify desirable requirements for a repository which improves the current situation, we performed a survey among the Petri net community members. We asked 40 groups and individual researchers to participate. We received feedback from 18 (45%) of them. Here we are only reporting pertinent questions and answers; the complete list is freely available on request to smv@unige.ch. Graphs with answer percentages for questions reported here are in appendix A.

A fundamental couple of questions we asked was:

– Would you find useful to have a central repository for models?
– Would you find useful to have test beds[1] for each axis of research in verification?

The answer has been a resounding 100% yes for both questions. This confirms the suggested lack of a unified resource for benchmarking models that can adapt to different types of research. There have been some however who expressed doubts about the fact that previous approaches have been tried and did not completely succeed. Also, a worrying factor among respondents was the burden of maintenance of a model repository.

Following this, the next useful step in understanding the requirements of the community is identifying what the mentioned axes of research are. This means differentiating and quantifying the following aspects:

– What modeling formalisms are used (Fig. 9). The vast majority of groups focus on PTNs or CPNs. Other types of Petri nets are however well represented. In most cases, the same researchers use multiple formalisms.

[1] Test beds can be a range of models with specific characteristics on which different research groups can perform benchmarks of their approach.

- What is the main purpose of the respondents' research (Fig. 10). 95% is interested in property validation and verification (the only respondent who isn't is rather interested in modeling and simulation). Those who use higher-level Petri nets formalisms all put an accent on Modeling as a central purpose.
- Among those who are interested in property verification, we asked to specify the kind of properties (Fig. 11). They all verify invariants, a majority (70%) verifies CTL, and about 50% verify LTL.

These points underline a need for a repository to manage models using different formalisms, and centered on different types of semantics. Also, models should be coupled with different types of property definitions.

Of the total number of respondents, 95% perform state space exploration, and almost half employ structural verification methods (Fig. 12). Among those exploring the state space, the majority use explicit state space representation, associated in most cases to some kind of symbolic encoding (Fig. 14). The quasi-totality of respondents is developing and maintaining a tool (Fig. 13). Also, a relevant 30% is using parallel algorithms to boost performance. According to these data, it is relevant to have a way to compare these tools on the basis of their efficiency when managing large state spaces (in terms of memory, time or other metrics).

Respondents to the survey also had the opportunity of adding free-text remarks to their answers. One interesting fact that emerged from this is that most of them have some sort of collection that they routinely use for benchmarks. These models are either from the literature or, in some cases, by some model collection which may or may not be publicly available. Apart from the most typical toy use cases (*e.g.,* dining philosophers, communication protocols...) no overlap was highlighted in the set of models used. This confirms the difficulty in comparing the benchmarks.

On the basis of these survey results, we think that a suitable repository should fulfill the following list of requirements:

i. It should allow for formalism-independent descriptions of models.
ii. It should be possible to include one or more formalism-specific implementations (or *instances*) of the models, possibly in different formalisms.
iii. It should be possible to express properties to check, using different types of property formalisms that focus on different semantic aspects
iv. Models, instances and properties should be classified and searchable by *characteristics*. These can be seen as a sort of "tags" with which models can be selected that are suitable for a certain type of verification activity. Characteristics should be suggested by users when creating models.
v. The repository should store "benchmarks". A benchmark should be seen here as a collection of runs, done on a certain platform, and performed in order to verify some property on a given model. Each run should be characterized by a tool and tool version, as well as by used source files and possible parameters.
vi. It should be possible for registered users to add or update new models, instances, properties or benchmarks in a collaborative way, classifying them accordingly.
vii. Registered users should be able to comment and rate existing models, instances and benchmarks.

viii. The administration workload should be kept to a minimum. Two fundamental tasks should be managing users (to remove unused profiles or fix access problems) and characteristics (to solve eventual duplications). Deleting models/instances/benchmarks should also be an admin's on-request task, while registered users should rather update them.

We defined use cases for these requirements. Their visual representation is found in Fig. 7 (we will not give here the full textual representation of each use case). They will be discussed in more detail in Sec. 4.

3 State of the Art

As we already said, there is a clear need among the Petri nets community for a central repository for models and test beds. There have already been proposals in this direction: we can cite for example Avrunin et al. from the University of Massachusetts (UMASS) [2] who studied the question and even started an implementation, and [3] who proposed a set of interesting benchmarks, without adding source codes for them. We will present here some of the existing repositories and how they fare with the requirements from the previous section.

BEEM Benchmarks for Explicit Model Checkers (BEEM) [4] is a repository of benchmarks. It contains a predefined set of parametric models expressed in a dedicated low level language called DVE. This language can be automatically translated to other formalisms, the repository provides automatic translation to the Promela language. Models are organized by categories of problems (e.g. mutual exclusion, leader election, etc.). Properties are expressed in LTL. Benchmarks are performed using a model checking tool called DiVinE. BEEM contains 57 benchmarks.

PetriWeb PetriWeb [5] is a collaborative repository of Petri nets, i.e. users can submit and store their models. It supports a particular variant of Petri nets with special features like XOR transitions. Models are expressed with a PNML based definition of these Petri Nets, called EPNML. They are organized by properties that are also managed by the community. PetriWeb contains 79 models, by 5 different authors.

pnmlWEB pnmlWEB [6] is another collaborative repository of PetriNets. Users can submit models expressed in PNML, without restriction, along with a description in natural language. This repository doesn't have actual models yet.

VLTS The Very Large Transition Systems benchmark suite (VLTS) [7] contains a set of benchmarks for concurrent systems. These benchmarks are sometimes taken from industrial examples. They are expressed as Labelled Transition Systems, which makes it difficult to reuse the same benchmarks on any given tool. VLTS contains 40 different benchmarks.

UMASS repository Avrunin et al. proposed a repository [8] where models would be expressed in natural language, with implementations written in Ada. Models are submitted along with examples of properties to check, in different languages (CTL, QREs, etc.). No benchmarks are stored in the repository. This project has not been updated since 2003, and it seems that the development is suspended.

The summary of the position of these examples with regards to the requirements expressed in Sec. 2 can be seen in Table 1. We can see that:

– there is no repository that allows the storage of benchmarks for models without relying on a precise formalism.
– there is no collaborative repository that allows the storage of benchmarks.
– there are some collaborative repositories, but none of them allows evaluation by other users using comments or ratings.

We should also mention that the collaborative experiments were not completely successful: PetriWEB has a set of models but they were submitted by a reduced number of authors, and pnmlWEB is not actually used. The UMASS repository also aimed to be collaborative, but this does not seem to be implemented and the project is now abandoned.

Repository		BEEM	PetriWEB	pnmlWEB	VLTS	UMASS
Formalism independent models	i			✓		✓
Implementations	ii	✓	✓		✓	✓
Properties	iii	✓	✓			✓
Classification, search	iv	✓	✓		✓	
Benchmarks	v	✓			✓	
Collaborative	vi		✓	✓		
Comments, rates	vii					
Low admin workload	viii		✓	✓		

Table 1. Repositories classification.

4 Proposal

In this section we propose a high level design that partially fulfills the requirements that have been gathered in Sec. 2. We will go through the requirements and see how they are satisfied. The proposal's design is described using standard Unified Modeling Language (UML) artifacts that are presented in appendix A. Fig. 7 shows the use case diagram that describes the main functionalities. Fig. 8 presents a class diagram of the repository's Domain Object Model (DOM). For the sake of explanation we will extract and present selected chunks of the main class diagram. We reference classes of the DOM using the following notation: [*DOMClass*].

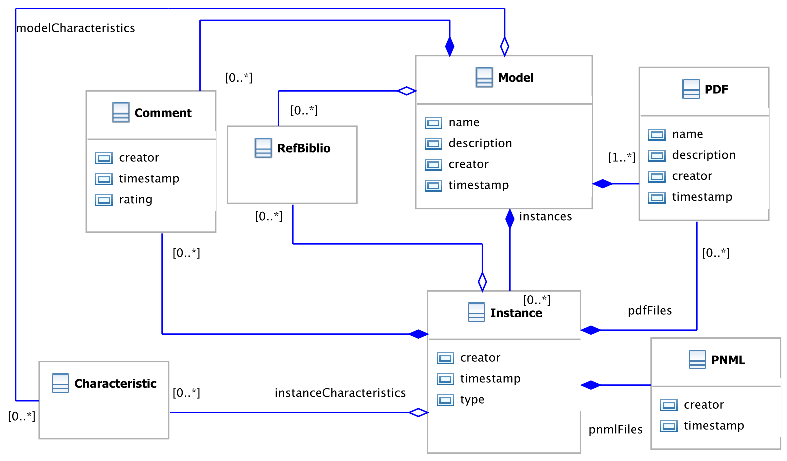


Fig. 1. Detailed view of the Model-Instance semantics group

Requirements i & ii are handled by a clear separation between the problem to represent and its implementation. This is presented in Fig. 1 by the [*Model*] class that represents the problem to model (*e.g.,* the dining philosophers with deadlock) along with its instances ([*Instance*]) (a model expressed in a specific formalism, *e.g.,* CPN or Timed Petri Net (TPN)). The model can be accompanied by [*PDF*] files, containing a description that should be agnostic of a specific formalism or Petri net flavor.

Model creators can also add bibliographies ([*RefBiblio*]), comments ([*Comment*]) and characteristics ([*Characteristic*]) to the problem description. Characteristics are like tags, defining what interesting aspects the model is concerned with (*e.g.,* concurrency, state space explosion, real time...). These artifacts should give a clear and detailed representation and enable subsequent uses of the model by other scientists, like inferring new instances of the model using different formalisms in order to model different aspects of the problem. The problem description can be instantiated ([*Instance*]) several times, for example using different class of Petri nets. Ideally the instance itself should be described using the Petri Net Markup Language (PNML) ([*PNML*]) as it provides a portable and standard way of describing Petri net models, or using PDF documents ([*PDF*]) if PNML is not possible. As for the model ([*Model*]), its [*Instance*] description can be specified using bibliographies ([*RefBiblio*]), comments ([*Comment*]) and characteristics ([*Characteristic*]). Making these fields searchable would enact requirement iv. Because we believe that the only applicable way to maintain such a repository in the long term is a community driven approach (requirement vi), any logged-in user can modify models or instances. For the sake of simplicity we did not represent the versioning mechanism in the DOM, as we consider it a detail concerning the implementation platform.

To fulfill requirement iii, instances ([*Instance*]) and goals ([*Goal*]) are linked together (see Fig. 2). The [*Goal*] is what the benchmark is focusing on. It can be some-

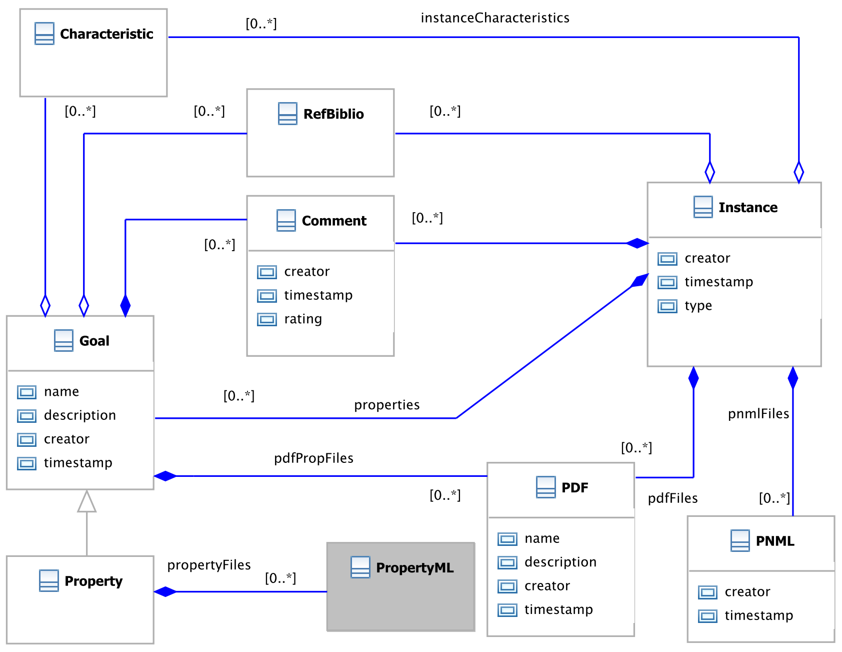


Fig. 2. Detailed view of the Instance-Property semantics group

thing general, like evaulating state space generation performance, or it can also be a [*Property*] to check. Each instance may have several associated goals. Goal descriptions can be enriched using bibliographies ([*RefBiblio*]), comments ([*Comment*]) and characteristics ([*Characteristic*]) (requirement iv). In case the goal is checking a property, a property specification [*Property*] should be included. Unfortunately, there is no standard language for properties equivalent to what PNML is for models. The greyed class [*PropertyML*] represents a future extension to the repository that would allow uploading property files expressed in such a standard language, if and when the community agrees on one.

Fig. 3 proposes a solution to satisfy requirement v. Benchmarks ([*Benchmark*]) are contained by a goal ([*Goal*]). A benchmark is a set of runs ([*Run*]) that have been done on the same platform (to be comparable). Each run contains a result, that is expressed as free text describing, for example, performance information. A run has one and only one context ([*Context*]) that contains all the required information and steps to reproduce the run. These are namely the source files ([*SourceFile*]) and the arguments ([*Argument*]) which the tool ([*Tool*]) should be launched with.

As for the rest of the DOM, [*Benchmark*] and [*Argument*] can be characterized and commented by any contributor.

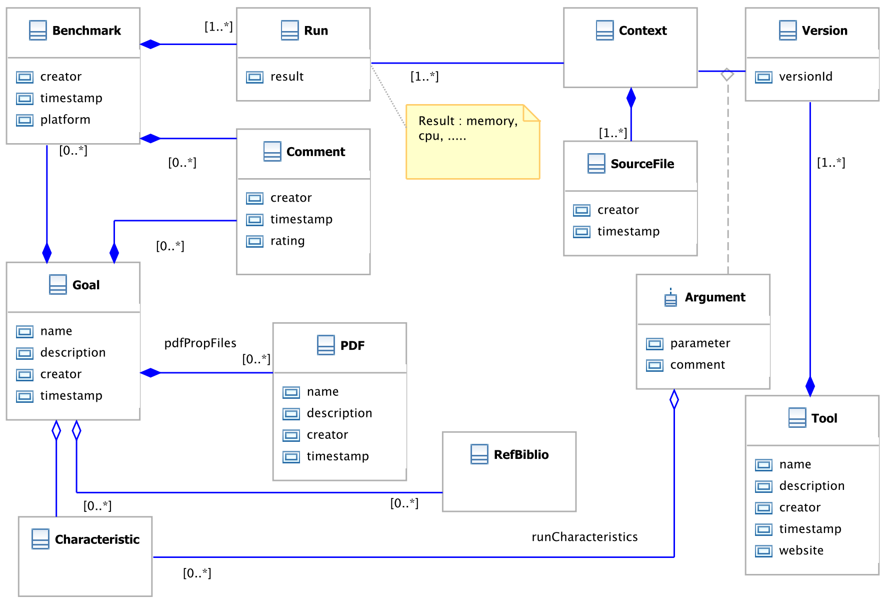


Fig. 3. Detailed view of the Benchmark semantics group

From a functional point of view, we identify three user roles (see Fig. 7):

– A guest that can search and browse the repository (requirement iv).
– A registered user, that inherits from the guest and can also create/update objects, and add comments and evaluations (requirements vi & vii).
– An administrator that can handle basic housekeeping tasks (requirement viii).

Fig. 4 shows the use cases that are executable by the guest user. Guests can apply for registration to gain access to the repository as a registered user. It is worth noting that the registration does not have to be validated by the repository administrator. Indeed, to reduce maintenance cost any human being can register her/himself as a repository user. The fact that it is a human should be enforced by a Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) mechanism. Guests are also able to search the repository, however Denial of Service (DOS) attack should be prevented with some limiting mechanism. Once the results of the search have been returned, users can browse them and eventually they can see the details of a given record. These details present the complete object graph ([*Model*], [*Instance*], [*Goal*] or [*Benchmark*]) related to a given result, as well as their dependencies ([*Comment*], [*PDF*], [*Characteristic*]. . .).

Fig. 5 presents the use cases executable by the registered user. This is the main user role, it inherits from the guest role as described in the main use case diagram. This role has all the capabilities of the guest, plus the ability to create and modify content in the repository. There is no authorization mechanism, that is once users have been logged

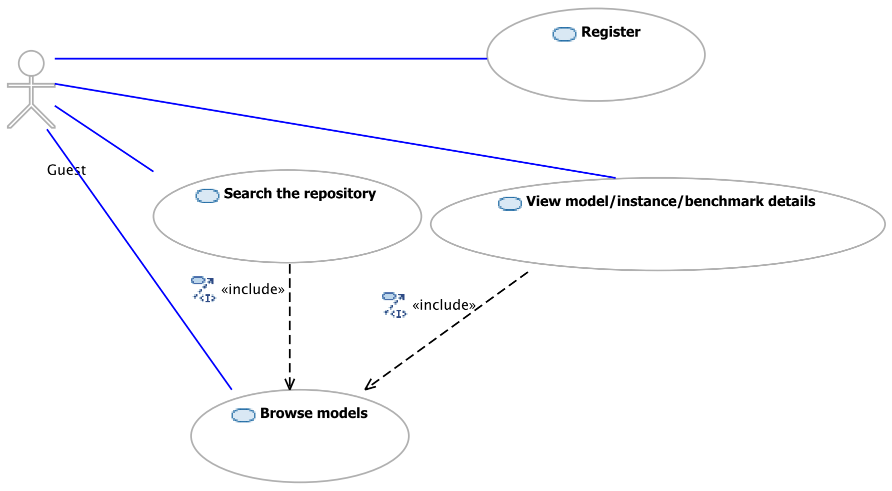


Fig. 4. Detailed view of the guest use cases

they can update any data (in a wikified way: requirement vi). Although ownership is not used for authorization purposes, the repository is protected against malicious users as well as against human mistakes by a versioning mechanism.

Finally, Fig. 6 shows the administrator use cases. This is mainly housekeeping, *e.g.,* removing malicious users or cleaning up characteristics. Since the repository should be driven by the community in a wikified way, the amount of administration should be kept to a minimum (requirement viii).

We will now give two example workflows. The first workflow describes a scientist that is looking for a new model in order to evaluate her own research. The second presents a typical review process involving benchmarks submitted to a computer science conference.

Scientist Sue connects to the repository and is looking for interesting models and properties to check. Since Sue is working on high-level models such as CPN that are also highly-concurrent, she constraints the search by setting criteria such as the instance formalism and by setting some characteristics such as a high concurrency level. She then browses the returned results and finds an interesting model: the dining philosophers. Although the instance is exactly what she is looking for, there is no associated PNML file. She therefore creates a PNML version of the CPN instance of the problem. In order to upload the PNML version she registers and logs in as a registered user. She also leaves some comments about the PNML version of the model and adds some information such as the theoretical bounds of the domains. She comes back a week later, logs in and add a bunch of properties to check. After that, she registers her tool and adds benchmarks. Because she wants to submit a paper to Petri Nets 2012, she gives the reviewers the address of her benchmarks in the repository.

Here comes the second workflow, reviewer Rob has to evaluate Sue's paper and thus has to validate the benchmarks. Rob simply downloads the context [*Context*] of

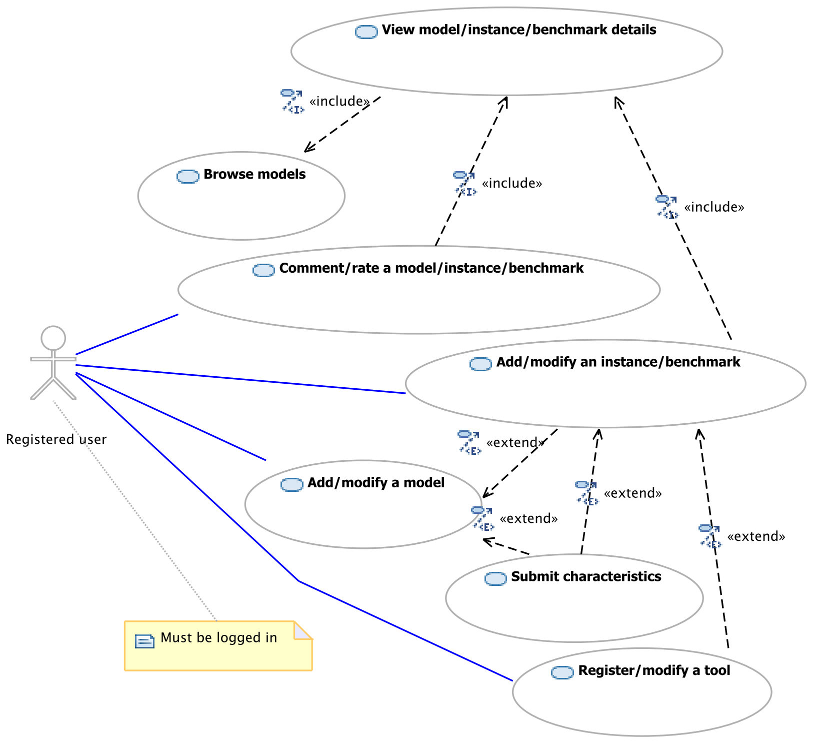


Fig. 5. Detailed view of the registered user use cases

Sue's benchmarks and reproduces them on his multicore machine. He then comments and rates the benchmarks using a special anonymous referee account. He also adds his own benchmarks, that quantify how much this highly-concurrent model benefits from parallel execution on his multiple processors.

5 Discussion

This proposal raises a number of question marks, which should be object for discussion. We identified a list of open issues and potentially critical aspects.

What is the best type of collaborative form? A shared, multiuser repository where each user can contribute with custom material demands for a structure that can manage multiple contributions and concurrent edit. Given the current state of the art in web portals, a way to achieve this could be basing the repository on a wiki. However this raises a number of concerns and points to discuss, *e.g.,* , what is the policy *w.r.t.* permissions, updating and creating new models, and approving changes.

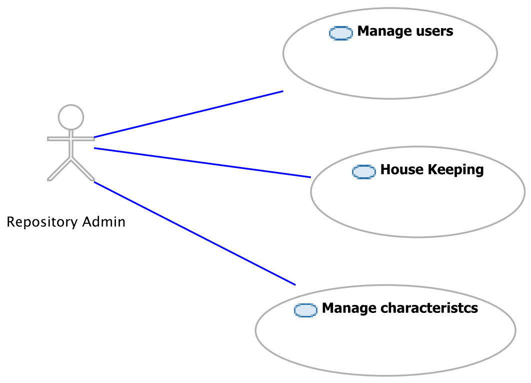


Fig. 6. Detailed view of the admin use cases

Building on existing work. The state of the art reveals that repositories exist that fulfill part of the requirements we are aiming for. Would it be reasonable to extend/revisit existing approaches rather than building new repositories?

Managing characteristics. As it was proposed, characteristics are like a tag cloud. As with tag clouds, one potential issue is duplication and chaos. Careful consideration should be given to the mechanism for submitting new features. It could be possible to rely on users, but this could be weak. Or the admin could approve characteristics for insertion, but this could slow down insertion and create contrasts between the admin and the users. Then again, a semi-automated, knowledge-based mechanism could be devised that tries to detect duplication and suggests to the users to re-use an existing characteristics instead of creating a new one.

Achieving critical mass. The rate of adoption is a key factor in the success of such a repository. The Petri nets community seems to express a strong interest in the repository concept, but probably something should be done to push the repository forward. For example, it could be used as a required standard form to upload benchmarks for articles submitted to the Petri Nets conference.

Workforce. We think that, although the repository can count on the community for building content, there has to be at least a kernel of people who are constantly implied in managing it and keeping it clean and functional. Given the nature of many academic positions, this could be a non-trivial requirement, that demands careful consideration.

6 Conclusion

We presented a proposal for a common repository of benchmark models for the Petri nets community. The proposal is based on a survey led in the community at the beginning of 2010. The purpose of this article is fostering a discussion on how to push forward the state of benchmarking for Petri nets.

While building a repository is a central aspect towards this goal, this is certainly not the only one. Community members should consider how to adopt practices that improve the quality of their results publications. We feel we can share the advice given in [2] when it comes to what the community should do to help itself. Tools should be freely available and accessible, in order to facilitate tuning results. Examples should be published in full, and not only as partial descriptions in already short articles. Notations and languages should be standardized to facilitate interchange. Common benchmarks should be identified, and research articles should really stress the empirical aspects. Furthermore, the scientific community should enforce a policy where empirical benchmarking is a required component for sound publication of tool results.

Initiatives in this direction, together with a common repository, can definitely bring some improvement to the quality of published results.

References

1. Frank R. Abate and Elizabeth Jewell. *The new Oxford American dictionary, second edition*. Oxford University Press, New York :, 2005.
2. George S. Avrunin, James C. Corbett, and Matthew B. Dwyer. Benchmarking finite-state verifiers. *STTT*, 2(4):317–320, 2000.
3. Diyaa addein Atiya, Néstor Cataño, and Geral Lüttgen. Towards a benchmark for model checkers of asynchronous concurrent systems. In *University of Warwick, United Kingdom*, 2005.
4. Radek Pelánek. BEEM: Benchmarks for explicit model checkers. In *in SPIN 07*, pages 263–267. Springer, 2007.
5. R. Goud, Kees M. van Hee, R. D. J. Post, and Jan Martijn E. M. van der Werf. Petriweb: A Repository for Petri Nets. In *ICATPN*, pages 411–420, 2006.
6. Lom Hillah, Rachid Alahyane, Cuauhtemoc Castellanos, Monir Chaouki, and Issam Saïd. pnmlWEB, 2008. http://pnmlweb.lip6.fr/.
7. Stefan Blom and Hubert Garavel. Very Large Transition Systems, VLTS, 2009. http://www.inrialpes.fr/vasy/cadp/resources/benchmark_bcg.html.
8. LASER laboratory, University of Massachusetts. Example repository for finite state verification tools, 2003. http://laser.cs.umass.edu/verification-examples/.

A Appendix

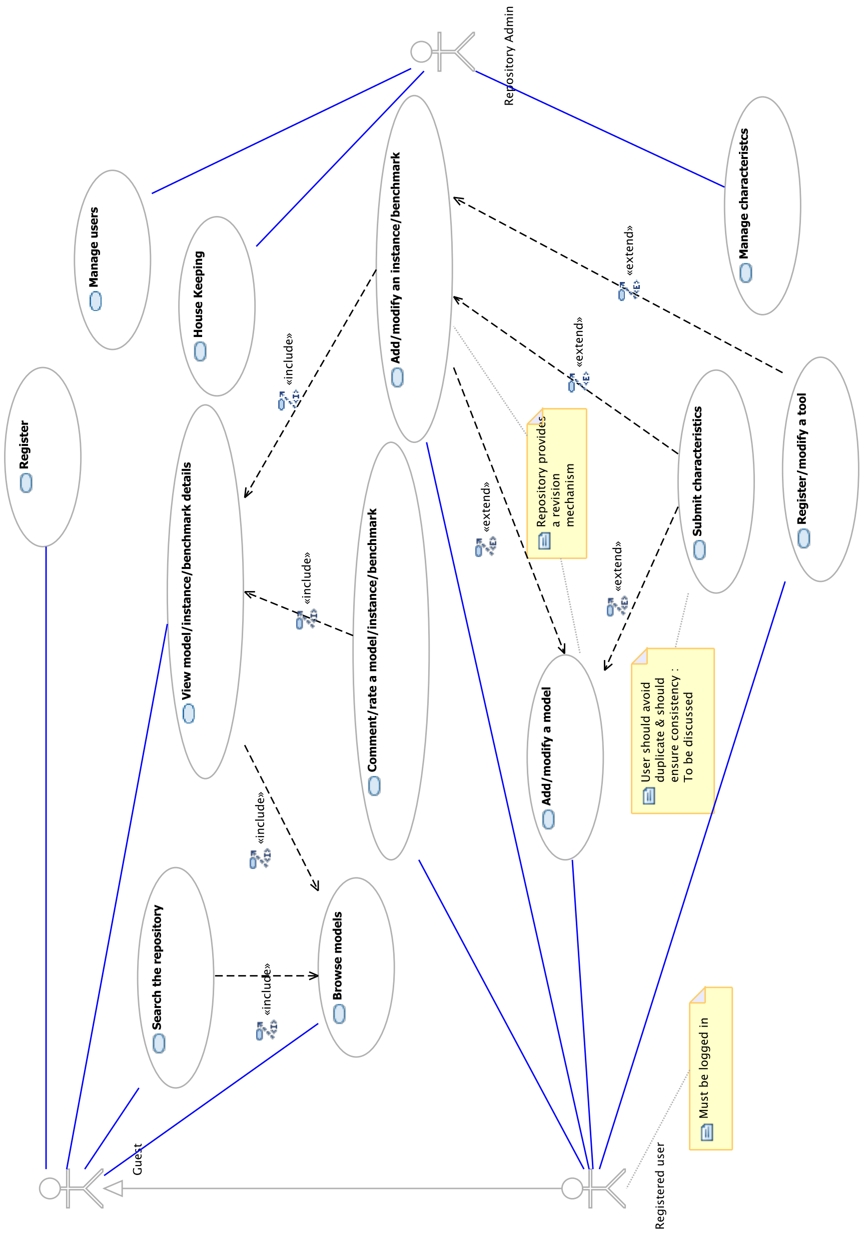

Repository Admin
Manage users
House Keeping
Add/modify an instance/benchmark
Manage characteristcs
Register
View model/instance/benchmark details
Comment/rate a model/instance/benchmark
Repository provides a revision mechanism
Submit characteristics
Register/modify a tool
Add/modify a model
User should avoid duplicate & should ensure consistency : To be discussed
Search the repository
Browse models
Must be logged in
Guest
Registered user
«include»
«extend»

Fig. 7. Requirements for a model repository

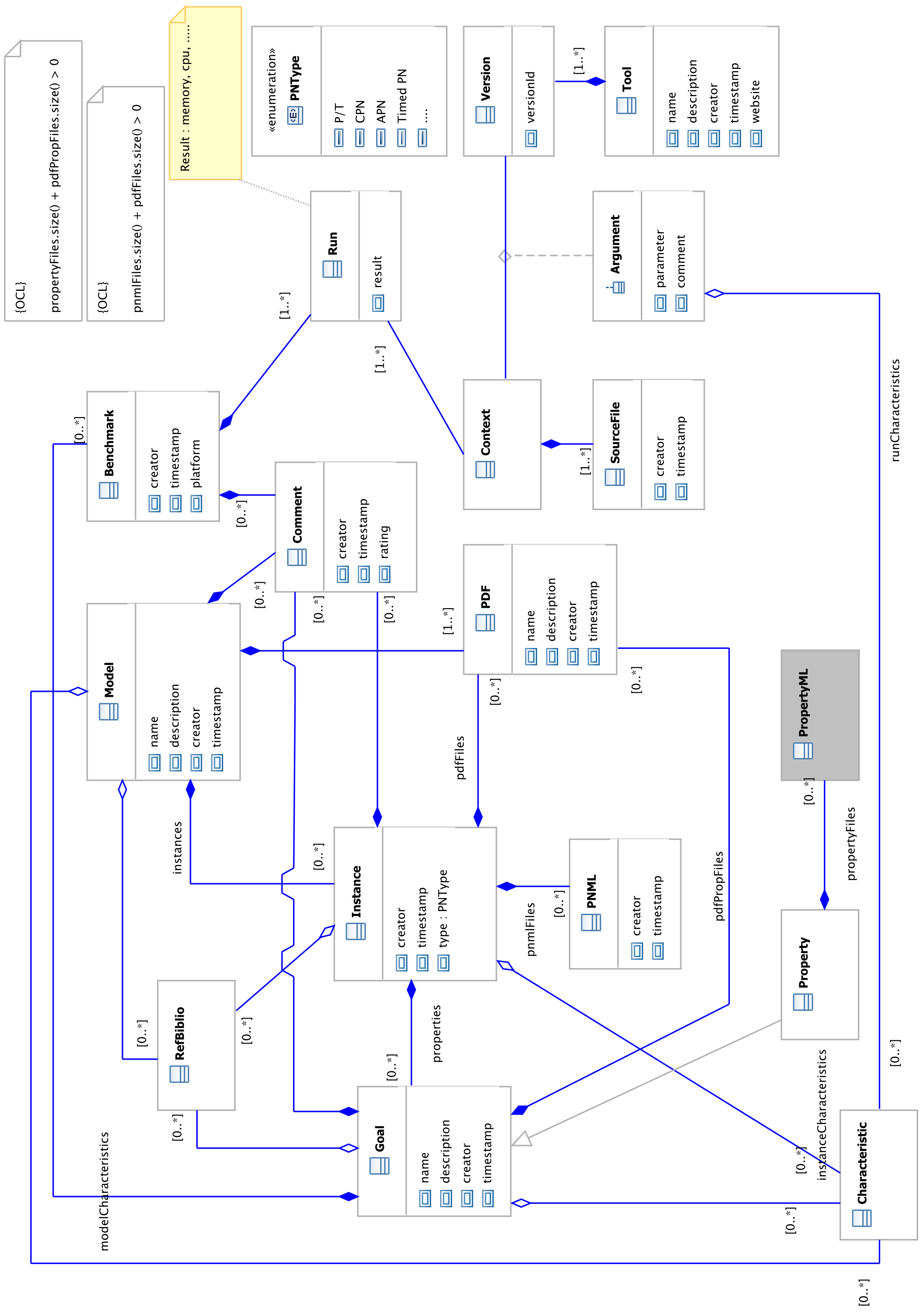


Fig. 8. Proposal for a model repository

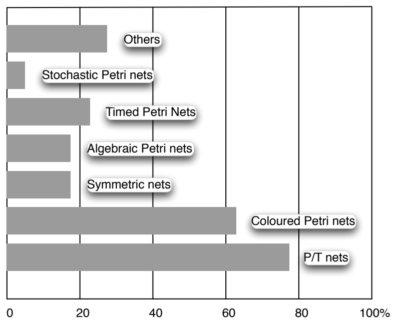


Fig. 9. *Which formalism are you using?*

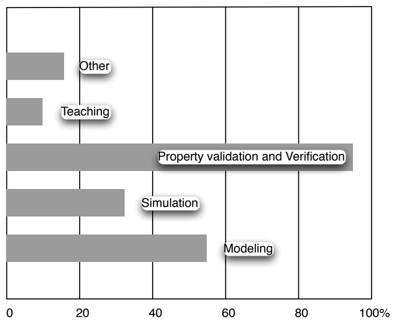


Fig. 10. *What is your main purpose?*

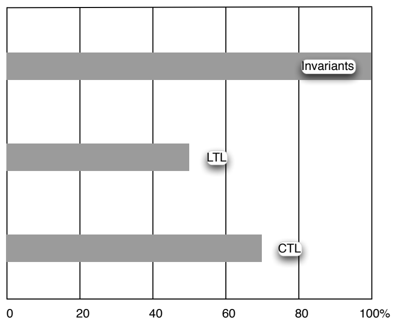


Fig. 11. *Which kind of property do you check?*

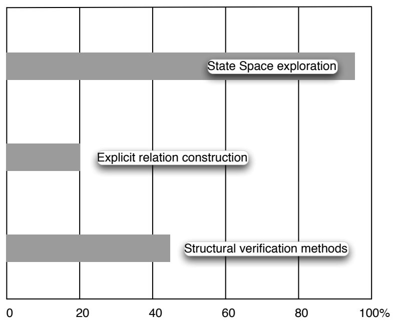


Fig. 12. *Which kind of verification techniques?*

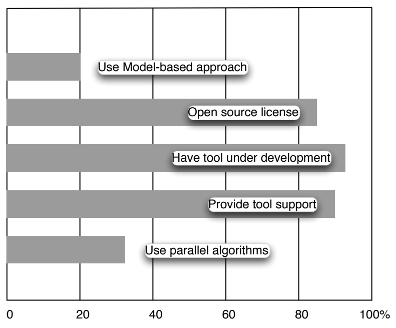


Fig. 13. *Miscellanea*

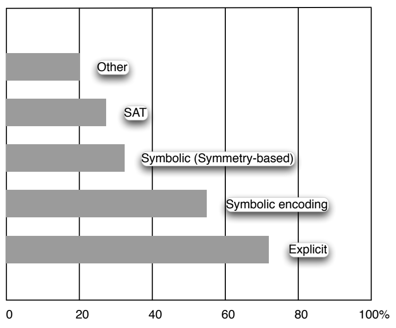


Fig. 14. *What kind of state space techniques?*

Index