Analysis and Simulation of Manufacturing Systems using SimHPN toolbox

Jorge Júlvez, Cristian Mahulea, and Carlos-Renato Vázquez

Abstract—SimHPN is a software tool embedded in MATLAB that has been developed for simulation, analysis and design of systems modeled by hybrid Petri nets. This paper is an extension of our previous work [1] where only continuous Petri nets have been considered. In the current extension we consider infinite server semantics for the continuous part and exponential and/or deterministic firing delays for the discrete part. In particular, the new facilities of SimHPN allow the use of purely discrete or purely continuous models. As an application, we investigate the simulation and analysis of manufacturing systems.

I. INTRODUCTION

Petri nets (PN) are a mathematical formalism for the description of discrete-event systems, successfully used for modeling, analysis and synthesis of such systems. One of their main features is that their state spaces belong to the cartesian product of sets of non-negative integers [2]. Another key feature of PN is their capacity to represent graphically and visualize primitives such as parallelism, synchronization, mutual exclusion, etc.

As any other formalism for discrete event systems, PN suffer from the state explosion problem especially when the system is heavily populated. Among the different procedures to overcome this problem, fluidification is a promising one. In the case of PN this leads to continuous Petri nets if the system is completed relaxed or hybrid Petri nets (HPN) if only a partial relaxation is considered [3]. In HPNs, the firing of continuous transitions is performed in real amounts, and the firing of discrete transitions in natural amounts. As a consequence of this, the marking of a place can be either a natural or a real number. Different time interpretations for the firing of transitions can be considered, being infinite and finite server semantics the most popular ones. A third firing semantics, called product semantics, can be used to study nets obtained by decolorization [4] and population dynamics.

The SimHPN toolbox was designed to offer specific instruments for simulation, analysis and synthesis of discrete event systems modeled by hybrid Petri nets. In particular, infinite server semantics is assumed for continuous transitions while for discrete transitions both deterministic and exponential firing delays can be considered. Its embedding in the MATLAB environment presents the considerable advantage (with respect to other PN software) of creating powerful algebraic, statistical and graphical instruments, which exploit the high quality routines available in MATLAB. Moreover, this MATLAB orientation of the SimHPN was intended to permit further developments.

The paper is organized as follows: Section II introduces the formal definition of HPN that will be considered. Section III creates an overview of the main tools developed for handling HPN in MATLAB. In Section IV a manufacturing example is considered in order to illustrate the futures of SimHPN, and finally in Section V some conclusions and future works are given.

II. HYBRID PETRI NETS: NOTATIONS AND DEFINITIONS

Hybrid Petri nets [3], [5] 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 [2], [6] for a gentle introduction).

A. Untimed Hybrid Petri net systems

Definition 2.1: A Hybrid Petri Net (HPN) system is a pair \(\langle N, m_0 \rangle\), where: \(N = \langle P, T, Pre, Post \rangle\) is a net structure, with set of places \(P\), set of transitions \(T\), pre and post incidence matrices \(Pre, Post \in \mathbb{R}_{\geq 0}^{\left|P\right|\times\left|T\right|}\), and \(m_0 \in \mathbb{R}_{\geq 0}^{|P|}\) is the initial marking.

The token load of the place \(p_i\) at marking \(m\) is denoted by \(m_i\), and the preset and postset of a node \(x \in P \cup T\) are denoted by \(^xP\) and \(^xP\), respectively. For a given incidence matrix, e.g., \(Pre, Pre(p_i, t_j)\) denotes the element of \(Pre\) 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, and thus, the input and output transitions of a place can be both continuous and discrete. This way the marking of a place is a natural or real number depending on the firings of its input and output transitions.

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 \(^t_i \cap ^t_j \neq \emptyset\), and in that case it is said that \(t_i\) and \(t_j\) are in structural conflict relation. Right and left non negative annihilers of the token flow matrix \(C\) are called T- and P-semiflows, respectively. A semiflow \(v\) is minimal when its support, \(\|v\| = \{ i \mid v(i) \neq 0 \}\), 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 \(y > 0\) such that
The enabling degree of a transition $t_j \in T$ is:

$$
enab(t_j, m) = \begin{cases} 
\min_{p_i \in t_j} \frac{m_i}{\text{Pre}(p_i, t_j)} & \text{if } t_j \in T^d \\
\min_{p_i \in t_j} \frac{m_i}{\text{Pre}(p_i, t_j)} & \text{if } t_j \in T^c 
\end{cases}
$$

(1)

Transition $t_j \in T$ (continuous or discrete) is enabled at $m$ iff $\text{enab}(t_j, m) > 0$. An enabled transition $t_j \in T$ can fire in any amount $0 < \alpha \leq \text{enab}(t_j, m)$, where $\alpha \in \mathbb{N}$ 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 $m' = m + \alpha \cdot C(\cdot, t_j)$, where $C = \text{Post} - \text{Pre}$ is the token-flow matrix and $C(\cdot, t_j)$ is its $j$ column. If $m$ is reachable from $m_0$ through a finite sequence $\sigma$, the state (or fundamental) equation $m = m_0 + C \cdot \sigma$ is satisfied, where $\sigma \in \mathbb{R}_+^{\text{inj}}$ 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 [3], [5].

### B. 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., $m(\tau) = m_0 + C \cdot \sigma(\tau)$. The term $\sigma(\tau)$ is the firing count vector at time $\tau$. 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 $f_j(\tau) = \sigma_j(\tau)$ 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 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 $(N, m_0, Type, \lambda)$ where $(N, m_0)$ is a HPN, $Type : T \rightarrow \{c, d, q\}$ 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.

The following two delay types are allowed for a discrete transition $t_i \in T^d$:

- **Stochastic** ($Type(t_i) = d$): The time to fire of an enabled discrete transition with stochastic delay follows an exponentially distributed random variable with parameter $\lambda_i \cdot \text{enab}(t_i, m)$.

- **Deterministic delay** ($Type(t_i) = q$): A transition $t_i$ with deterministic delay is scheduled to fire $1/\lambda_i$ time units after it became enabled. If $t_i$ is not in conflict with other transitions it is fired as scheduled, if it is in conflict then it is fired only if its schedule firing time is less than the firing time of the conflicting transition. The transition fires in conflict, in the case of several conflicting deterministic transitions with same scheduled firing instance, is randomly chosen assigning the same probability to each conflicting transition. Furthermore, an enabling memory is assumed, i.e., after the firing of a deterministic transition, the timers of all the transitions in the same conflict are discarded.

For a continuous transition $t_i \in T^c$, infinite server semantics ($Type(t_i) = c$) is assumed. The flow of a transition $t_i$ is given by:

$$f_i = \lambda_i \cdot \text{enab}(t_i, m) = \lambda_i \cdot \min_{p_i \in t_i} \left\{ \frac{m_j}{\text{Pre}(p_j, t_i)} \right\}
$$

(2)

Such an expression for the flow is obtained from a first order approximation of the discrete net [4] and corresponds to the variable speed of [7].

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 [3]; and deterministic delays allow one to represent pure delays and clocks that appear, for instance, when modeling traffic lights in automotive traffic systems [8].

### III. SimHPN toolbox for MATLAB

In the current version of SimHPN, the only firing semantics for the timed continuous transition is infinite server semantics while deterministic (constant) or exponential firing delays can be associated with timed discrete transitions. In the case of conflicting transitions with identical deterministic delays, the same probability is assumed to each transition. For the exponential transitions, a racing policy is adopted when two or more transitions are in conflict, i.e., the one with smaller time delay will fire first. At the end of the simulation, the user can export the data to the MATLAB workspace where it can be used for further analysis.

#### A. Graphical interface

The SimHPN toolbox [1] (http://webdiis.unizar.es/GISED/?q=tool/simhpn) provides a Graphical User Interface (GUI) to perform the simulations and analysis procedures. The data of the net system can be introduced either manually or through Petri nets editors: PMEditeur or TimeNet [9]. Moreover, the matrices can be automatically loaded from a .mat file or loaded from variables defined in the MATLAB workspace just writing the name of the desired variable that want to be opened in the corresponding edit boxes. 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. 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 facilities available in the SimHPN toolbox. These menus are: Model, Options, Simulation, and Optimal. The Model
menu (containing the pop-up menus Import from Pmediator, Import from TimeNet, Import from .mat file) offers facilities for importing models from two Petri net graphical editors or from a .mat file. The Options menu (containing 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 facility on the displayed graphic in Drawing Area. The Simulation menu (containing the pop-up menus Markings to plot, Flows to plot, and Save results to workspace) provides tools for selecting the components of marking vector and flow vector that will be represented after a simulation in the Drawing area and a tool that permits to export, after a simulation, the marking and flow evolution to variables in the MATLAB workspace. The Optimal menu (containing the pop-up menus Optimal Observability and Optimal Control) permits calling the algorithms for computing optimal steady state and optimal sensor placement for continuous Petri nets with infinite server semantics.

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

The Options panel (placed, as an horizontal bar, on the right part of the window Fig. 1) presents a number of options related with the model. From top to bottom: (i) two radio buttons that permit selecting the firing semantics for continuous and discrete exponential transitions. In the actual implementation only infinite server semantics is allow; (ii) three radio buttons allowing the desired evolutions that are plotted in the Drawing Area; (iii) three edit boxes to change the errors and sampling time used in simulations; (iv) Simulate button that starts a new simulation; (v) Compute Bounds that computes performance bounds for continuous nets under infinite server semantics; (vi) P T semiflows computes the P and T semiflows; and (vii) Close button that closed 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 for the model that is loaded and permits to select the simulation time and the number of simulations when hybrid nets are downloaded.

B. Internal simulation

A continuous PN under infinite 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.

A discrete PN under infinite server semantics is stochastic and can be simulated by using an event-based 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 approach for discrete PNs. For models having discrete stochastic transitions the output of the SimHPN is the average trajectories (of the marking or throughput) obtained after several simulations (the number of this is specified by the user in the corresponding edit box of Model Management Panel).

The simulation becomes more complex for hybrid PNs, since neither an ODE solver nor an event-based simulation can be efficiently used. In such case, a discrete-time simulation is achieved. The sampling time can be variable, computed during the simulation as $\Delta \tau = \min(DSet)$, where $DSet = \{\text{clocks}_{i} - \tau | t_{i} \in T^{d} \} \cup \{0.1/f_{i}| t_{i} \in T^{c}, f_{i} > 0\}$, i.e., $\Delta \tau$ is the minimum between the next scheduled firing of a discrete transition and 10% of the average delays (the inverse of the flow) of the continuous transitions. As another option, the sampling time can be fixed and settled by the user. If a sampling is specified as zero or negative, SimHPN computes a suitable sampling based on a trial simulation, in which variable sampling is used, and then the minimum $\Delta \tau$ computed is used for the rest of simulations.

For hybrid models, SimHPN performs four basic operations at each sampling: 1) it fires the corresponding discrete transitions (according to the clocks) and updates all the clocks; 2) updates the marking due to the flow of the continuous transitions (using a finite difference equation for the continuous subnet); 3) updates the enabling degree of the discrete transitions (a change in the continuous marking can enable or disable discrete transitions); and 4) it computes the next sampling and updates the simulation time.

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 with the same scheduled firing instances) 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.

IV. A MANUFACTURING SYSTEM

The Petri net system in Fig. 2 represents an assembly line with kanban strategy adapted from [10]. 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.

Let us first assume that all transitions are continuous and work under infinite server semantics. Let the initial marking be $m_{0}(p_{1}) = m_{0}(p_{32}) = 10, m_{0}(p_{18}) = m_{0}(p_{23}) = m_{0}(p_{24}) = m_{0}(p_{25}) = m_{0}(p_{29}) = 1, m_{0}(p_{26}) = m_{0}(p_{27}) = m_{0}(p_{28}) = 30$ and the marking of the rest of places be equal to zero. Let us assume that the firing rates of the transitions are $\lambda(t_{2}) = \lambda(t_{3}) = \lambda(t_{4}) = \lambda(t_{8}) = \lambda(t_{9}) = \lambda(t_{10}) = \lambda(t_{14}) = \lambda(t_{15}) = \lambda(t_{17}) = \lambda(t_{19}) = \lambda(t_{20}) = 10, \lambda(t_{1}) = \lambda(t_{5}) = \lambda(t_{6}) = \lambda(t_{7}) = \lambda(t_{11}) = \lambda(t_{12}) = \lambda(t_{13}) = \lambda(t_{18}) = \lambda(t_{19}) = \lambda(t_{21}) = 1$.

Computation of minimal P-T semiflows: SimHPN implements the algorithm proposed in [11] to compute the minimal P and T-semiflows of a Petri 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.

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 [12]:

$$\max\phi_{j} \mid \mu^{ss} = m_{0} + \{C \cdot \sigma, \phi^{ss} = \lambda_{j} \cdot \min_{t \in T, \forall t_{j} \in T} \left\{ \frac{\mu^{ss}}{Pre(p_{i}, t_{j})} \right\} \}$$

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).

$$\phi^{ss}_{j} = \lambda_{j} \cdot \frac{\mu^{ss}}{Pre(p_{i}, t_{j})}, \text{if } p_{i} \neq t_{j} \quad (4)$$

$$\phi^{ss}_{j} \leq \lambda_{j} \cdot \frac{\mu^{ss}}{Pre(p_{i}, t_{j})}, \text{if } p_{i} \neq t_{j}, \text{otherwise} \quad (5)$$

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, $y \cdot m_{0}$. 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 $m_{0}$ and $A$ 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 can be measured at a different cost, the optimal sensor placement problem of continuous Petri Nets under infinite server semantics is to decide the set of places to be measured such
that the net system is observable at minimum cost. Measuring a place allows the observation of a set of others ("covered" by that measure) but, the problem is not a simple covering one [13]. The question is studied at the structural level in [14] and the results obtained are used in the implementation of an algorithm to reduce the computational burden. In the case of the manufacturing system under consideration, all input places in synchronization transitions should be measured. In fact, the places with minimum cost coincide in this case with the set of places ensuring observability of the system.

Optimal Steady-State: The only action that can be performed on a continuous $P\!N$ 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., $\lambda$ defined, the profit function may have the following form:

$$w^T \cdot f - z^T \cdot m - q^T \cdot m_0$$

where $f$ is the throughput vector, $m$ the average marking, $w$ a gain vector w.r.t. flows, $z^T$ is the cost vector due to immobilization to maintain the production flow and $q^T$ 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 hybrid mode provides a better approximation to the discrete one. The linear programming problem that needs to be solved is:

$$\begin{align*}
\max \{ & w^T \cdot f - z^T \cdot m - q^T \cdot m_0 \mid C \cdot f = 0, \\
& m = m_0 + C \cdot \sigma, \\
& f_j = \lambda_j \left( \frac{m}{\mathcal{P}(p_i, t_j)} \right) - v(p_i, t_j), \\
& \forall p_i \in \mathcal{T}, v(p_i, t_j) \geq 0, \\
& f, m, \sigma \geq 0 \}
\end{align*}$$

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 [15].

**Simulation:** SimHPN enables us to simulate the net model as continuous, discrete and hybrid. This allows us to efficiently compare the timed behavior of the system under different modeling approximations.

Fig. 3, 4 and 5 show the time evolution of the marking of places $p_{11}$ and $p_{26}$. Each plot includes three trajectories: the dashed trajectories correspond to the marking of the place when all the transitions are taken as continuous (this trajectory is clearly deterministic); the solid trajectory corresponds to a model in which all transitions are discrete with stochastic delays (in this case the average of 1000 simulation is plot); and the dash-dot trajectory is associated to a model in which all transitions except $t_1$, $t_{14}$, $t_{26}$ and $t_{21}$ are discrete (again the plots correspond to the average of 1000 simulations). It can be observed, that the continuous and hybrid trajectories represent a good approximation to the discrete one.

**V. Conclusions**

The new features of the SimHPN toolbox permit the use of hybrid Petri nets for the analysis and design of manufacturing systems with high level of complexity. The considered case study illustrates the role played by PN techniques, embedded in the powerful software environment offered by MATLAB, in approaching the performance evaluation for various structures of manufacturing systems. In the future, we plan to extend the firing semantics supported by the tool considering the product enabling semantics extensively used to model biochemical reactions.

**REFERENCES**


