HYPENS: a Matlab tool for timed discrete, continuous and hybrid Petri nets

Fausto Sessego, Alessandro Giua, Carla Seatzu

Dip. Ingegneria Elettrica ed Elettronica, Università di Cagliari, Italy

{fausto.sessego,giua,seatzu}@diee.unica.it

Abstract

HYPENS is an open source tool to simulate timed discrete, continuous and hybrid Petri nets. It has been developed in Matlab to allow designer and user to take advantage of several functions and structures already defined in Matlab, such as optimization routines, stochastic functions, matrices and arrays, etc. The tool can also be easily interfaced with other Matlab programs and be used for analysis and optimization via simulation. The large set of plot functions available in Matlab allow one to represent the results of the simulation in a clear and intuitive way.

Published as:

F. Sessego, A. Giua, C. Seatzu, "HYPENS: a Matlab tool for timed discrete, continuous and hybrid Petri nets," *Applications and Theory of Petri Nets*: Proc. 29th Int. Conf. on Applications and Theory of Petri nets 2008 (Xi'an, China), June 23-27, 2008. Lecture Notes in Computer Science No. 5062, pp. 419-428, Springer-Verlag, 2008.

I. INTRODUCTION

In this paper we present a Matlab tool for the simulation of timed Petri nets (PN), called *HYPENS* (HYbrid PEtri Nets Simulator) to emphasize that it deals not only with *discrete* nets, but with *continuous* and *hybrid* nets as well.

In many applications dealing with complex systems, a plant has a discrete event dynamics whose number of reachable states is typically very large, and problems of realistic scale quickly become analytically and computationally untractable. To cope with this problem it is possible to give a continuous approximation of the ”fast” discrete event dynamics by means of *continuous Petri nets*, i.e., nets obtained from discrete nets by ”fluidification” [4], [10].

In general, different fluid approximations are necessary to describe the same system, depending on its discrete state. Thus, the resulting models can be better described as *hybrid Petri nets* (HPN) that combine discrete and continuous dynamics [4], [5]. Several HPN models have been defined (see [5]). The model considered in HYPENS is called *First–Order Hybrid Petri nets* (FOHPN) because its continuous dynamics are piece-wise constant. FOHPN were originally presented in [3] and have been successfully used to model and analyze manufacturing systems [2].

In the last years several tools for the simulation of timed discrete PN have been proposed. Very few tools on the contrary, also deal with hybrid PN: we are aware of *HISim* [7] and *SIRPHYCO* [9].

SIRPHYCO is the most complete in terms of modeling power and performance analysis, but still it presents the following limitations. (a) It has a limited number of modeling primitive. As an example: only stochastic discrete transitions with exponential firing delays are considered; it only assumes infinite-server semantics, thus if we want to consider a finite server semantics we have to add dummy places that complicate the model. (b) It uses a ”reserved marking policy” ([4]), i.e., as soon as a discrete transition is enabled, the tokens necessary for its firing are reserved and cannot be used to enable other transitions. In other words, conflicts are solved at the enabling time not at the firing time. (c) Conflict resolution, both between discrete and continuous transitions, are only solved by priorities. (d) The input graphical interface is not practical for large nets, and only a few statistics can be be computed from the simulation run. (e) It is not an open source software and cannot be easily interfaced with other programs.

HYPENS overcomes the above limitations, and presents several other advantages. (a) It has

many useful modeling primitives such as: finite/infinite server semantics, general stochastic firing delays for discrete transitions. (b) It does not use reserved marking but the tokens in a place can concurrently enable several transitions. In other words, conflicts are solved at the firing time: this policy can be shown to be more general that based on reserved marking. (c) It offers several conflict resolution policies as we will discuss later. (d) It uses a textual input interface but provides several statistical data both numerically and graphically. (e) It is an open source software written in Matlab, to allow designer and user to take advantage of several functions and structures already defined in Matlab and to easily interface it with other programs.

A final remark concerns the issue of *conflict resolution* that is fundamental in any PN simulator [1]. For discrete transitions we use a general approach that combines both priorities and random weighted choices as in [1]: this is coded in the structure of the net. In the case of continuous transitions, on the contrary, we assume that the choice of the firing speed vector $\boldsymbol{v}$ is made at run-time and represents a control input. Thus, for solving conflicts among continuous transitions we use a more general optimization rule: at each step the net evolves so as to (myopically) optimize a linear performance index $J(\boldsymbol{v})$.

The software, manual and demos of HYPENS can be downloaded from [8].

II. HYBRID PETRI NETS

We recall the FOHPN formalism used by HYPENS, following [3].

Net structure: A FOHPN is a structure $N = (P, T, Pre, Post, \mathcal{D}, \mathcal{C})$.

The set of *places* $P = P_d \cup P_c$ is partitioned into a set of *discrete* places P_d (represented as circles) and a set of *continuous* places P_c (represented as double circles). The cardinality of P, P_d and P_c is denoted n, n_d and n_c.

The set of *transitions* $T = T_d \cup T_c$ is partitioned into a set of discrete transitions T_d and a set of continuous transitions T_c (represented as double boxes). The cardinality of T, T_d and T_c is denoted q, q_d and q_c.

The *pre-* and *post-incidence functions* that specify the arcs are (here $\mathbb{R}_0^+ = \mathbb{R}^+ \cup \{0\}$): $Pre, Post : P_c \times T \to \mathbb{R}_0^+$, $P_d \times T \to \mathbb{N}$. We require that $\forall t \in T_c$ and $\forall p \in P_d$, $Pre(p,t) = Post(p,t)$, so that the firing of continuous transitions does not change the marking of discrete places.

Transitions in T_d may either be deterministic or stochastic. In the case of deterministic transitions the function $\mathcal{D} : T_d \to \mathbb{R}_0^+$ specifies the timing associated to timed discrete transitions.

In the case of stochastic transitions $\mathcal{D}$ defines the parameter(s) of the distribution function corresponding to the timing delay. The function $\mathcal{C} : T_c \rightarrow \mathbb{R}_0^+ \times \mathbb{R}_\infty^+$ specifies the firing speeds associated to continuous transitions (here $\mathbb{R}_\infty^+ = \mathbb{R}^+ \cup \{\infty\}$). For any continuous transition $t_j \in T_c$ we let $\mathcal{C}(t_j) = [V_j', V_j]$, with $V_j' \leq V_j$: V_j' represents the *minimum firing speed* (mfs), V_j represents the *maximum firing speed* (MFS).

The *incidence matrix* of the net is defined as $C(p,t) = Post(p,t) - Pre(p,t)$. The restriction of $\boldsymbol{C}$ (Pre, $Post$, resp.) to P_x and T_y, with $x, y \in \{c, d\}$, is denoted $\boldsymbol{C}_{xy}$ (Pre_{xy}, $Post_{xy}$, resp.).

A *marking* is a function that assigns to each discrete place a non-negative number of tokens, and to each continuous place a fluid volume. Therefore, $M : \ P_c \rightarrow \mathbb{R}_0^+$, $P_d \rightarrow \mathbb{N}$. The marking of place p_i is denoted M_i, while the value of the marking at time τ is denoted $M(\tau)$. The restriction of M to P_d and P_c are denoted with M^d and M^c, resp.

A *system* $\langle N, M(\tau_0)\rangle$ is an FOHPN N with an initial marking $M(\tau_0)$.

Net dynamics: The enabling of a discrete transition depends on the marking of all its input places, both discrete and continuous. More precisely, a discrete transition t is *enabled* at M if for all $p_i \in {}^\bullet t$, $M_i \geq Pre(p_i, t)$, where ${}^\bullet t$ denotes the preset of transition t. The *enabling degree* of t at M is equal to $enab(M,t) = \max\{k \in \mathbb{N} \mid \ M \geq k \cdot Pre(\cdot,)\}$.

If t is *infinite-server semantics*, we associate to it a number of clocks that is equal to $enab(M,t)$. Each clock is initialized to a value that is equal to the time delay of t, if t is deterministic, or to a random value depending on the distribution function of t, if t is stochastic. If a discrete transition is *k-server semantics*, then the number of clocks that are associated to t is equal to $\min\{k, enab(M,t)\}$. The values of clocks associated to t decrease linearly with time, and t fires when the value of one of its clocks is null (if $\bar{k}$ clocks reach simultaneously a null value, then t fires $\bar{k}$ times). Note that here we are considering *enabling memory policy*, not *total memory policy*. This means that if a transition enabling degree is reduced by the firing of a different transition, then the disabled clocks have no memory of this in future enabling [1], [4].

If a discrete transition t_j fires k times at time τ, then its firing at $M(\tau^-)$ yields a new marking $M(\tau)$ such that $M^c(\tau) = M^c(\tau^-) + \boldsymbol{C}_{cd}\boldsymbol{\sigma}$, and $M^d(\tau) = M^d(\tau^-) + \boldsymbol{C}_{dd}\boldsymbol{\sigma}$, where $\boldsymbol{\sigma} = k \cdot \boldsymbol{e}_j$ is the *firing count vector* associated to the firing of transition t_j k times.

Every continuous transition t_j is associated with an instantaneous firing speed (IFS) $v_j(\tau)$. For all τ it should be $V_j' \leq v_j(\tau) \leq V_j$, and the IFS of each continuous transition is piecewise constant between events.

A continuous transition is enabled only by the marking of its input discrete places. The marking of its input continuous places, however, is used to distinguish between *strong* and *weakly enabled*: if all input continuous places of t_j have a not null marking, then t_j is called *strongly enabled*, else t_j is called *weakly enabled* [1].

We can write the equation which governs the evolution in time of the marking of a place $p_i \in P_c$ as $\dot{m}_i(\tau) = \sum_{t_j \in T_c} C(p_i, t_j) v_j(\tau)$ where $\boldsymbol{v}(\tau) = [v_1(\tau), \ldots, v_{n_c}(\tau)]^T$ is the IFS vector at time τ.

The enabling state of a continuous transition t_j defines its admissible IFS v_j. If t_j is not enabled then $v_j = 0$. If t_j is strongly enabled, then it may fire with any firing speed $v_j \in [V_j', V_j]$. If t_j is weakly enabled, then it may fire with any firing speed $v_j \in [V_j', \overline{V}_j]$, where $\overline{V}_j \leq V_j$ since t_j cannot remove more fluid from any empty input continuous place $\overline{p}$ than the quantity entered in $\overline{p}$ by other transitions. Linear inequalities can be used to characterize the set of *all* admissible firing speed vectors $\mathcal{S}$. Each vector $\boldsymbol{v} \in \mathcal{S}$ represents a particular mode of operation of the system described by the net, and among all possible modes of operation, the system operator may choose the best, i.e., the one that maximize a given performance index $J(\boldsymbol{v})$ [2].

We say that a *macro–event* (ME) occurs when: (a) a discrete transition fires, thus changing the discrete marking and enabling/disabling a continuous transition; (b) a continuous place becomes empty, thus changing the enabling state of a continuous transition from strong to weak; (c) a continuous place, whose marking is increasing (decreasing), reaches a flow level that increases (decreases) the enabling degree of discrete transitions.

Let τ_k and τ_{k+1} be the occurrence times of two consecutive ME as defined above; we assume that within the interval of time $[\tau_k, \tau_{k+1})$, denoted as a *macro–period* (MP), the IFS vector is constant and we denote it $\boldsymbol{v}(\tau_k)$. Then, the continuous behavior of an FOHPN for $\tau \in [\tau_k, \tau_{k+1})$ is described by $M^c(\tau) = M^c(\tau_k) + \boldsymbol{C}_{cc}\boldsymbol{v}(\tau_k)(\tau - \tau_k)$, $M^d(\tau) = M^d(\tau_k)$.

***Example** 2.1:* Consider the net system in Fig. 1(a). Place p_1 is a continuous place, while all other places are discrete. Continuous transitions t_1, t_2 have MFS $V_1 = 1$, $V_2 = 2$ and null mfs. Deterministic timed discrete transitions t_3, t_5 have timing delays 2 and 1.5, resp. Exponential stochastic discrete transitions t_4, t_6 have average firing rates are $\lambda_4 = 2$ and $\lambda_6 = 1.5$.

[1] We are using an enabling policy for continuous transitions slightly different from the one proposed by David and Alla [4]. See [2] for a detailed discussion.

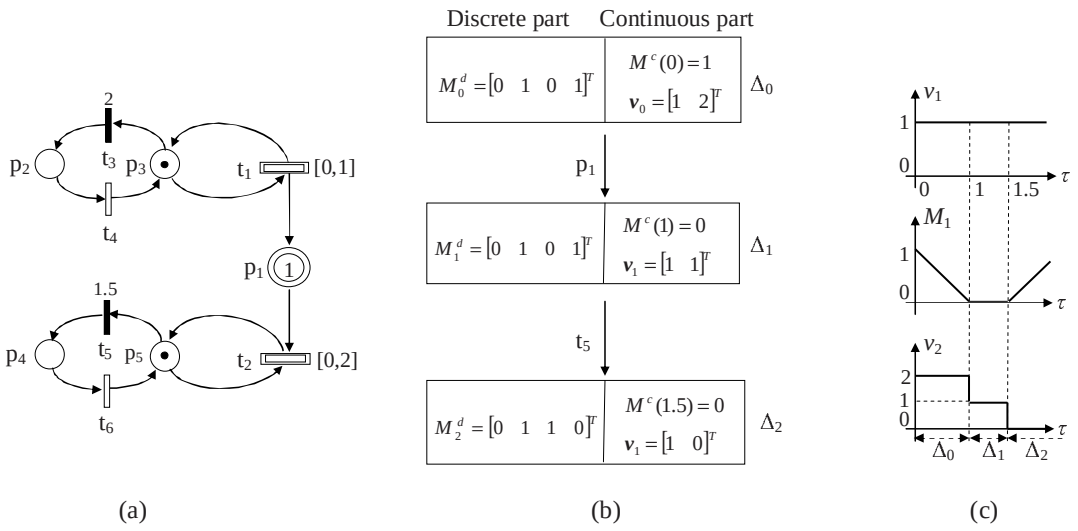

Fig. 1. (a) An FOHPN, (b) its evolution graph, and (c) its evolution in time.

The continuous transitions represent two unreliable machines; parts produced by the first machine (t_1) are put in a buffer (p_1) before being processed by the second machine (t_2). The discrete subnet represents the failure model of the machines. When p_3 is marked, t_1 is enabled, i.e. the first machine is operational; when p_2 is marked, transition t_1 is not enabled, i.e. the first machine is down. A similar interpretation applies to the second machine.

Assume that we want to maximize the production rates of machines. In such a case, during any MP continuous transitions fire at their highest speed. This means that we want to maximize $J(\boldsymbol{v}) = v_1 + v_2$ under the constraints $v_1 \leq V_1$, $v_2 \leq V_2$, and — when p_1 is empty — $v_2 \leq v_1$.

The resulting *evolution graph* and the time evolution of M_1, v_1 and v_2 are shown in Fig. 1(b) and (c). During the first MP (of length 1) both continuous transitions are strongly enabled and fire at their MFS. After one time unit, p_1 gets empty, thus t_2 becomes weakly enabled fires at the same speed of t_1. At time $\tau = 1.5$, transition t_5 fires, disabling t_2. ■

III. THE HYPENS TOOL

HYPENS has been developed in Matlab (Version 7.1). It is composed of 4 main files. The first two files, *make_HPN.m* and *enter_HPN.m* create the net to be simulated: the former requires input data from the workspace while the latter is a guided procedure.

The file *simulator_HPN.m* computes the timing evolution of the net that is summarized in an array of cells called *Evol*. Based on this array, the file *analysis_HPN.m* computes useful statistics and plots the simulation results.

Function make_HPN *[Pre, Post, M0, vel, v, D, s, $alpha$] = make_HPN (Pre$_{cc}$, Pre$_{cd}$, Pre$_{dc}$, Pre$_{dd}$, Post$_{cc}$, Post$_{cd}$, Post$_{dd}$, M0$_c$, M0$_d$, vel, v, D, s, $alpha$).*

Input arguments.

– *Matrices* Pre_{cc}, Pre_{cd}, Pre_{dc}, Pre_{dd}, $Post_{cc}$, $Post_{cd}$, $Post_{dd}$;

– The *initial marking* M0$_c$ and M0$_d$ of continuous/discrete places.

– Matrix $vel \in (\mathbb{R}_0^+)^{q_c \times 2}$ specifies, for each continuous transition, the mfs and the MFS.

– Vector $v \in \mathbb{N}^{1 \times q_d}$ specifies the timing structure of each discrete transition. The entries of this vector may take the following values: 1 - *deterministic*; 2 - *exponential distribution*; 3 - *uniform distribution*; 4 - *Poisson distribution*; 5 - *Rayleigh distribution*; 6 - *Weibull distribution*; etc.

– Matrix $D \in (\mathbb{R}_0^+)^{q_d \times 3}$ associates to each discrete transition a row vector of length 3. If the transition is deterministic, the first element of the row is equal to the time delay of transition. If the transition is stochastic, the elements of the row specify the parameters of the corresponding distribution function (up to three, given the available distribution functions).

– Vector $s \in \mathbb{N}^{1 \times q_d}$ keeps track of the number of servers associated to discrete transitions. The entries take any value in $\mathbb{N}$: 0 if the corresponding transition has *infinite servers*; $k > 0$ if the corresponding transition has k *servers*.

– Vector $alpha$ specifies the conflict resolution policy among discrete transitions.

- If $alpha \in \mathbb{N}^{1 \times q_d}$ two cases are possible. If all its entries are zero, conflict resolution is solved by priorities that depend on the indices of transitions (the smallest the index, the highest the priority). Otherwise, all its entries are greater than zero and specify the weight of the corresponding transition, i.e., if T_e is the set of enabled transitions, the probability of firing transition $t \in T_e$ is $\pi(t) = alpha(t) / \left(\sum_{t' \in T_e} alpha(t')\right)$.
- If $alpha \in \mathbb{N}^{2 \times q_d}$ the first row specifies the weights associated to transitions (as in the previous case) while the second row specifies the priorities associated to transitions. During simulation, when a conflict arises, priority are first considered; in the case of equal priority, weights are used to solve the conflict. See [4] for details.

Output arguments. (They are nothing else than input data, appropriately rewritten to be passed to function *simulator_HPN*).

– Matrices Pre and $Post$ are defined as:

$$Pre = \begin{bmatrix} Pre_{cc} & NaN & Pre_{cd} \\ NaN & NaN & NaN \\ Pre_{dc} & NaN & Pre_{dd} \end{bmatrix}, \quad Post = \begin{bmatrix} Post_{cc} & NaN & Post_{cd} \\ NaN & NaN & NaN \\ Post_{dc} & NaN & Post_{dd} \end{bmatrix}$$

where a row and a column of NaN (not a number) have been introduced to better visualize the continuous and/or discrete sub-matrices.

– The initial marking is denoted as $M0$ and is defined as a column vector.

– All other output data are identical to the input data.

Function enter_HPN: *[Pre, Post, M0, vel, v, D, s, alpha] = enter_HPN*.

This function creates the net following a guided procedure. The parameters are identical to those defined for the previous function *make_HPN.m*.

Function simulator_HPN: *Evol = simulator_HPN(Pre, Post, M, vel, v, D, s,* $alpha$*, time_stop, simulation_type, J)*.

Input arguments. They coincide with the output argument of the previous interface functions, plus three additional parameters.

– *time_stop* is equal to the time length of simulation.

– *simulation_type* $\in \{2, 1, 0\}$ specifies the simulation mode. Mode 0: no intermediate result is shown but only array $Evol$ is created to be later analyzed *analysis_HPN*. Modes 1 and 2 generate on screen the *evolution graph*: in the fist case the simulation proceeds without interruptions until *time_stop* is reached; in the second case the simulation is carried out step-by-step.

– $J \in \mathbb{R}^{1 \times q_c}$ is a row vector that associates to each continuous transition a weight: $J \cdot \boldsymbol{v}$ is the linear cost function that should be maximized at each MP to compute the IFS vector $\boldsymbol{v}$. This optimization problem is solved using the subroutine *glpkmex.m* of Matlab.

Output arguments. The output is an array of cells called *Evol*, with the following entries (here K is the number of ME that occur during the simulation run).

– *Type* $\in \{1, 2, 3\}$: 1 (2, 3) if the net is continuous (discrete, hybrid).

– *M_Evol* $\in (\mathbb{R}_0^+)^{n \times (K+1)}$ keeps track of the marking of the net during all the evolution: an n–dimensional column vector is associated to the initial time instant and to all the time instants

in which a different ME occurs, each one representing the corresponding value of the marking at that time instant.

– *IFS_Evol* $\in (\mathbb{R}_0^+)^{q_c \times (K+1)}$ keeps track of the IFS vectors during all the evolution. In particular, a q_c–dimensional column vector is associated to the initial configuration and to the end of each ME.

– *P_macro* and *Event_macro_Evol* are $(K+1)$-dimensional row vectors and keep track of the ME caused by continuous places. If the generic r-th ME is due to continuous place p_j, then the (r+1)–th entry of *P_macro* is equal to j; if it is due to the firing of a discrete transition, then the (r+1)–th entry of *P_macro* is equal to NaN. The entries of *Event_macro_Evol* may take values in $\{0, 1, -1, NaN\}$: 0 means that the corresponding continuous place gets empty; 1 means that the the continuous place enables a new discrete transition; -1 means that the continuous place disables a discrete transition. If the generic r-th ME is due to the firing of a discrete transition, then the (r+1)-th entry of *Event_macro_Evol* is equal to NaN as well. The first entries of both *P_macro* and *Event_macro_Evol* are always equal to NaN.

– *firing_transition* is a $(K+1)$–dimensional row vector that keeps track of the discrete transitions that have fired during all the evolution. If the r-th ME is caused by the firing of discrete transition t_k, then the $(r+1)$–th entry of *firing_transition* is equal to k; if it is caused by a continuous place, then the $(r+1)$–th entry of *firing_transition* is equal to 0. Note that the first entry of *firing_transition* is always equal to NaN.

– *timer_macro_event* is a $(K+1)$–dimensional row vector that keeps into memory the length of ME. The first entry is always equal to NaN.

– τ is equal to the total time of simulation.

– *Q_Evol* is a $((K+1) \times q_d)$–dimension array of cells, whose generic $(r+1, j)$-th entry specifies the clocks of transition t_j at the end of the r-th MP.

– $P_c\_P_d\_T_c\_T_d \in \mathbb{N}^4$ is a 4-dimensional row vector equal to $[n_c \;\; n_d \;\; q_c \;\; q_d]$.

Function analysis_HPN: *[P_ave, P_max, Pd_ave_t, IFS_ave, Td_ave, Pd_freq, Md_freq] = analysis_HPN (Evol, static_plot, graph, marking_plot, Td_firing_plot, up_marking_plot, Pd_prob_plot, Pd_ave_t_plot, IFS_plot, up_IFS_plot, IFS_ave_plot, Td_freq_plot).*

This function computes useful statistics and plots the results of the simulation run contained

in *Evol*.

Input arguments. – *statistic_plot* $\in \{0, 1\}$: if 1 two histograms are created showing for each place, the maximum and the average marking during the simulation.

– *graph* $\in \{0, 1\}$: when set to 1 the evolution graph is printed on screen.

– *marking_plot*: is a vector used to plot the marking evolution of selected places in separate figures. As an example, if we set *marking_plot*$= [x\ y\ z]$, the marking evolution of places p_x, p_y and p_z is plotted. If *marking_plot*$= [-1]$, then the marking evolution of all places is plotted.

– *Td_firing_plot* $\in \{0, 1\}$: when set to 1 a graph is created showing the time instants at which discrete transitions have fired.

– *up_marking_plot* is a vector used to plot the marking evolution of selected places in a single figure. The syntax is the same as that of *marking_plot*.

– *Pd_prob_plot* $\in \{0, 1\}$: 1 means that as many plots as the number of discrete places will be visualized, each one representing the frequency of having a given number of tokens during the simulation run.

– *Pd_ave_t_plot* is a vector used to plot the average marking in discrete places with respect to time. A different figure is associated to each place, and the syntax to select places is the same as that of *marking_plot*.

– *IFS_plot* (resp., *up_IFS_plot*): is a vector used to plot the IFS of selected transitions in separate figures (resp., in a single figure). The syntax is the same as that of *marking_plot* and *up_marking_plot*.

– *IFS_ave_plot* $\in \{0, 1\}$: if 1 an histogram is created showing the average firing speed of continuous transitions.

– *Td_freq_plot* $\in \{0, 1\}$: if 1 an histogram is created showing the firing frequency of discrete transitions during the simulation run.

Output arguments.

– *P_ave* (*P_max*) $\in \mathbb{R}^{1\times n}$: each entry is equal to the average (maximum) marking of the corresponding place during the simulation run.

– *Pd_ave_t* $\in \mathbb{R}^{n_d\times K}$: column k specifies the average marking of discrete places from time $\tau_0 = 0$ to time τ_k when the k-th ME occurs.

– *IFS_ave* $\in \mathbb{R}^{1\times q_c}$: each entry is equal to the average IFS of the corresponding continuous transition.

– *Td_ave* $\in \mathbb{R}^{1\times q_d}$: each entry is equal to the average enabling time of the corresponding discrete transition.

– *Pd_freq* $\in \mathbb{R}^{n_d\times(z+1)}$ specifies for each discrete place the frequency of a given number of tokens during the simulation run. Here z is the maximum number of tokens in any discrete place during the simulation run.

– *Md_freq* $\in \mathbb{R}^{(n_d+1)\times\tilde{K}}$ where $\tilde{K}$ denotes the number of different discrete markings that are reached during the simulation run. Each column of *Md_freq* contains one of such markings and its corresponding frequency as a last entry.

IV. A NUMERICAL EXAMPLE

Let us consider again the FOHPN system in Fig. 1(a) with performance index to be maximized $J = v_1 + v_2$ and a simulation time of 20 time units. The most significant results of the simulation run are shown in Fig. 2: the marking evolution of continuous place p_1, of discrete places p_3 and p_5, the IFS v_1 and v_2, and the occurrence of ME. In particular, the bottom right figure specifies if the ME is due to a discrete transition (the index i of transition is shown in the y axis) or to a continuous place (the value of the y axis is equal to zero).

The main results of function analysis_HPN.m are:

$$P\_ave = \begin{array}{c} \begin{array}{ccccc} p_1 & p_2 & p_3 & p_4 & p_5 \end{array} \\ \left[\begin{array}{ccccc} 0.127 & 0.600 & 0.400 & 0.475 & 0.525 \end{array}\right] \end{array},$$

$$P\_max = \begin{array}{c} \begin{array}{ccccc} p_1 & p_2 & p_3 & p_4 & p_5 \end{array} \\ \left[\begin{array}{ccccc} 1.321 & 1 & 1 & 1 & 1 \end{array}\right] \end{array},$$

$$Pd\_freq = \begin{array}{c} \begin{array}{cc} & \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{c} p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} & \left[\begin{array}{cc} 0.400 & 0.600 \\ 0.600 & 0.400 \\ 0.525 & 0.475 \\ 0.475 & 0.525 \end{array}\right] \end{array} \end{array},$$

$$IFS\_ave = \begin{array}{c} \begin{array}{cc} t_1 & t_2 \end{array} \\ \left[\begin{array}{cc} 0.400 & 0.450 \end{array}\right] \end{array},$$

$$Td\_ave = \begin{array}{c} \begin{array}{cccc} t_3 & t_4 & t_5 & t_6 \end{array} \\ \left[\begin{array}{cccc} 0.200 & 0.150 & 0.350 & 0.300 \end{array}\right] \end{array}.$$

The evolution graph created by *simulator_HPN*, and matrices *Md_ave_t*, *Md_freq* are not reported here, but can be downloaded from the web site [8].

In this paper we only present a very simple numerical example but we proved the efficiency of HYPENS via real dimensional cases examples in [6]. Here we considered both timed nets, modeling a family of queueing networks, and a hybrid net modeling a job shop system with finite capacity buffers and unreliable multi-class machines.

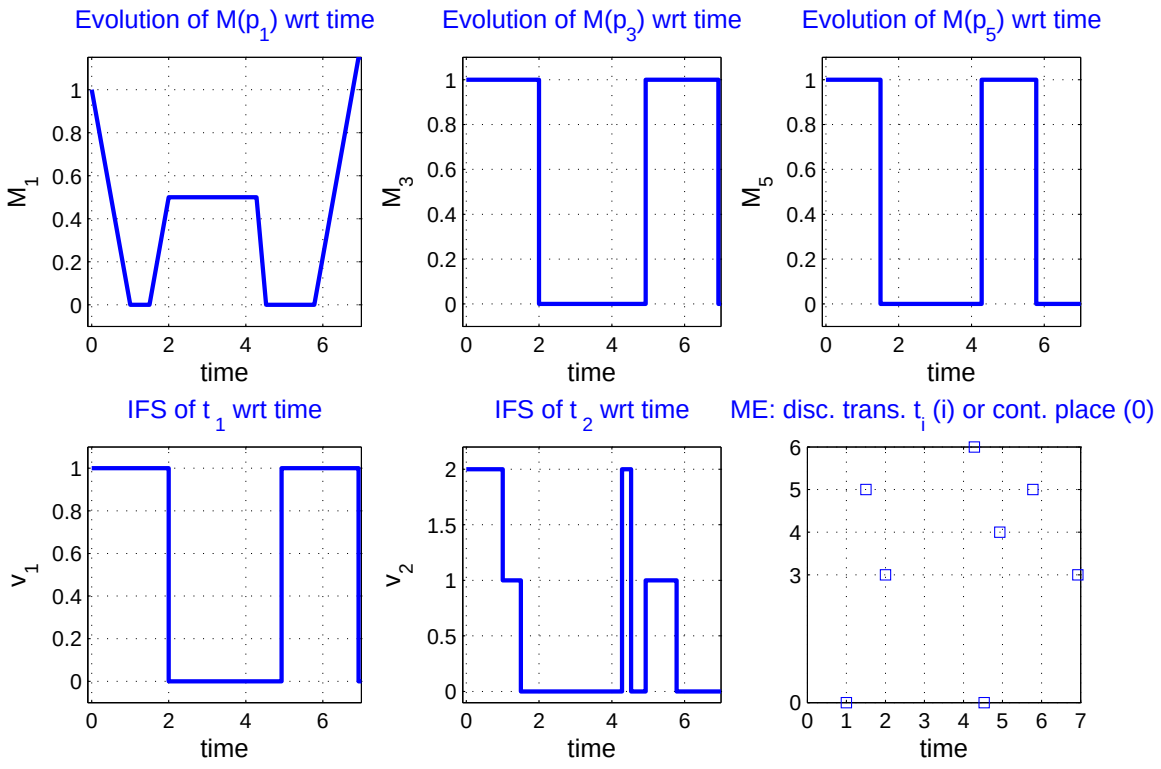

Fig. 2. Simulations carried out on the FOHPN system in Fig. 1(a) using HYPENS.

V. Conclusions

We presented a Matlab tool for the simulation and analysis of timed discrete, continuous and hybrid PNs. Very general distribution functions can be considered, and the analysis of simulation may be efficiently carried out both graphically and numerically. The optimization of the net with respect to a given performance index may be easily carried out, and different conflict resolution policies may be considered. We plan to extend the tool adding the primitive ”total memory policy”.

References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley, 1995.

[2] F. Balduzzi, A. Giua, and C. Seatzu. Modelling and simulation of manufacturing systems with first-order hybrid Petri nets. *Int. J. of Production Research*, 39(2):255–282, 2001.

[3] 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.

[4] R. David and H. Alla. *Discrete, Continuous and Hybrid Petri Nets*. Springer, 2004.

[5] A. Di Febbraro, A. Giua, G. Menga, and (Eds.). Special issue on Hybrid Petri nets. *Discrete Event Dynamic Systems*, 2001.

[6] A. Giua, C. Seatzu, and F. Sessego. Simulation and analysis of hybrid Petri nets using the Matlab tool HYPENS. In *Proc. 2008 IEEE Int. Conf. on Systems, Man and Cybernetics*, Singapore, 2008, submitted.

[7] http://sourceforge.net/projects/hisim.

[8] http://www.diee.unica.it/automatica/hypens.

[9] http://www.lag.ensieg.inpg.fr/sirphyco.

[10] M. Silva and L. Recalde. On fluidification of Petri net models: from discrete to hybrid and continuous models. *Annual Reviews in Control*, 28(2):253–266, 2004.