

SimHPN: a MATLAB toolbox for simulation, analysis and design with hybrid Petri nets

-draft-

Jorge Júlvez, Cristian Mahulea and Carlos-Renato Vázquez *

December 23, 2014

Abstract

This paper presents a MATLAB embedded package for hybrid Petri nets called *SimHPN*. It offers a collection of tools devoted to simulation, analysis and synthesis of dynamical systems modeled by hybrid Petri nets. The package supports several server semantics for the firing of both, discrete and continuous, types of transitions. Besides providing different simulation options, *SimHPN* offers the possibility of computing steady state throughput bounds for continuous nets. For such a class of nets, optimal control and observability algorithms are also implemented. The package is fully integrated in MATLAB what allows the creation of powerful algebraic, statistical and graphical instruments that exploit the routines available in MATLAB.

Published as:

J. Julvez, C. Mahulea, and C.R. Vazquez, “SimHPN: a MATLAB toolbox for simulation, analysis and design with hybrid Petri nets,” *Nonlinear Analysis: Hybrid Systems*, vol. 6, no. 2, pp. 806-817, May 2012. DOI: <http://doi.org/10.1016/j.nahs.2011.10.001>

*The authors are with The Aragón Institute for Engineering Research (I3A), University of Zaragoza, Maria de Luna 1, 50018 Zaragoza, Spain. E-mail: {julvez,cmahulea,cvazquez}@unizar.es.

This work has been partially supported by the European Community’s Seventh Framework Programme under project DISC (Grant Agreement n. INFOS-ICT-224498), by CICYT - FEDER grants DPI2010-20413 and TIN2007-66523 and by Fundación Aragón I+D.

1 Introduction

Petri nets (PNs)[14, 5] is a mathematical formalism for the description of discrete-event systems, that has been successfully used for modeling, analysis and synthesis purposes of such systems. A key feature of a PN is that its structure can capture graphically fundamental primitives in concurrency theory such as parallelism, synchronization, mutual exclusion, etc. The state of a PN system is given by a vector of non-negative integers representing the marking of its places.

As any other formalism for discrete event systems, PN suffer from the *state explosion problem* which produces an exponential growth of the size of the state space with respect to the initial marking. One way to avoid the state explosion is to relax the integrality constraint in the firing of transitions and deal with transitions that are fired in real amounts. A transition whose firing amount is allowed to be any real number between zero and its enabling degree is said to be a *continuous transitions*. The firing of a continuous transition can produce a real, not integer, number of tokens in its input and output places. If all transitions of a net are continuous, then the net is said to be continuous. If a non-empty proper subset of transitions is continuous, then the net is said to be hybrid [4].

Different time interpretations can be considered for the firing of continuous transitions. The most popular ones are infinite and finite server semantics which represent a first order approximation of the firing frequency of discrete transitions. For a broad class of Petri nets, infinite server semantics offers a better approximation of the steady-state throughput than finite server semantics [12]. Moreover, finite server semantics can be exactly mimicked by infinite server semantics in discrete transitions simply by adding a self-loop place. A third firing semantics, called product semantics, is also frequently used when dealing with biochemical and population dynamics systems.

In this paper we present a new MATLAB embedded software called *SimHPN* that provides support for infinite server and product semantics in both, discrete and continuous, types of transition. A description of a preliminary version of this software can be found in [8, 9]. As far as we know this is the first MATLAB package that enables the analysis and simulation of hybrid nets with these two firing semantics. There already exists a toolbox dealing with discrete Petri nets [13], and one for the so-called first order hybrid Petri nets [15] which provides support for continuous transitions under finite server semantics. The main features of the *SimHPN* toolbox are: (i) simulation of hybrid Petri nets under different server semantics; (ii) computation of steady state throughput bounds; (iii) computation of minimal P-T *semiflows*; (iv) optimal sensor placement; (v) optimal control algorithm; (vi) import models from different graphical Petri net editors.

The paper is organized as follows: Section 2 introduces the formal definition of the hybrid Petri nets supported by *SimHPN*. Section 3 briefly presents the main features of the package. Sections 4, 5 and 6 exemplify those features by applying them to three case studies. Section 7 concludes the paper.

2 Hybrid Petri nets

Hybrid Petri nets [4, 2] represent a powerful modeling formalism that allows the integration of both continuous and discrete dynamics in a single net model. This section defines the class of hybrid nets supported by *SimHPN*. In the following, the reader is assumed to be familiar with Petri nets (PNs) (see [14, 5] for a gentle introduction).

2.1 Untimed Hybrid Petri net systems

Definition 2.1 A Hybrid Petri Net (HPN) system is a pair $\langle \mathcal{N}, \mathbf{m}_0 \rangle$, where: $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$ is a net structure, with set of places P , set of transitions T , pre and post incidence matrices $\mathbf{Pre}, \mathbf{Post} \in \mathbb{R}_{\geq 0}^{|P| \times |T|}$, and $\mathbf{m}_0 \in \mathbb{R}_{\geq 0}^{|P|}$ is the initial marking.

The token load of the place p_i at marking \mathbf{m} is denoted by m_i and the *preset* and *postset* of a node $X \in P \cup T$ are denoted by $\bullet X$ and X^\bullet , respectively. For a given incidence matrix, e.g., \mathbf{Pre} , $\mathbf{Pre}(p_i, t_j)$ denotes the element of \mathbf{Pre} in row i and column j .

In a HPN, the set of transitions T is partitioned in two sets $T = T^c \cup T^d$, where T^c contains the set of continuous transitions and T^d the set of discrete transitions. In contrast to other works, the set of places P is not explicitly partitioned, i.e., the marking of a place is a natural or real number depending on the firings of its input and output transitions. Nevertheless, in order to make net models easier to understand, those places whose marking can be a real non-integer number will be depicted as double circles (see p_1^1 in Fig. 3), and the rest of places will be depicted as simple circles (such places will have integer markings, see p_5^1 in Fig. 3). Continuous transitions are graphically depicted as two bars (see t_4^1 in Fig. 3), while discrete transitions are represented as empty bars (see t_5^1 in Fig. 3), .

Two enabled transitions t_i and t_j are in conflict when they cannot occur at the same time. For this, it is necessary that $\bullet t_i \cap \bullet t_j \neq \emptyset$, and in that case it is said that t_i and t_j are in structural conflict relation. Right and left non negative annullers of the token flow matrix \mathbf{C} are called T- and P-*semiflows*, respectively. A semiflow \mathbf{v} is *minimal* when its support, $\|\mathbf{v}\| = \{i \mid \mathbf{v}(i) \neq 0\}$, is not a proper superset of the support of any other semiflow, and the greatest common divisor of its elements is one. If there exists $\mathbf{y} > 0$ such that $\mathbf{y} \cdot \mathbf{C} = 0$, the net is said to be *conservative*, and if there exists $\mathbf{x} > 0$ satisfying $\mathbf{C} \cdot \mathbf{x} = 0$, the net is said to be *consistent*. As it will be seen, the basic tasks that *SimHPN* can perform on untimed hybrid Petri nets are related to the computation of minimal T- and P-semiflows.

The enabling degree of a continuous transition $t_j \in T$ is:

$$enab(t_j, \mathbf{m}) = \begin{cases} \min_{p_i \in \bullet t_j} \left\lfloor \frac{m_i}{\mathbf{Pre}(p_i, t_j)} \right\rfloor & \text{if } t_j \in T^d \\ \min_{p_i \in \bullet t_j} \frac{m_i}{\mathbf{Pre}(p_i, t_j)} & \text{if } t_j \in T^c \end{cases} \quad (1)$$

Transition $t_j \in T$ is *enabled* at \mathbf{m} iff $enab(t_j, \mathbf{m}) > 0$. An enabled transition $t_j \in T$ can fire in any amount α such that $0 \leq \alpha \leq enab(t_j, \mathbf{m})$ and

$\alpha \in \mathbb{N}$ if $t_j \in T^d$ and $\alpha \in \mathbb{R}$ if $t_j \in T^c$. Such a firing leads to a new marking $\mathbf{m}' = \mathbf{m} + \alpha \cdot \mathbf{C}(\cdot, t_j)$, where $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$ is the token-flow matrix and $\mathbf{C}(\cdot, t_j)$ is its j column. If \mathbf{m} is reachable from \mathbf{m}_0 through a finite sequence σ , the *state (or fundamental) equation*, $\mathbf{m} = \mathbf{m}_0 + \mathbf{C} \cdot \sigma$ is satisfied, where $\sigma \in \mathbb{R}_{\geq 0}^{|T|}$ is the firing count vector. According to this firing rule the class of nets defined in Def 2.1 is equivalent to the class of nets defined in [4, 2].

2.2 Timed Hybrid Petri net systems

Different time interpretations can be associated to the firing of transitions. Once an interpretation is chosen, the state equation can be used to show the dependency of the marking on time, i.e., $\mathbf{m}(\tau) = \mathbf{m}_0 + \mathbf{C} \cdot \sigma(\tau)$. The term $\sigma(\tau)$ is the firing count vector at time τ . Depending on the chosen time interpretation, the firing count vector $\sigma_j(\tau)$ of a transition $t_j \in T^c$ is differentiable with respect to time, and its derivative $f_j(\tau) = \dot{\sigma}_j(\tau)$ represents the *continuous flow* of t_j . As for the timing of discrete transitions, several definitions exist for the flow of continuous transitions. *SimHPN* accounts for infinite server and product server semantics in both continuous and discrete transitions, and additionally, discrete transitions are also allowed to have deterministic delays.

Definition 2.2 A Timed Hybrid Petri Net (THPN) system is a tuple $\langle \mathcal{N}, \mathbf{m}_0, \text{Type}, \lambda \rangle$ where $\langle \mathcal{N}, \mathbf{m}_0 \rangle$ is a HPN, $\text{Type} : T \rightarrow \{id, pd, dd, ic, pc\}$ establishes the time semantics of transitions and $\lambda : T \rightarrow \mathbb{R}_{\geq 0}$ associates a real parameter to each transition related to its semantics.

Any of the following semantics is allowed for a discrete transition $t_i \in T^d$:

- *Infinite server semantics* ($\text{Type}(t_i) = id$): Under infinite server semantics, the time delay of a transition t_i , at a given marking \mathbf{m} , is an exponentially distributed random variable with parameter $\lambda_i \cdot \text{enab}(t_i, \mathbf{m})$, where the integer enabling $\text{enab}(t_i, \mathbf{m})$ represents the number of active servers of t_i at marking \mathbf{m} .
- *Product server semantics* ($\text{Type}(t_i) = pd$): Under product server semantics, the time delay of a transition t_i at \mathbf{m} is an exponentially distributed random variable with parameter $\lambda_i \cdot \prod_{p_j \in \bullet t_i} \left\lfloor \frac{\mathbf{m}(p_j)}{\mathbf{Pre}(p_j, t_i)} \right\rfloor$, where $\prod_{p_j \in \bullet t_i} \left\lfloor \frac{\mathbf{m}(p_j)}{\mathbf{Pre}(p_j, t_i)} \right\rfloor$ is the number of active servers.
- *Deterministic delay* ($\text{Type}(t_i) = dd$): A transition t_i with deterministic delay is scheduled to fire $1/\lambda_i$ time units after it became enabled.

Conflict resolution: When several discrete exponential transitions, under either infinite or product server semantics, are in conflict, a racing policy is adopted, i.e., the one with smaller time delay will fire first.

If a discrete transition with deterministic delay is not in conflict with other transitions, it is fired as scheduled, if it is in conflict then it is fired only if its scheduled firing time is less than the firing time of the conflicting transition. The transition to fire, in the case of several conflicting deterministic transitions with same scheduled firing instance, is chosen probabilistically assigning the

same probability to each conflicting transition. Furthermore after the firing of a deterministic transition, the timers of all the transitions in the same conflict are discarded.

For a continuous transition $t_i \in T^c$ the following semantics are allowed:

- *Infinite server semantics* ($Type(t_i) = ic$): Under *infinite server* the flow of a transition t_i is:

$$f_i = \lambda_i \cdot enab(t_i, \mathbf{m}) = \lambda_i \cdot \min_{p_j \in \bullet t_i} \left\{ \frac{m_j}{Pre(p_j, t_i)} \right\} \quad (2)$$

Such an expression for the flow is obtained from a first order approximation of the discrete case [18] and corresponds to the *variable speed* of ([1])

- *Product server semantics* ($Type(t_i) = pc$): In a similar way to discrete transitions, the continuous flow under product server semantics is given by:

$$f_i = \lambda_i \cdot \prod_{p_j \in \bullet t_i} \left\{ \frac{m_j}{Pre(p_j, t_i)} \right\}$$

The described supported semantics cover the modeling of a large variety of actions usually associated to transitions. For instance, infinite server semantics, which are more general than finite server semantics, are well suited for modeling actions in manufacturing, transportation and logistic systems [4]; product server semantics are specially useful to modeling population dynamics [17] and biochemical reactions [7]; and deterministic delays allow one to represent pure delays and clocks that appear, for instance, when modeling traffic lights in automotive traffic systems [19].

3 The *SimHPN* package

The *SimHPN* simulator supports infinite server and product server semantics for both discrete and continuous transitions. Moreover, deterministic delays with single server semantics are also supported for discrete transitions. Both the data related to the model description, i.e., net structure, initial marking and timing parameter, and the output results, i.e., time trajectories, are MATLAB variables. At the end of the simulation, the user can export the data to the MATLAB workspace where can be used for further analysis. The next two subsections describe the functionality of the graphical interface and the implemented simulation algorithm. Besides simulation, *SimHPN* offers some analysis algorithms which are presented together with the case study in Section 4.

3.1 Graphical interface

The *SimHPN* toolbox (<http://webdiis.unizar.es/GISED/?q=tool/simhpn>) provides a Graphical User Interface (GUI) that enables the user to easily perform simulations and carry out analysis methods. This GUI consists of a MATLAB figure window, exhibiting a *Menu bar* and three control panels: (i) *Drawing Area*, (ii) *Options panel*, and (iii) *Model Management panel*. Fig.

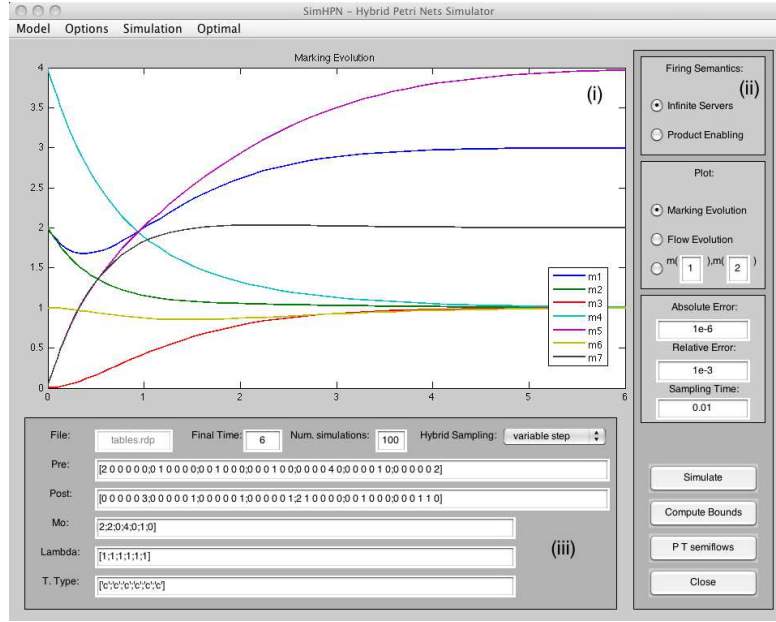


Figure 1: Sketch of the main window of *SimHPN*

1 presents a hard-copy screenshot of the main window opened by *SimHPN* toolbox, where all the component parts of the GUI are visible.

The *Menu bar* (placed horizontally, on the top of the window in Fig. 1) displays a set of four drop-down menus at the top of the window, where the user can select different features available in the *SimHPN* toolbox. These menus are: *Model*, *Options*, *Simulation*, and *Optimal*.

The *Model* menu contains the pop-up menus *Import from Pmeditor*, *Import from TimeNet* and *Import from .mat file* that implement several importing options for the matrices, ***Pre***, ***Post***, ***m₀***, etc, that describe the net system: Such matrices can be introduced manually or through two Petri nets editors: PMEditeur and TimeNet [20]. Moreover, the matrices can be automatically loaded from a *.mat* file (MATLAB file format) or loaded from variables defined in the workspace, this is done just by writing the name of the variable to be used in the corresponding edit boxes.

The *Options* menu contains only the pop-up menu *Show Figure Toolbar* allows to show the characteristic toolbar of the MATLAB figure object that permits, for example, the use of zoom tool on the displayed graphic in the *Drawing Area*.

The *Simulation* menu contains the pop-up menus *Markings to plot*, *Flows to plot*, and *Save results to workspace*. The pop-up menus *Markings to plot*, *Flows to plot* allow the user to select the components of marking vector and flow vector that will be plotted after simulation in the *Drawing area*. The pop-up menu *Save results to workspace* permits to export, after simulation, the marking and flow evolution to variables in the MATLAB workspace.

The *Optimal* menu contains the pop-up menus *Optimal Observability* and *Optimal Control*. Such pop-up menus perform calls to the algorithms for computing optimal steady state and optimal sensor placement for continuous Petri

nets with infinite server semantics (see Section 4 for more details).

The *Drawing area* (located in the left and central side of the window in Fig. 1), is a MATLAB axes object where the trajectories of the simulation results are plotted. The components of markings and flows that will be represented are selected from the menu.

The *Options panel* (placed, as a horizontal bar, on the right part of the window Fig. 1) presents a number of options related to the model. From top to bottom: (a) two radio buttons to select the firing semantics for continuous and discrete exponential transitions; (b) three radio buttons allowing to select the variables to be plotted in the *Drawing Area*, the simulator allows one to plot the evolution of the marking of the places, the evolution of the flow of the transitions and the evolution of the marking of one place vs. the marking of other place; (c) three edit boxes to fix the maximum absolute and relative errors allowed by the simulated trajectory and the sampling time used in simulations (see next subsection for more details on the selection of the sampling time); (d) a *Simulate* button to start a new simulation; (e) a *Compute Bounds* button that computes performance bounds for continuous nets under infinite server semantics; (f) a *P T semiflows* button to compute the minimal P- and T-semiflows of the net, the results are displayed on the MATLAB command window and can be used for future analysis tasks; and (g) a *Close* button to close the *SimHPN* toolbox.

The *Model Management Panel* panel is composed of different edit boxes (placed in the bottom left corner of the window in Fig. 1), where the *SimHPN* toolbox displays the current values of the matrices describing the net system and permits to select the simulation time and the number of simulations to be performed (this last parameter is ignored if the net contains no stochastic transitions). The required matrices for a system in order to be simulated are: ***Pre*** and ***Post*** matrices, initial marking \mathbf{m}_0 , the parameter λ of each transition, and the type of each transition. This last parameter is equal to 'c' for continuous transitions, to 'd' for stochastic discrete transitions and to 'q' for deterministic discrete transitions. Notice that if the type of a transition is 'q' then single server semantics is adopted for its firing and therefore the selection of firing semantics in the *Options panel* will be ignored for this transition.

3.2 Internal simulation

Particular classes of hybrid Petri nets are continuous Petri nets and discrete Petri nets. As *SimHPN* supports hybrid Petri nets, it also supports pure continuous and pure discrete nets. For the sake of computational efficiency, the simulation of the different cases is dealt separately.

Continuous Petri nets. A continuous PN under either infinite or product server semantics is deterministic and is described by a set of differential equations. In such case, the *SimHPN* uses a standard equation solver (ODE function) of MATLAB to simulate the time trajectory of the system.

Discrete Petri nets. On the other hand, a discrete PN under either infinite or product server semantics is stochastic and can be simulated by using an event-base approach, i.e., after each firing the simulator computes the marking reached and the time of the next potential firing of the enabled transitions (stored in variables called *clocks*), next, the simulation time is updated as the minimum of such firing times. The *SimHPN* applies such an approach for discrete PNs. For models having discrete stochastic transitions, *SimHPN* computes the *average*

trajectories (of the marking or firing frequencies of the transitions) obtained after several simulations (the number of simulations is specified by the user using the edit box called *Num. Simulations* of the *Model Management Panel*).

Hybrid Petri nets. The simulation becomes more complex for hybrid PNs, since neither an ODE solver nor an event-base simulation can be efficiently used. In such case, a discrete-time simulation is carried out. The sampling time can be fixed and specified by the user (through the box labeled as *Sampling Time* of the *Model Management Panel*). If a sampling is not specified (introducing a negative number as a sampling time), *SimHPN* computes a suitable sampling based on a trial simulation. It is also possible to use a variable sampling time, which is computed during the simulation (this is specified in the *Hybrid Sampling* popup menu of the *Model management Panel*). For hybrid models, *SimHPN* performs five basic operations at each sampling instant: 1) the simulation time is updated, 2) the scheduled discrete transitions are fired (according to the clocks) and all the clocks are updated; 3) the marking is updated according to the flow of the continuous transitions (using a finite difference equation for the continuous subnet); 4) the enabling degree of the discrete transitions and clocks are updated (a change in the continuous marking can enable or disable discrete transitions); and 5) the next sampling time is computed.

Conflicts involving stochastic (discrete) transitions are solved by a race policy. For conflicts involving deterministic transitions, the first rule is a race policy. If the conflict remains (discrete transitions that should fire at the same time) then it is randomly solved by considering equal firing probabilities. After the firing of a deterministic transition, the clocks of the other deterministic transitions in the conflict are discarded.

Algorithm 1: Update clocks

```

1 Input:  $\langle \mathcal{N}, \mathbf{m}_0, Type, \lambda \rangle, \mathbf{m}, clocks, S$ 
2 Output: clocks
3 for all the  $t_i \in T^d$  do
4   if  $t_i$  is not enabled then
5      $clocks_i := \infty$ ;
6   else if ( $t_i$  is enabled and ( $t_i \in S$  or  $t_i$  is newly enabled)) then
7     switch the value of  $Type(t_i)$  do
8       case  $Type(t_i) = dd$ 
9          $\theta = 1/\lambda_i$ 
10      case  $Type(t_i) = id$ 
11        get  $\theta \in \mathbb{R}_{>0}$  from an exponential p.d.f with parameter
           $\lambda_i \cdot enab(t_i, \mathbf{m})$ 
12      case  $Type(t_i) = pd$ 
13        get  $\theta \in \mathbb{R}_{>0}$  from an exponential p.d.f with parameter
           $\lambda_i \cdot \prod_{p_j \in \bullet t_i} [\mathbf{m}(p_j) / Pre(p_j, t_i)]$ 
14     $clocks_i := \tau + \theta$ 

```

Procedure 'Update clocks' described in Alg. 1, updates the clock variables associated to discrete transitions (the schedule time). In each iteration of the

Algorithm 2: Simulation of a HPN using a variable sampling time

```

1 Input:  $\langle \mathcal{N}, \mathbf{m}_0, Type, \lambda \rangle$ 
2 Output:  $\mathbf{m}$  /* Time evolution of marking */
3  $\tau = 0$ 
4  $\mathbf{m}(0) = \mathbf{m}_0$ 
5  $\forall t_i \in T^d : clocks_i = \infty$ 
6  $S = \emptyset$ 
7 Call procedure(Update clocks)
8  $\Delta\tau = \min(DSet)$ , where
    $DSet = \{clocks_i | t_i \in T^d\} \cup \{0.1/f_i | t_i \in T^c, f_i > 0\}$ 
9 while  $\tau \leq \text{total simulation time}$  do
10    $\tau := \tau + \Delta\tau$ 
11    $\mathbf{m}(\tau) := \mathbf{m}(\tau - \Delta\tau)$ 
12   /* Fire scheduled discrete transitions */
13   while  $\exists t_j \in T^d$  such that  $clocks_j \leq \tau$  do
14     Let  $t_j \in T^d$  such that  $clocks_j \leq \tau$ 
15     Solve conflicts involving  $t_j$ 
16     Define  $\Delta\sigma = \begin{cases} \Delta\sigma_k := 1 & \text{if } t_k \text{ fires} \\ \Delta\sigma_k := 0 & \text{if } t_k \text{ does not fire} \end{cases}$ 
17      $\mathbf{m}(\tau) := \mathbf{m}(\tau) + \mathbf{C} \cdot \Delta\sigma$ 
18      $S := ||\Delta\sigma||$ 
19     Call procedure(Update clocks)
20   /* Fire continuous transitions */
21    $\mathbf{f} := \mathbf{0}$ ;
22   forall the  $t_i \in T$  do
23     switch the value of  $Type(t_i)$  do
24       case  $Type(t_i) = id$  or  $Type(t_i) = pd$ 
25          $f_i := 0$ 
26       case  $Type(t_i) = ic$  :
27          $f_i := \lambda_i \cdot enab_i(\mathbf{m})$ 
28       case  $Type(t_i) = pc$  :
29          $f_i := \lambda_i \cdot \prod_{p \in \bullet t_i} \mathbf{m}[p] / Pre[p, t_i]$ 
30    $\mathbf{m}(\tau) = \mathbf{m}(\tau) + \mathbf{C} \cdot \mathbf{f} \cdot \Delta\tau$ 
31    $S = \emptyset$ 
32   Call procedure(Update clocks)
33   /* Compute next sampling */
34    $\Delta\tau = \min(DSet)$ , where
    $DSet = \{clocks_i - \tau | t_i \in T^d\} \cup \{0.1/f_i | t_i \in T^c, f_i > 0\}$ 

```

algorithm, the set S contains the transition to fire and *newly enabled* expresses the transitions that were not enabled in the previous iteration and become enabled in the current one. If the transition is not enabled, its clock is fix to ∞ . Otherwise, if it is enabled and either it has been fired at the current time τ or has just become enabled its clock is computed as follows:

1. $1/\lambda_i$ if the transition t_i is deterministic;
2. a random number obtained from an exponential p.d.f. if it is exponential (infinite server or product semantics).

Algorithm 2 is the main algorithm for simulation, it performs only one simulation (realization). It consists in the following: First, the current time is initialized to zero, the current marking to \mathbf{m}_0 , the *clocks* to ∞ , the set S as empty (the set of currently fired transitions, required for calling the procedure 'updating clocks'), and the function 'Update clocks' defined in Alg. 1 is called. Then, a suitable sampling $\Delta\tau$ is computed, which is used when the variable sampling option has been selected. Such a sampling is the minimum amount between the next firing time (schedule) of the discrete transitions, and $1/10$ of the maximum instantaneous flow of the continuous transitions (in our experience, this ratio $1/10$ provides a good enough approximation). The simulation loop starts in line 9. Inside the loop, the simulation time is updated, the scheduled discrete transitions are fired according to the conflict resolution policy described above (after that, clocks are updated), next, the continuous transitions are fired (after that, clocks are updated again), and finally, the next sampling is computed. The simulation loop stops when the the current simulation time is larger than the total simulation time, indicated by the user.

4 An assembly line

The Petri net system in Fig. 2 represents an assembly line with kanban strategy (see [21]). The system has two stages that are connected by transition t_{14} . The first stage is composed of three lines (starting from p_2 , p_3 and p_4 respectively) and three machines (p_{23} , p_{24} and p_{25}). Places p_{26} , p_{27} and p_{28} are buffers at the end of the lines. The second stage has two lines that require the same machine/resource p_{18} . The number of kanban cards is given by the marking of places p_2 , p_3 and p_4 for the first stage, and by the marking of p_{32} for the second stage. The system demand is given by the marking of p_1 . We will make use of this net system to illustrate some of the features of *SimHPN*.

Given that all transitions represent actions that can potentially have high working loads, all transitions are considered continuous. Moreover, infinite server semantics will be adopted for all of them. Let the initial marking be $\mathbf{m}_0(p_1) = \mathbf{m}_0(p_{18}) = \mathbf{m}_0(p_{23}) = \mathbf{m}_0(p_{24}) = \mathbf{m}_0(p_{25}) = \mathbf{m}_0(p_{29}) = \mathbf{m}_0(p_{32}) = 1$, $\mathbf{m}_0(p_{26}) = \mathbf{m}_0(p_{27}) = \mathbf{m}_0(p_{28}) = 3$ and the marking of the rest of places be equal to zero. Let us assume that the firing rates of the transitions are $\lambda(t_2) = \lambda(t_3) = \lambda(t_4) = \lambda(t_8) = \lambda(t_9) = \lambda(t_{10}) = \lambda(t_{14}) = \lambda(t_{15}) = \lambda(t_{17}) = \lambda(t_{19}) = \lambda(t_{20}) = 10$, $\lambda(t_1) = \lambda(t_5) = \lambda(t_6) = \lambda(t_7) = \lambda(t_{11}) = \lambda(t_{12}) = \lambda(t_{13}) = \lambda(t_{16}) = \lambda(t_{18}) = \lambda(t_{21}) = 1$.

Computation of minimal P-T *semiflows*: *SimHPN* implements Alg. 3, proposed in [16] to compute the minimal P-*semiflows* of a Petri net. The

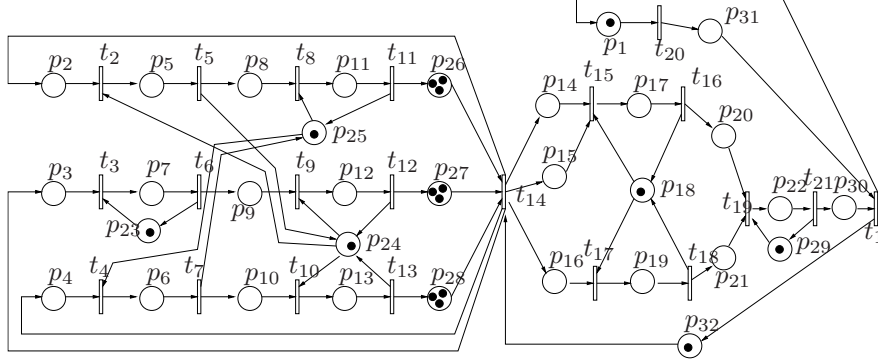


Figure 2: An assembly line with kanban strategy.

same algorithm applied on the transpose of the incidence matrix yields the minimal T-*semiflows* of the net. Notice that P and T-*semiflows* just depend on the structure of the net and not on the continuous or discrete nature of the transitions. The result of applying the algorithm on the net in Fig. 2 is the set of 12 minimal P-*semiflows* that cover every place, i.e., it is conservative, and the set that contains the only minimal T-*semiflow* which is a vector of ones, i.e., it is consistent.

Algorithm 3: Computation of minimal P-*semiflows*

- 1 Input: $\langle \mathcal{N} \rangle$
 - 2 Output: \mathbf{D}
 - 3 $\mathbf{A} := \mathbf{C}$ ($m \times n$);
 - 4 $\mathbf{D} :=$ Identity matrix of n rows
 - 5 **for** $i=1$ **to** n **do**
 - 6 Add to matrix $[\mathbf{D}|\mathbf{A}]$ every row that is a positive linear combination of couples of rows of $[\mathbf{D}|\mathbf{A}]$ and are opposite (in sign) to the i^{th} column of \mathbf{A}
 - 7 Remove from $[\mathbf{D}|\mathbf{A}]$ those rows in which the i^{th} column of \mathbf{A} is not null
 - 8 The rows of matrix \mathbf{D} are the P-*semiflows* of the net
-

Throughput bounds: When all transitions are continuous and work under infinite server semantics, the following programming problem can be used to compute an upper bound for the throughput, i.e., flow, of a transition ([10]):

$$\begin{aligned}
 \max \{ & \phi_j \mid \boldsymbol{\mu}^{ss} = \mathbf{m}_0 + \mathbf{C} \cdot \boldsymbol{\sigma}, \\
 & \phi_j^{ss} = \lambda_j \cdot \min_{p_i \in \bullet t_j} \left\{ \frac{\mu_i^{ss}}{Pre(p_i, t_j)} \right\}, \forall t_j \in T, \\
 & \mathbf{C} \cdot \boldsymbol{\phi}^{ss} = \mathbf{0}, \\
 & \boldsymbol{\mu}^{ss}, \boldsymbol{\sigma} \geq \mathbf{0} \}.
 \end{aligned} \tag{3}$$

This non-linear programming problem is difficult to solve due to the minimum operator. When a transition t_j has a single input place, the equation reduces to (4). And when t_j has more than an input place, it can be relaxed (linearized) as (5).

$$\phi_j^{ss} = \lambda_j \cdot \frac{\mu_i^{ss}}{Pre(p_i, t_j)}, \text{ if } p_i = \bullet t_j \quad (4)$$

$$\phi_j^{ss} \leq \lambda_j \cdot \frac{\mu_i^{ss}}{Pre(p_i, t_j)}, \forall p_i \in \bullet t_j, \text{ otherwise} \quad (5)$$

This way we have a single linear programming problem, that can be solved in polynomial time. Unfortunately, this LPP provides in general a non-tight bound, i.e., the solution may be non-reachable for any distribution of the tokens verifying the *P-semiflow* load conditions, $\mathbf{y} \cdot \mathbf{m}_0$. One way to improve this bound is to force the equality for at least one place per synchronization (a transition with more than one input place). The problem is that there is no way to know in advance which of the input places should restrict the flow. In order to overcome this problem, a branch & bound algorithm can be used to compute a reachable steady state marking.

SimHPN implements such a branch & bound algorithm to compute upper throughput bounds of continuous nets under infinite server semantics. For the system in Fig. 2 with the mentioned \mathbf{m}_0 and λ the obtained throughput bound for t_1 is 0.3030. Given that the only T-semiflow of the net is a vector of ones, this value applies as an upper bound for the rest of transitions of the net.

Optimal Sensor Placement:

Assuming that each place p can be measured at a different cost $c(p) > 0$ the optimal sensor placement problem of continuous nets under infinite server semantics is to decide the set of places $P_o \subseteq P$ to be measured such that the net system is observable at minimum cost. This problem can be seen as a Set Covering Problem which is NP-hard in the strong sense [6]. For a set of places P_o , let K_{P_o} be the set of observable places. It will be said that P_o is covering K_{P_o} . The problem is to determine a set P_{o_i} with minimum cost such that the covered elements contain all the places of the net.

Considering that n is the number of places, the brute force approach to solve this problem is to try all subsets of places of size n , $n - 1, \dots, 1$. From those subsets ensuring the observability of the continuous Petri net system, the one with minimum cost is taken. In order to reduce the number of the subsets, some graph-based properties can be used. The idea is to group the set of places in equivalence classes such that only one place per class can belong to the optimal solution. These equivalence classes are called *threads* and are the places belonging to the maximally connected subnets finishing in an attribution place (place with more than one input transitions) or in a join (transition with more than one input place). Initially, all places of the net belong only to one thread. The following algorithm can be used to reduce the number of elements from threads.

These reductions preserve the optimality of the solution and the covering problem can be started using the resulted threads. It is necessary to generate all combinations taking at most one place from each thread and then check the observability of the system. If the system is observable, the solution is kept if has a cost lower than the candidate solution. A good choice is starting with the first places of each thread and going backward since the following property is true: if the system is not observable for the current set of measured places, it is not necessary to advance in the threads because the system will not be observable.

Algorithm 4: Reduce the places from threads

```

1 Input:  $\langle \mathcal{N}, c \rangle$ 
2 Output: threads
3 for every cycle of  $\mathcal{N}$  without attributions and joins do
4   | choose the place with minimum cost;
5   | remove the other places from all threads.
6 Compute all maximal paths without joins and attributions that finish in
  a measured place.
7 Eliminate all places belonging to the previous paths from all threads.
8 for each thread do
9   | if  $\exists$  a path from  $p$  to  $p'$  and  $c(p') > c(p)$  then
10  |   | delete  $p'$  from the thread.
11 Make the threads disjoint.

```

The algorithm is still exponential but the structural properties presented can reduce drastically the number of observability checks. For the continuous nets considered in this section, measuring all input places in join transitions the net system is observable. This is also the solution of the optimal sensor placement problem for any cost associated to transitions.

Optimal Steady-State: The only action that can be performed on a continuous Petri nets is to slow down the flow of its transitions. If a transition can be controlled (its flow reduced or even stopped), we will say that is a *controllable* transition. The forced flow of a controllable transition t_j becomes $f_j - u_j$, where f_j is the flow of the unforced system, i.e. without control, and u_j is the control action $0 \leq u_j \leq f_j$.

In production control is frequent the case that the profit function depends on production (benefits in selling), working process and amortization of investments. Under linear hypothesis for fixed machines, i.e., λ defined, the profit function may have the following form:

$$\mathbf{w}^T \cdot \mathbf{f} - \mathbf{z}^T \cdot \mathbf{m} - \mathbf{q}^T \cdot \mathbf{m}_0 \quad (6)$$

where \mathbf{f} is the throughput vector, \mathbf{m} the average marking, \mathbf{w} a gain vector w.r.t. flows, \mathbf{z}^T is the cost vector due to immobilization to maintain the production flow and \mathbf{q}^T represents depreciations or amortization of the initial investments.

The algorithm used to compute the optimal steady state flow (and marking) is very much alike the one used to compute the performance bounds, with the difference that the linear programming problem that needs to be solved is:

$$\left\{ \begin{array}{l} \max\{\mathbf{w}^T \cdot \mathbf{f} - \mathbf{z}^T \cdot \mathbf{m} - \mathbf{q}^T \cdot \mathbf{m}_0 \mid \mathbf{C} \cdot \mathbf{f} = 0, \\ \mathbf{m} = \mathbf{m}_0 + \mathbf{C} \cdot \boldsymbol{\sigma}, \\ f_j = \lambda_j \cdot \left(\frac{m_i}{Pre(p_i, t_j)} \right) - v(p_i, t_j), \\ \forall p_i \in \bullet t_j, v(p_i, t_j) \geq 0 \\ \mathbf{f}, \mathbf{m}, \boldsymbol{\sigma} \geq 0 \end{array} \right. \quad (7)$$

where $v(p_i, t_j)$ are slack variables. These slack variables give the control action for each transition. For more details on this topic, see ([11]).

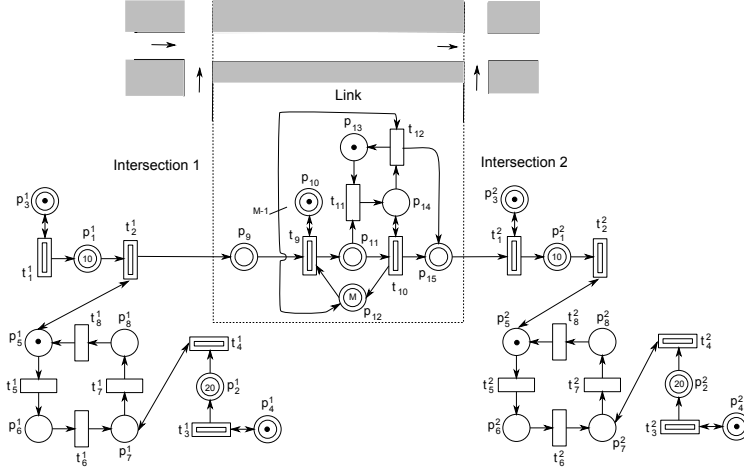


Figure 3: *HPN* model of 2 intersections connected by a link.

5 A traffic system

Here, a hybrid PN model, that represents a traffic system consisting of two (one-way streets) intersections connected by a (one-way street) link, is introduced and simulated. The proposed example is shown in fig. 3 (studied in [19] for control purposes). In this, the dynamic of the vehicles is represented by continuous nodes, while the traffic lights are modeled as discrete.

Let us firstly explain the model of one intersection. Places $\{p_1^1, p_2^1\}$ represent the queues of vehicles before crossing the intersection 1. Cars arrive through $\{t_1^1, t_3^1\}$, being transitions of type *ic* constrained by self-loops $\{p_3^1, p_4^1\}$ that represent the number of servers (street lanes). Vehicles depart through t_2^1 or t_4^1 (type *ic*) when the traffic light enabled them, i.e., when there is a token in p_5^1 or p_7^1 , respectively. The traffic light for this intersection is represented by nodes $\{p_5^1, p_6^1, p_7^1, p_8^1, t_5^1, t_6^1, t_7^1, t_8^1\}$, which describe the sequence of the traffic light stages. In this, the transitions are of type *dd*. A token in p_5^1 means a green signal for the queue p_1^1 , but red for p_2^1 .

Similarly, a token in p_7^1 represents a green signal for the queue p_2^1 but red for p_1^1 . Places p_6^1 and p_8^1 represent intermediate stages when the traffic light is amber for one queue but red for the other, so no car can cross the intersection. Similarly, nodes with the superscript 2, i.e., $\{p_x^2, t_x^2\}$, represent the nodes of the second intersection and its traffic light. In this, the place p_9^2 and the transition t_9^2 (type *dd*) have been added in order to simulate the offset, i.e., the relative time between the cycles of both traffic lights, which is given by the delay of t_9^2 . The output flow of intersection 1 feeds the second one through a link, which imposes a constant delay (given by the delay of t_{11}^1). A detailed explanation of the link model can be found in [19]. Let us just mention here that, due to the traffic light, vehicles departing intersection 1 describe a bursting signal (like platoons or batches of cars) that arrives to the second intersection through t_1^2 .

Let us consider the following delays: for $\{t_5^1, t_6^1, t_7^1, t_8^1\}$ the delays are (20, 5, 20, 4) seconds (in the same order). For $\{t_1^1, t_2^1, t_3^1, t_4^1\}$ are (1, 1/3, 1/3, 1/5) seconds, for $\{t_1^2, t_2^2, t_3^2, t_4^2\}$ are (1/3, 1/5, 1, 1/3) seconds, and for the link

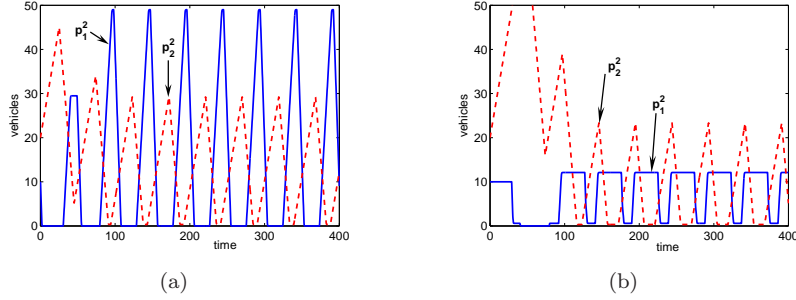


Figure 4: Queue lengths at intersection 2 obtained with delays for $\{t_5^2, t_7^2, t_9^2\}$ as a) $\{20, 20, 0.1\}$ and b) $\{14, 26, 29\}$.

$\{t_9, t_{10}, t_{11}, t_{12}\}$ are $(1/10, 1/3, 30, 1/3)$ seconds (the link delay is $\theta_{11} = 30$ seconds). The initial marking is as described in fig. 3.

The goal in this example is to obtain, through simulations, suitable switching delays for the second traffic light, in order to reduce the queue lengths at intersection 2. The parameters to optimize are the green periods (amber periods are fixed and equal to $\theta_6^2 = 5$ and $\theta_8^2 = 4$), i.e., the delays of t_5^2, t_7^2 , and the offset represented by the delay of t_9^2 . Fig. 4 shows the evolution of the queues for the first 400 seconds, for two cases: a) with green periods 20 seconds and no offset, and b) with green periods 14 for the queue p_1^2 and 26 for the queue p_2^2 and with an offset of 29 seconds. Note that the second combination of parameters provide shorter queues. For this, the effect of the offset is very important. This example shows that simulations based on hybrid PN models can provide information about the optimal parameters for traffic lights (duration of stages and offset), in order to improve the performance in neighboring traffic intersections.

6 A biochemical system

This section presents and simulates a biochemical system modeled by continuous Petri nets. In most chemical models, the different amounts of chemical substances are expressed in terms of concentrations rather than as whole numbers of molecules. This implies that the state of the system is given by a vector of real numbers, where each component represents the concentration of a compound. On the other hand, the dynamics of most reactions is driven by the mass action law, what roughly implies that the speed of a reaction is proportional to the product of the concentrations of the reactants. These facts make continuous Petri nets under product semantics an appealing modeling formalism for biochemical systems.

The net system in Fig. 5 models a signaling pathway described and studied in [3]. More precisely, the net is a graphical representation of the Extracellular signal Regulated Kinase (ERK) signaling pathway regulated by Raf Kinase Inhibitor Protein (RKIP). The marking of each place represents the concentration of the compound associated to it, and the transitions represent the different chemical reactions that take place (see [3] for a more detailed description of the pathway). Notice that, although the net has conflicts, the assumed product

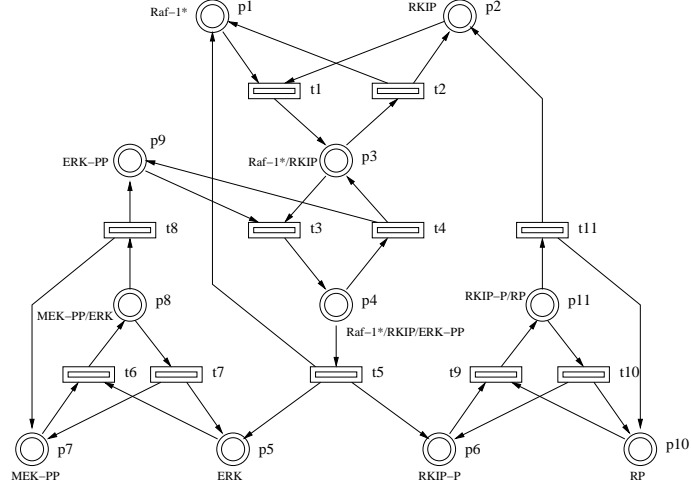


Figure 5: Petri net modeling the ERK signaling pathway regulated by RKIP.

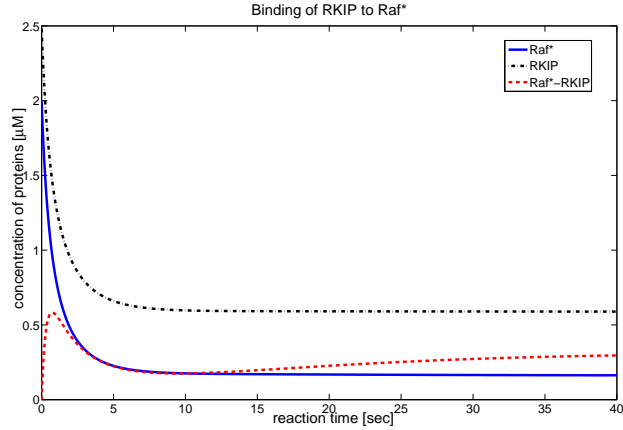


Figure 6: Time evolution of Raf-1*, RKIP and their complex Raf-1*/RKIP.

semantics fully determines the flows of all continuous transitions, and therefore it is not necessary to impose a conflict resolution policy.

Since the state of the system is expressed as concentration levels, every transition is considered continuous and product server semantics is adopted. The parameter λ estimated in [3] is $\lambda = [0.53, 0.0072, 0.625, 0.00245, 0.0315, 0.8, 0.0075, 0.071, 0.92, 0.00122, 0.87]$. As initial concentrations of the compounds we take the following values: $\mathbf{m}_0 = [2, 2.5, 0, 0, 0, 0, 0, 2.5, 0, 2.5, 3, 0]$.

Figures 6 and 7 show the time evolution of some of the compounds in the system along 40 time units. In particular, Fig. 6 shows the dynamics of Raf-1*, RKIP and their complex Raf-1*/RKIP, and Fig. 7 shows the activity of MEK-PP which phosphorylates and activates ERK. As discussed in [3], intensive simulations can be used to perform sensitivity analysis with respect to the variation of initial conditions.

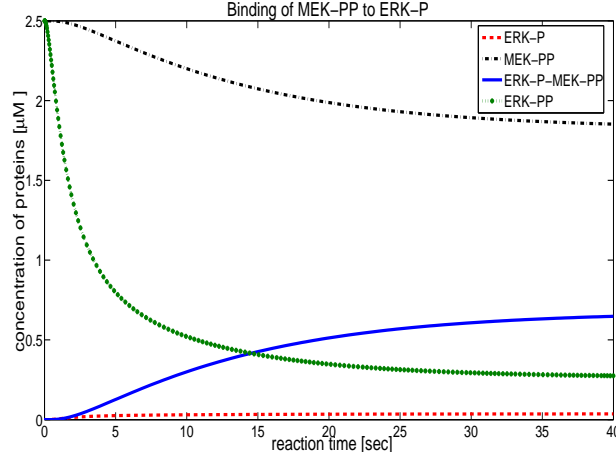


Figure 7: Activity of MEK-PP which phosphorylates and activates ERK.

7 Conclusions

This paper has presented a new MATLAB package, called *SimHPN*, that allows us to perform several analysis and synthesis tasks on hybrid Petri nets working under different server semantics. In particular, *SimHPN* provides procedures to compute minimal P and T - *semiflows*, throughput bounds, optimal steady state and optimal sensor placement.

Additionally, *SimHPN* is able of simulating hybrid Petri nets evolving under infinite server and product semantics. The package is equipped with a Graphical User Interface that offers a friendly interaction with the user.

References

- [1] H. Alla and R. David. Continuous and hybrid Petri nets. *Journal of Circuits, Systems, and Computers*, 8(1):159–188, 1998.
- [2] F. Balduzzi, G. Menga, and A. Giua. First-order hybrid Petri nets: a model for optimization and control. *IEEE Trans. on Robotics and Automation*, 16(4):382–399, 2000.
- [3] 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 Corrado Priami, editor, *Computational Methods in Systems Biology*, volume 2602 of *Lecture Notes in Computer Science*, pages 127–141. Springer Berlin, Heidelberg, 2003.
- [4] R. David and H. Alla. *Discrete, Continuous and Hybrid Petri Nets*. Springer-Verlag, 2010. 2nd edition.
- [5] F. DiCesare, G. Harhalakis, J. M. Proth, M. Silva, and F. B. Vernadat. *Practice of Petri Nets in Manufacturing*. Chapman & Hall, 1993.
- [6] M.R. Garey and D.S. Johnson. *Computers and Interactability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

- [7] M. Heiner, D. Gilbert, and R. Donaldson. Petri nets for systems and synthetic biology. In M. Bernardo, P. Degano, and G. Zavattaro, editors, *Formal Methods for Computational Systems Biology*, Lecture Notes in Computer Science, pages 215–264. Springer Berlin, Heidelberg, 2008.
- [8] J. Júlvez and C. Mahulea. SimHPN: a MATLAB toolbox for continuous Petri nets. In *Proc. of the 10th Workshop on Discrete Event Systems*, pages 24–29, Berlin, Germany, August 2010.
- [9] J. Júlvez, C. Mahulea, and C.R. Vázquez. Analysis and Simulation of Manufacturing Systems using SimHPN toolbox. In *Proc. of the 7th IEEE Conf. on Automation Science and Engineering*, pages 432–437, Trieste, Italy, August 2011.
- [10] J. Júlvez, L. Recalde, and M. Silva. Steady-state performance evaluation of continuous mono-T-semiflow Petri nets. *Automatica*, 41(4):605–616, 2005.
- [11] C. Mahulea, A. Ramírez, L. Recalde, and M. Silva. Steady state control reference and token conservation laws in continuous Petri net systems. *IEEE Trans. on Autom. Science and Engineering*, 5(2):307–320, 2008.
- [12] C. Mahulea, L. Recalde, and M. Silva. Basic Server Semantics and Performance Monotonicity of Continuous Petri Nets. *Discrete Event Dynamic Systems: Theory and Applications*, 19(2):189 – 212, June 2009.
- [13] M. Matcovschi, C. Mahulea, and O. Pastravanu. Petri Net Toolbox for MATLAB. In *11th IEEE Mediterranean Conference on Control and Automation MED’03*, Rhodes, Greece, July 2003.
- [14] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
- [15] F. Sessego, A. Giua, and C. Seatzu. HYPENS: a Matlab tool for timed discrete, continuous and hybrid Petri nets. In *Application and Theory of Petri Nets 2008*, volume 5062 of *Lecture Notes in Computer Science*, pages 419–428. Springer-Verlag, 2008.
- [16] M. Silva. *Las Redes de Petri: en la Automática y la Informática*. AC, 1985.
- [17] M. Silva and L. Recalde. Réseaux de Petri et relaxations de l’intégralité: Une vision des réseaux continus. In *Conférence Internationale Francophone d’Automatique (CIFA 2000)*, pages 37–48, 2000.
- [18] M. Silva and L. Recalde. Petri nets and integrality relaxations: A view of continuous Petri nets. *IEEE Trans. on Systems, Man, and Cybernetics*, 32(4):314–327, 2002.
- [19] C.R. Vázquez, H.Y. Sutarto, R. Boel, and M. Silva. Hybrid petri net model of a traffic intersection in an urban network. In *2010 IEEE Multiconference on Systems and Control*, Yokohama, Japan, 09/2010 2010.
- [20] A. Zimmermann and M. Knoke. Timenetsim - a parallel simulator for stochastic petri nets. In *Proc. 28th Annual Simulation Symposium*, pages 250–258, Phoenix, AZ, USA, 1995.

- [21] A. Zimmermann, D. Rodríguez, and M. Silva. A Two Phase Optimisation Method for Petri Net Models of Manufacturing Systems. *Journal of Intelligent Manufacturing*, 12(5):421–432, October 2001.