Extending the Modeling and Analysis Capabilities of Continuous Petri Nets by Flexible Nets

Jorge Júlvez and Stephen G. Oliver

Abstract— **Continuous Petri nets aim to avoid the state explosion problem of classical discrete Petri nets by relaxing the integrality constraint of the firing of transitions. Although the resulting formalism can successfully model a number of features of dynamic systems, its use in complex real systems can be hindered by the limited number of possible dynamics and by the difficulty in accommodating uncertain parameters. The modeling formalism of Flexible Nets can overcome these difficulties by significantly extending the modeling and analysis possibilities of continuous Petri nets. The modeling capabilities of Flexible Nets will be presented together with their analysis possibilities in both the transient and steady state.**

I. INTRODUCTION

Petri nets (PNs) [1] is a popular modeling formalism for discrete event systems which has been used in a wide range of application domains such as business process modeling, systems biology, and game theory [2], [3], [4]. As with any other formalism for discrete event systems, PNs suffer from the state explosion problem and, hence, many analysis methods are not efficient. A way to avoid this difficulty is to relax the integrality constraint of the firing of transitions and deal with Continuous Petri Nets (CPNs) [5]. Although such a relaxation might involve the loss of some properties [6], more efficient analysis and control methods can be developed [7], [8], [9]. Despite their modeling power, some common features of real concurrent systems (such as uncertainties, complex firing semantics and shared resources) remain difficult to be modeled with CPNs.

Flexible Nets (FNs) [10] is a recent modeling formalism inspired by PNs which can overcome some of the modeling and analysis limitations of CPNs. An FN is composed of two nets, an *event net* which models the state changes produced by the processes of the system, and an *intensity net* which captures how the state determines the speeds of the processes. Both, the event and the intensity net are tripartite graphs whose vertices are places, transitions and (event and intensity) handlers. Places and transitions are connected through handlers which can be associated with sets of linear inequalities.

This work was supported by the Spanish Ministry of Science and Innovation [ref. DAMOCLES-PID2020-113969RB-I00] to JJ, by the Biotechnology & Biological Sciences Research Council (UK) grant no. BB/N02348X/1 to SGO, as part of the IBiotech Program, and by the Industrial Biotechnology Catalyst (Innovate UK, BBSRC, EPSRC) to support the translation, development and commercialisation of innovative Industrial Biotechnology processes.

J. Júlvez is with the Department of Computer Science and Systems Engineering, University of Zaragoza, Zaragoza, Spain julvez@unizar.es

S.G. Oliver is with the Cambridge Systems Biology Centre and the Department of Biochemistry, University of Cambridge, Cambridge, UK sgo24@cam.ac.uk

The introduction of handlers and inequalities enhances the modeling power of FNs and opens the door to new analysis possibilities. The use of inequalities, for instance to model uncertain parameters, results in a nondeterministic model with a range of different potential trajectories. Such a model can be studied both in its transient and steady state by means of mathematical constraints on its state variables [10], [11].

The remainder of the paper is organized as follows: Sections II and III introduce CPNs and FNs respectively. Section IV discusses how some basic features of CPNs can be modeled by FNs. Section V shows how FNs can extend the modeling and analysis capabilities of CPNs. Section VI concludes the paper.

II. CONTINUOUS PETRI NETS

In the following, the reader is assumed to be familiar with Petri nets (see [1], [5] for a gentle introduction to discrete and continuous Petri nets).

A Petri net (PN) is a tuple $\mathcal{N} = \{P, T, \mathbf{Pre}, \mathbf{Post}\}$ where $P = \{p_1, p_2, ..., p_n\}$ and $T = \{t_1, t_2, ..., t_m\}$ are disjoint and finite sets of places and transitions, and $\mathbf{Pre}$ and $\mathbf{Post}$ are $|P|\times|T|$ sized, natural valued, incidence matrices. The *preset* and *postset* of a node $X \in P \cup T$ are denoted by ${}^\bullet X$ and $X^\bullet$ respectively.

In a Continuous PN (CPN), every place p_i has a nonnegative real marking (or number of tokens) $m[p_i]\in\mathbb{R}_{\geq 0}$, the initial marking is denoted $m_0[p_i]$. A CPN system is a tuple $\{\mathcal{N}, m_0\}$ where $\mathcal{N}$ is the net and m_0 is the initial marking. In a CPN, the enabling degree of transition t_j at marking m is defined as $enab(t_j, m)= \min\limits_{p_i\in{}^\bullet t_j} \left\{ \dfrac{m[p_i]}{\mathbf{Pre}[p_i, t_j]} \right\}$.

The state equation of a CPN, $m=m_0+\mathbf{C}\cdot\sigma$ where $\mathbf{C}$ is the incidence matrix, summarizes the system evolution. The derivative of this equation with respect to time is $\dot{m} = \mathbf{C}\cdot\dot{\sigma}$ where $\dot{\sigma} = \mathbf{f}$ is the speed (or flow) of transitions. Different semantics exist to define $\mathbf{f}$, the two most important being infinite server and finite server semantics [5], [12], [13].

In order to define the flow under a firing semantics, a vector $l \in \mathbb{R}_{>0}^{|T|}$ which associates a positive value with each transition is used. Under finite server semantics, the speed of a transition t_i at marking m such that $enab(t_i, m)>0$ is equal to $l[t_i]$ (if $enab(t_i, m)=0$ several possibilities have been proposed [5], [13]). Under infinite server semantics, the speed of a transition t_i at marking m is equal to $l[t_i]\cdot enab(t_i, m)$ [12].

For instance, the speed of t_1 of the CPN in Fig.1(a) at marking $m=(2\ 3\ 0)$ is $l[t_1]$ under finite server semantics and $l[t_1]\cdot enab(t_1, m)=l[t_1]\cdot 1.5$ under infinite server semantics.

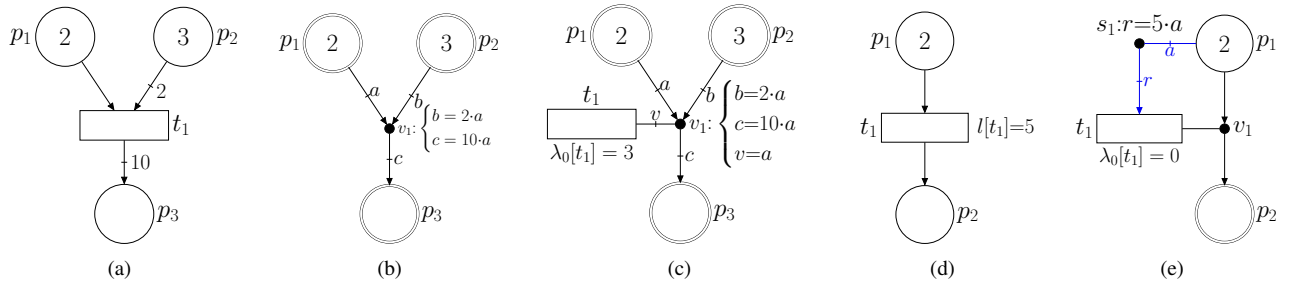


Fig. 1: (a) A simple CPN; (b) Event net modeling the untimed behavior of the CPN in (a); (c) FN with a constant firing speed of t_1; (d) CPN under infinite server semantics; (e) FN mimicking the CPN in (d).

III. FLEXIBLE NETS

Flexible Nets (FNs) are composed of an event net and an intensity net: the event net determines how the system processes change the state of the system, and the intensity net establishes the speeds of such processes as a function of the state. This section outlines the definition of FNs, see [10] for a detailed definition.

Definition 1 (Event net): An event net is a tuple $\mathcal{N}_V{=}(P, T, V, E_V, A, B)$ where (P, T, V, E_V) is a tripartite graph determining the net structure, and (A, B) are matrices determining the potential evolutions of the marking.

The vertices of the event net are P, T and V, where P is a set of $|P|$ places, T is a set of $|T|$ transitions and V is a set of $|V|$ event handlers. Similarly to Petri nets, places are depicted as circles and model the types of components in the system. As in CPNs, every place $p_i \in P$ has a nonnegative real number of tokens, $m[p_i] \in \mathbb{R}_{\geq 0}$, called marking (the initial marking is denoted $m_0[p_i]$). Transitions are depicted as rectangles and model the system processes. Event handlers are depicted as dots and model the different ways in which the transitions can change the marking.

The vertices of the event net are connected by the edges in E_V. Each pair of vertices can be connected by at most one edge. The set E_V is partitioned into two sets E_V^P and E_V^T: The edges in E_V^P are directed and are referred as *event arcs*, every $e \in E_V^P$ is either an arc $e{=}(p_i, v_k)$ from a place p_i to a handler v_k, or an arc $e{=}(v_k, p_i)$ from a handler v_k to a place p_i. Every $e \in E_V^T$ is an edge $e{=}\{t_j, v_k\}$ connecting a transition t_j and a handler v_k.

Each event handler is associated with a set of inequalities which establishes how the marking can change, the inequalities of all handlers are captured in matrices A and B [10]. For instance, the handler v_1 in the event net in Fig.1(b) has two equalities, $b{=}2{\cdot}a$ and $c{=}10{\cdot}a$, where a, b and c are the labels of the arcs. These equalities imply that for every token consumed from p_1 (label a), 2 tokens will be consumed from p_2 (label b) and 10 tokens will be produced in p_3 (label c).

For the sake of simplicity, the inequalities of handlers will be omitted if all the labels are equal, e.g. the omission of the inequalities of v_1 would imply $a{=}b{=}c$.

Definition 2 (Intensity net): An intensity net is a tuple $\mathcal{N}_S{=}(P, T, S, E_S, C, D, \mathcal{P}, \varphi)$ where (P, T, S, E_S) is a tripartite graph determining the net structure, and (C, D) are matrices determining the potential intensity changes produced by the marking. $\mathcal{P}$ is a set of *partitions*, and φ is a function that associates *guards* with *intensity arcs*.

The vertices of the net are P, T and S, where P is a set of $|P|$ places, T is a set of $|T|$ transitions and S is a set of $|S|$ intensity handlers. Places and transitions model the same system features as in the event net. The intensity handlers are depicted as dots and model the different ways in which the tokens can generate intensities, or speeds, in the transitions. Every transitions t_j has a nonnegative real intensity, or speed, $\lambda[t_j] \in \mathbb{R}_{\geq 0}$ which can depend on the marking (the default speed, denoted $\lambda_0[t_j]$, is constant and does not depend on the marking).

The vertices of the intensity net are connected by the edges in E_S. Each pair of vertices can be connected by at most one edge. The set E_S is partitioned into two sets E_S^T and E_S^P: The edges in E_S^T are directed and are referred as *intensity arcs*, every $e \in E_S^T$ is either an arc $e{=}(t_j, s_l)$ from a transition t_j to a handler s_l, or an arc $e{=}(s_l, t_j)$ from a handler s_l to a transition t_j Every $e \in E_S^P$ is an edge $e{=}\{p_i, s_l\}$ connecting a place p_i and a handler s_l. Notice that although both, event handlers and intensity handlers, are represented as dots, they can be distinguished by the arcs and edges that connect them to transitions and places, e.g. event handles are connected to transitions by edges while intensity handlers are connected to transitions by arcs. For clarity, intensity edges and arcs will be drawn in blue.

The tokens in places can be used by the intensity handlers to produce intensities. A token is *active* if it is being used by an intensity handler, otherwise it is *idle*. While idle tokens are associated with places, active tokens are associated with edges. The places whose tokens are forced to be active are drawn as single circles, and the places whose tokens can be idle are drawn as double circles, see Fig.1(e).

An intensity handler determines how much intensity is produced in its arcs as a function of the number of active tokens in its edges. The intensity produced in an incoming arc (s_l, t_j) to t_j(an outgoing arc (t_j, s_l) from t_j) is added to(subtracted from) the default intensity $\lambda_0[t_j]$. For instance, the equation $r{=}5{\cdot}a$ of the intensity handler s_1 in Fig.1(e) implies that the intensity produced in arc (s_1, t_1) is equal to 5 times the number of active tokens in edge $\{p_1, s_1\}$. Given

that the tokens of p_1 are forced to be active (p_1 is drawn as a single circle) and (s_1, t_1) is an incoming to t_1, then the speed of t_1 at marking m=(2 0) is $\lambda[t_1]=\lambda_0[t_1]+5{\cdot}m[p_1]=10$. As in event nets, if the inequalities of an intensity handler are omitted, it is assumed that all the labels of its arcs and edges are made equal. Moreover, similarly to null initial markings, null default speeds can be omitted.

In order to enhance the modeling power of FNs, several sets of inequalities can be associated with each intensity handler. Only one of these sets is *active* at a given instant, and the active set depends on the marking of the net. The conditions for the activation of the sets are called *guards*. For example, if $r=\begin{cases}4 \text{ if } a \geq 1\\ a \text{ otherwise}\end{cases}$ is associated with s_1 in Fig.1(e), then the speed of t_1 is 4 if the marking of p_1 is higher than 1, and equal to the marking of p_1 otherwise, i.e. a guarded FN can be seen as a hybrid system. The sets of inequalities and guards of the intensity handlers are encoded in the net elements C, D, $\mathcal{P}$ and φ [10].

IV. FROM CONTINUOUS PETRI NETS TO FLEXIBLE NETS

The introduction of handlers with inequalities in FNs increases significantly their modeling power as compared to CPNs. In particular, uncertainties and nondeterminism can be easily modeled. The event net in Fig.1(b) mimics the untimed behavior of the CPN in Fig.1(a) (the equalities of v_1 just capture the relative quantities in which tokens are consumed and produced). Assume that $c{=}10{\cdot}a$ of v_1 is replaced by $9{\cdot}a{\leq}c{\leq}11{\cdot}a$. Then, for every token consumed from p_1, a real and nondeterministic quantity of tokens in the interval $[9, 11]$ is produced in p_3. Thus, in contrast to CPNs, the marking change produced by a handler is allowed to be nondeterministic.

With respect to Fig.1(b), the net in Fig.1(c) introduces a transition, t_1, and the equality $v{=}a$ in v_1. This equality implies that every action of t_1 that is executed will consume one token from p_1, 2 tokens from p_2, and will produce 10 tokens in p_3. The speed of t_1, i.e. the rate at which actions in t_1 are executed, is constant an equal to $\lambda_0[t_1]{=}3$. Instead of this equality for $\lambda_0[t_1]$, inequalities can be used, for instance $2.5{\leq}\lambda_0[t_1]{\leq}3.5$ would imply that the speed of t_1 is uncertain (and can vary over time) but constrained to the interval $[2.5, 3.5]$. The FN in Fig.1(e) mimics the timed behavior of the CPN in Fig.1(d) under infinite server semantics. As in the event handler, the equality in s_1 can be replaced by inequalities to model uncertain parameters, e.g. $4{\cdot}a{\leq}r{\leq}6{\cdot}a$ would mean that the speed of t_1 is uncertain but constrained to the interval $[4{\cdot}m[p_1], 6{\cdot}m[p_1]]$.

The use of inequalities in an FN results in a nondeterministic evolution of the model. In order to analyze the potential evolutions, sets of mathematical constraints that represent necessary reachability conditions in the transient [10] and steady state [11] can be developed. Such constraints together with an objective function can be used to build a programming problem whose solution represents state bounds for all potential evolutions.

V. MODELING AND ANALYSIS

This section exploits further the modeling capabilities of FNs. These capabilities are presented together with analysis possibilities of FNs by means of case studies.

A. Simultaneous enabling

The CPN in Fig. 2(a) has two self-loops, one consists of the pair (p_1, t_1) and the other of the pair (p_1, t_2). Thus, the firing of t_1 (and t_2) consumes and produces tokens simultaneously in p_1. Given that the arcs of the self-loops have different weights, 1 and 2, the role of such arcs is both a) to constrain the speeds of the transitions and; b) to specify the number of tokens consumed and produced by the firing of transitions. In contrast, FNs explicitly distinguish between the marking change produced by firings and the way places control the speed of transitions.

The event net in Fig. 2(b) models the marking changes produced by the firing of transitions. Each transition is connected to an event handler, and each event handler is connected to places by arcs to specify consumption and production of tokens. Given that no explicit inequalities are associated with the handlers, the execution of one action of a transition, e.g. t_1, consumes one token from p_1 and p_3 and produces one token in p_2. Notice that the event net models the overall marking change produced by the firings and, hence, self-loops are not required. The absence of self-loops makes it apparent that the markings of p_1 and p_3 are equally affected by the transitions.

According to the definition of CPNs, the tokens of p_1 in Fig. 2(a) enable simultaneously both t_1 and t_2. In particular, the speed of t_1 is given by $l[t_1]{\cdot}min(m[p_1]/2, m[p_3])$ and the speed of t_2 by $l[t_2]{\cdot}min(m[p_1], m[p_2])$. This relation between marking and speeds can be modeled by the intensity net in Fig. 2(c) in which the inequalities associated with the intensity handler s_1 state that the speeds of the transitions are $\lambda[t_1]{=}k{\cdot}min(m[p_1]/2, m[p_3])$ and $\lambda[t_2]{=}min(m[p_2], m[p_3])$, where the parameter k is equal to $l[t_1]$ and it is assumed that $l[t_2]{=}1$. Let us assume that the initial marking is $m_0{=}[5\ 0\ 4]$. Figure 2(e) shows the steady state speed of t_1, $\overline{\lambda}[t_1]$, computed according to the constraints in [11] for different values of k when $\overline{\lambda}[t_1]$ is maximized. It must be noticed that $\overline{\lambda}[t_1]$ is not monotonous with respect to k and presents a discontinuity at $k{=}2$ (see [14]).

In contrast to CPNs, FNs offer the possibility of avoiding the simultaneous enabling of transitions by the marking of places. This possibility is exemplified by the intensity net in Fig. 2(d) which has two intensity handlers, s_1 is connected to t_1 and s_2 is connected to t_2. Place p_1 has two edges, each edge is connected to an intensity handler. This implies that each token in p_1 can be used either by s_1 or by s_2 but not by both simultaneously. The tokens that are active in the edge $\{p_1, s_1\}$ will synchronize with the tokens in $\{p_3, s_1\}$ to produce speed in t_1, and the tokens that active in $\{p_1, s_2\}$ will synchronize with the tokens in $\{p_2, s_2\}$ to produce speed in t_2. The number of tokens active in $\{p_1, s_2\}$ and $\{p_1, s_2\}$ is not initially set and will be obtained when the associated programming problem is solved. Figure 2(f)

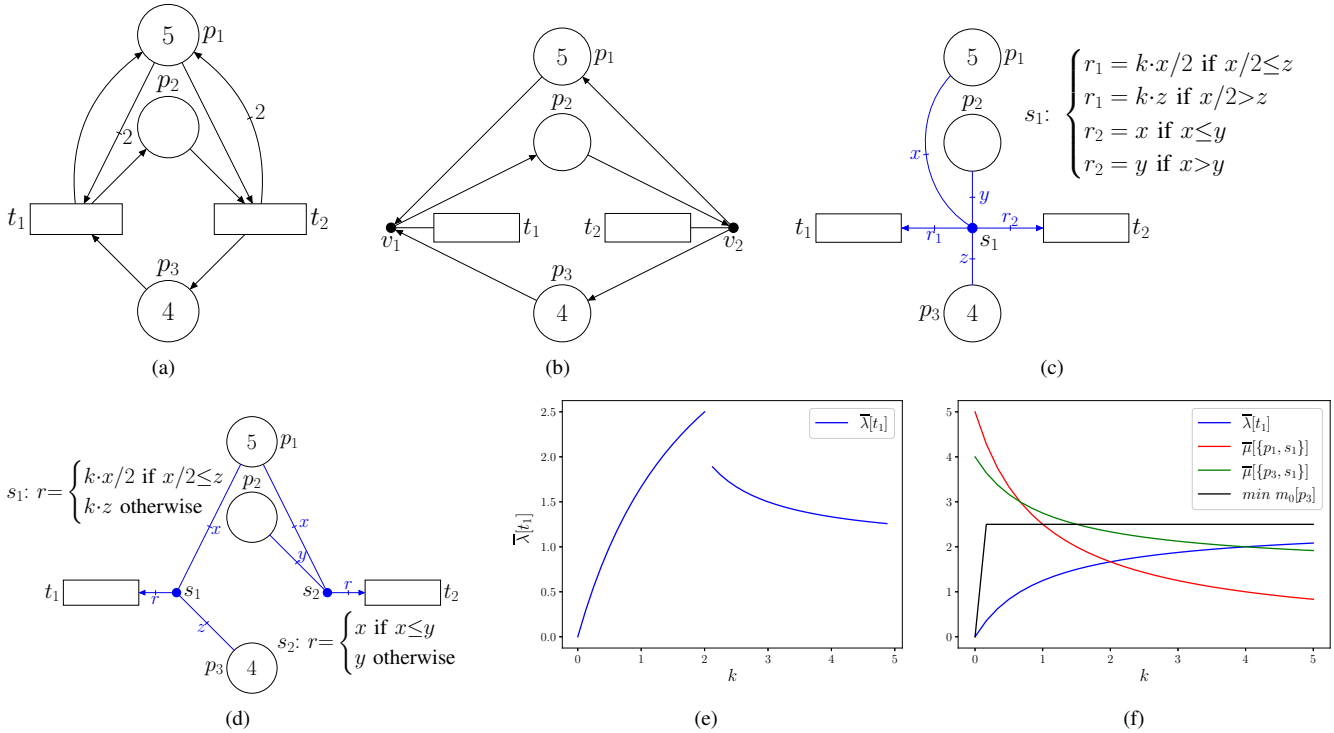


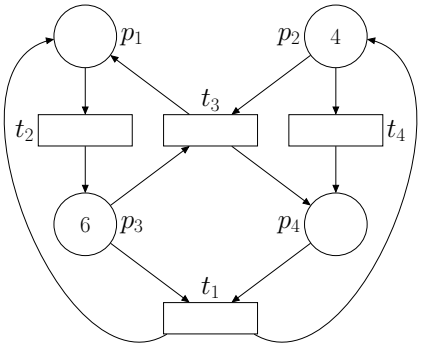
Fig. 2: (a) CPN with a self-loop place with different arc weights; (b) Event net modeling the marking changes produced by the firing of transitions; (c) Intensity net with simultaneous enabling; (d) Intensity net without simultaneous enabling; (e) Steady state speed of t_1 for different values of k under simultaneous enabling; (f) Steady state speed of t_1 for different values of k under non-simultaneous enabling.

shows the steady state speed of t_1, $\overline{\lambda}[t_1]$, for different values of k when $\overline{\lambda}[t_1]$ is maximized, it also shows the number of active tokens in the edges connected to s_1, $\overline{\mu}[\{p_1, s_1\}]$ and $\overline{\mu}[\{p_3, s_1\}]$. Notice that $\overline{\lambda}[t_1]$ is now continuous and monotonous with with respect to k. Moreover, from the plot it can be deduced that $m[p_3]$ never constrains the speed of t_1 ($\overline{\mu}[\{p_1, s_1\}]/2<\overline{\mu}[\{p_3, s_1\}]$ holds for any $k>0$) . In other words, there are unused tokens in p_3. FNs can be used to compute the minimum initial marking of p_3 that makes use of all the tokens in the steady state without decreasing the speed of t_1. This can be achieved by forcing such a speed as a constrain in the optimization problem and by minimizing $m_0[p_3]$. The resulting initial marking, as a function of k, is the trajectory $min\ m_0[p_3]$ in Fig. 2(f). It has been assumed that, both in Fig. 2(c) and Fig. 2(d), the tokens of all places are forced to be active.

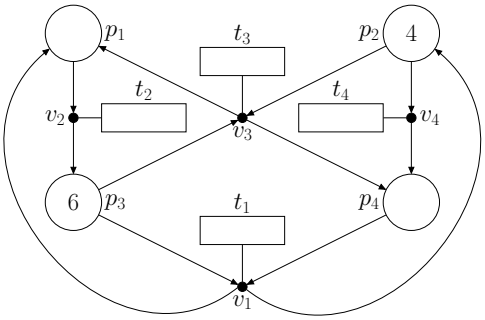
B. Allocation of resources

Let us consider the CPN in Fig. 3(a) with initial marking m_0=(0 4 6 0) and l=(1.0 1.0 1.5 1.0). The event net in Fig. 3(b) models the marking changes produced by the firing of transitions, and the intensity net in Fig. 3(c) models the dependence of the speeds of the transitions on the marking. Notice that t_2 is the only transition that is not a synchronization, i.e. it is enabled just by one place, p_1. Thus, the speed of t_2 is modeled by one intensity handler, s_2, which connects p_1 and t_2. As in the previous case study, see Subsection V-A, in order to model the simultaneous enabling of t_1, t_3 and t_4 by the tokens in p_2, p_3 and p_4, only one intensity handler, s_1, must be used. Such an intensity handler connects p_2, p_3 and p_4 to t_1, t_3 and t_4, and associates equations with its arcs and edges according to the vector l of the CPN, e.g. r_3=$1.5{\cdot}min(x, y)$ implies that the speed of t_3 will be equal to $1.5{\cdot}min(m[p_2], m[p_3])$ (recall that $l[t_3]$=1.5). In order to mimic exactly the behavior of the CPN the tokens of all places must be forced to be active.

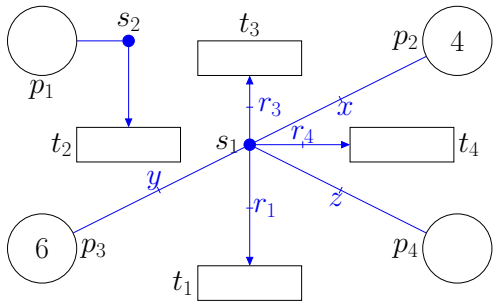
Two limitations of the intensity net in Fig. 3(c) are: 1) as in the previous case study, tokens are forced to simultaneously enable several transitions, which might not be realistic; 2) the use of min in the equations of s_1 requires the use of guards, e.g. r_1=$min(y, z)$ is equivalent to $r_1=\begin{cases} y \text{ if } y \leq z \\ z \text{ otherwise} \end{cases}$, and hence, the existence of binary variables in the optimization problems, which can result in a high computational complexity. In order to overcome such limitations, the intensity net in Fig. 3(d) can be used. Such a net has one intensity handler per transition and, hence, places p_2 and p_3 have several intensity edges implying that their tokens can enable any of the transitions to which they are connected but not more than one simultaneously. In this intensity net, the expressions of the handlers are equalities and tokens p_2, p_3 and p_4 are not forced to be active. Thus, the tokens of such places can either be inactive or active



(a)

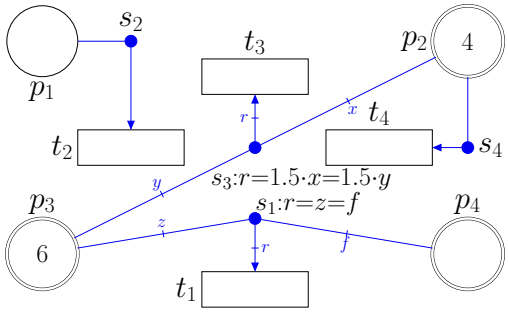


(b)

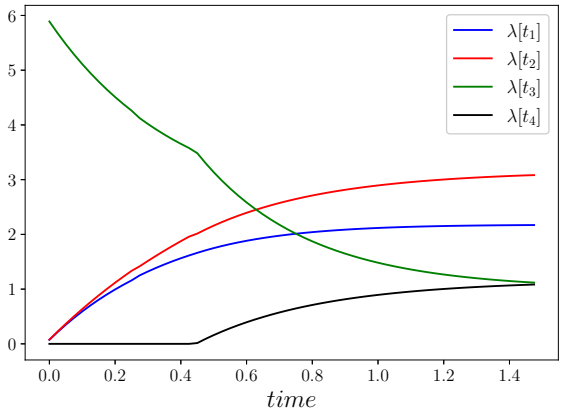


$s_1{:}r_1{=}min(y,z); r_3{=}1.5{\cdot}min(x,y); r_4{=}x$

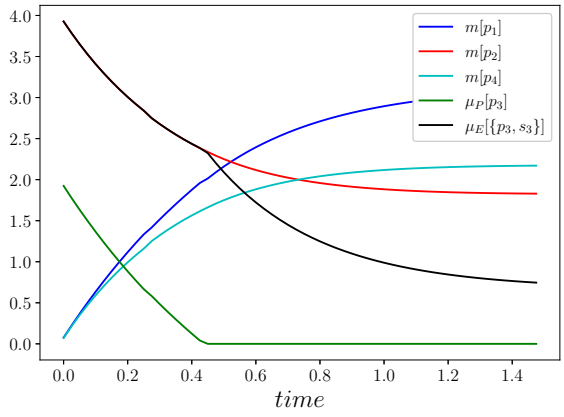
(c)



(d)



(e)



(f)

Fig. 3: (a) CPN with $l = (1.0\ 1.0\ 1.5\ 1.0)$; (b) Event net modeling the marking changes produced by the firing of transitions; (c) Intensity net mimicking the speeds of transitions with guards; (d) Alternative intensity net without guards; Time evolution of the intensities (e) and markings (f) of the FN composed of the event net in (b) and the intensity net in (d).

in any of their intensity edges. Notice that the equalities of the handlers forces an exact synchronization of tokens (or resources) to produce intensities in the transitions. The actual amounts of active and inactive tokens will be determined by the solution of the optimization problem, which is now linear, under consideration.

Assume that it is desired to maximize the average intensity of t_1, $\overline{\lambda}[t_1]$ over a period of 1.5 time units. In order to obtain a trajectory of the state variables over time for such an optimization problem, intermediate states can be considered [10]. Under this approach, the optimization period is sampled in intervals in which the final state at the end of one interval is forced to be equal to the initial state at the beginning of the next interval. Figures 3(e) show the intensity of the transitions over time when 60 intervals of length 0.025 time units are considered and the linear programming

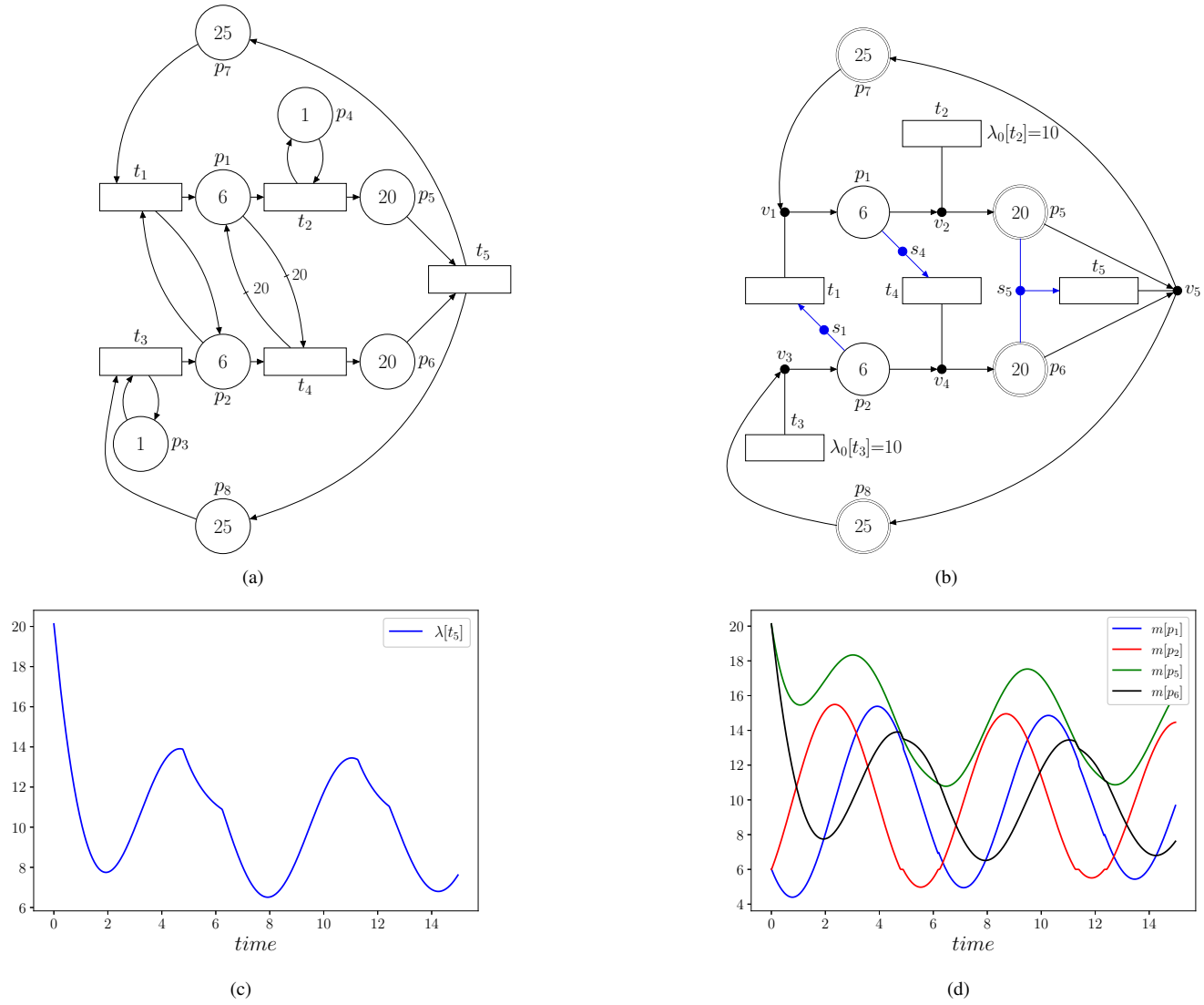


Fig. 4: (a) CPN showing an oscillatory behavior with $l = (1\ 10\ 10\ 20\ 1)$; (b) FN mimicking the behavior of the CPN without self-loops; (c) Time trajectory of $\overline{\lambda}[t_5]$ of the FN; (d) Time trajectory of the markings of p_1, p_2, p_5, and p_6.

problem resulting of maximizing $\overline{\lambda}[t_1]$ with the associated constraints [10] is solved.

Given that the initial marking of p_4 is 0, the initial speed of t_1, $\lambda[t_1]$, is necessarily 0. Thus, in order to maximize the objective function $\overline{\lambda}[t_1]$, tokens must be carried *as soon as possible* to p_4 to enable t_1. Notice that there are two ways to move tokens from p_2 to p_4: either use t_3 or t_4. Given that, in contrast to t_1, the intensity produced in t_3 by s_3 is 1.5 times the number of active tokens in the edges, it is faster to use t_3 than t_1 to move tokens to p_4 (notice, however, than in other to use t_3, tokens in p_3 are required). This way, $\lambda[t_4]$ is 0 during the first 0.48 time units, see Fig. 3(e), and all the tokens in p_2 are active in $\{p_2, s_3\}$ during this period. At time instant 0.48, $\lambda[t_4]$ becomes positive and $\lambda[t_3]$ exhibits a sharper decrease. Figure 3(f) shows the marking of p_1, p_2 and p_4 at each of the 60 steps, the number of idle tokens in p_3, $\mu_P[p_3]$, and the number of p_3 that are active in $\{p_3, s_3\}$, $\mu_E[\{p_3, s_3\}]$. Notice that during the first 0.48 time units there are idle tokens in p_3 and then all are active. The trajectories of both Fig. 3(e) and (f) tend to a steady state at which $\overline{\lambda}[t_1]$ is maximum.

C. Self-loops and firing semantics

Figure 4(a) shows a CPN with $l = (1\ 10\ 10\ 20\ 1)$ and self-loops in places p_1, p_2, p_3 and p_4 (in addition to self-loops, p_1 and p_2 also have t_2 and t_4 as output transitions respectively). Recall that in a CPN, arcs model consumption and production of tokens, and hence self-loops are required to model the fact that a transition speed depends on the marking of a place but, if the weights of the arcs are the same, its firing produces no change in the marking of such a place. The initial marking of p_3 and p_4, which will remain equal to 1 over time, is low enough to ensure that the speeds of t_3 and t_2 are always equal to $l[t_3]{\cdot}m[p_3]$=10 and

$l[t_2]{\cdot}m[p_4]{=}10$ respectively. Thus, the self-loops in p_3 and p_4 are used exclusively to model a constant speed, or finite server semantics, of transitions t_3 and t_2.

On the other hand, t_1 has two input places, p_2 and p_7, such that the marking of p_2 is always (for this net structure, initial marking and l) less than that of p_7. Hence, the speed of t_1 is always constrained by p_2 and is equal to $l[t_1]{\cdot}m[p_2]{=}m[p_2]$. Transition t_4 also has two input places, p_1 and p_2, with equal initial marking. In order to make the speed of t_4 equal to p_1, and hence independent of p_2, the arcs of the self-loop in p_1 are given a high value, e.g. 20, and $l[t_4]$ is given the same value, $l[t_4]{=}20$, this way it holds that the speed of t_4 is $l[t_4]{\cdot}m[p_1]/20{=}m[p_1]$.

Figure 4(b) shows an FN which mimics the behavior of the CPN. Given that in FNs, the marking changes and the production of intensities are modeled by two different nets, the event and the intensity net, no self-loops are required either to model constant speeds or to model the fact that the speed of a transition depends on a place whose tokens are not consumed by its firing. In the FN, $\lambda_0[t_2]$ and $\lambda_0[t_3]$ model constant speeds of t_2 and t_3, and s_1 and s_4 state that the speeds of t_1 and t_4 depend exclusively on p_2 and p_1 respectively, i.e. $\lambda[t_1]{=}m[p_2]$ and $\lambda[t_4]{=}m[p_1]$.

The synchronization in t_5 is modeled by s_5 which just establishes that the speed of t_5 is equal to the number of active tokens in $\{p_5, s_5\}$ which has to be equal to the number of active tokens in $\{p_6, s_5\}$ (notice that the tokens in p_5 and p_6 are not forced to be active). By using an objective function that maximizes $\lambda[t_5]$, it is ensured that at each time instant all the tokens of either p_5 or p_6 are active and equal to $\lambda[t_5]$. In this way, it is possible to model a synchronization without guards (and hence without binary variables), thus reducing significantly the CPU time required to obtain a time trajectory. The trajectories of $\lambda[t_5]$ and the marking of places over time are shown in Figs. 4(c) and (d) respectively. These trajectories are obtained by applying a model predictive control approach [10] in which the length of the control horizon was 0.025 time units.

The numerical results and trajectories in this section were obtained by the `fnyzer` tool [15] (executed on a desktop computer Intel i7, 2.00 GHz, 8 GiB, Ubuntu 20.04 LTS) which solves the programming problems [10] associated with the FNs. The CPU time to optimize each steady state in Subsection V-A was below $0.05s$ both with guards, intensity net in Fig. 2(c), and without guards, intensity net in Fig. 2(d). The CPU time to solve the overall optimization problem in Subsection V-B, intensity net Fig. 3(d), was 1.19 seconds. The CPU time to solve the optimization problem at each step of the model predictive control of the FN in SubsectionV-C, FN in Fig. 4(b), was 0.0744 seconds.

VI. CONCLUSIONS

FNs is a recent modeling formalism for dynamic systems which offers a number of modeling and analysis possibilities. An FN combines an event net, which captures the relationship between processes and marking changes, and an intensity net, which models the relationship between the marking and the speed of the processes. The vertices of these nets are transitions, places and handlers. Linear inequalities can be associated with handlers to account for the mentioned relationships. Handlers, together with the associated inequalities, can be exploited to model different server semantics, resource allocation strategies and uncertainties.

Nonlinear dynamics can also be modeled in FNs by means of guards. In a guarded net, an intensity handler can be associated with several sets of linear inequalities. The set of linear inequalities that rules the net dynamics at given time instant is determined by the current marking. Guards open the door to accommodate dynamics that are much more general than the min operator of CPNs.

The current analysis methods of FNs rely on the solution of programming problems that encode necessary reachability conditions together with an objective function. Such problems can be used to analyze both the transient and the steady state of the system. Among other possibilities, the use of appropriate objective functions allows simulation of the time evolution of a system with synchronizations just with linear inequalities, and the optimization of the allocation of resources in a system over time.

REFERENCES

[1] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Procs. of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[2] W. M. P. van der Aalst and C. Stahl, *Modeling Business Processes: A Petri Net-Oriented Approach*. Information Systems, Cambridge, MA: MIT Press, 2011.

[3] I. Koch, W. Reisig, and F. Schreiber, *Modeling in Systems Biology. The Petri Net Approach*. Springer-Verlag London, 2011.

[4] J. Clempner, "Modeling shortest path games with Petri nets: A Lyapunov based theory," *International Journal of Applied Mathematics and Computer Science*, vol. 16(3), pp. 387–397, 01 2006.

[5] R. David and H. Alla, *Discrete, Continuous and Hybrid Petri Nets*. Berlin: Springer, 2005. (2^{nd} edition, 2010).

[6] L. Recalde and M. Silva, "PN Fluidification revisited: Semantics and Steady state," in *ADPM 2000: 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems*, pp. 279–286, 2000.

[7] C. Mahulea, L. Recalde, and M. Silva, "Basic Server Semantics and Performance Monotonicity of Continuous Petri Nets," *Discrete Event Dynamic Systems*, vol. 19, p. 212, 06/2009 2009.

[8] J. A. Fraustro-Valdez, J. Ruiz-León, C. R. Vázquez, and A. Ramírez-Treviño, "Structural fault diagnosis in Timed Continuous Petri Nets," in *13th International Workshop on Discrete Event Systems, WODES 2016*, pp. 159–164, IEEE, 2016.

[9] M. Taleb, E. Leclercq, and D. Lefebvre, "Model Predictive Control for Discrete and Continuous Timed Petri Nets," *International Journal of Automation and Computing*, vol. 15, no. 1, pp. 25–38, 2018.

[10] J. Júlvez, D. Dikicioglu, and S. G. Oliver, "Handling variability and incompleteness of biological data by flexible nets: a case study for Wilson disease," *npj Systems Biology and Applications*, vol. 4, p. 7, 1 2018.

[11] J. Júlvez and S. G. Oliver, "Steady State Analysis of Flexible Nets," *IEEE Transactions on Automatic Control*, pp. 1–1, 2019.

[12] M. Silva and L. Recalde, "Petri nets and integrality relaxations: A view of continuous Petri net models," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 32, no. 4, pp. 314–327, 2002.

[13] F. Balduzzi, A. Giua, and G. Menga, "First-order hybrid Petri nets: a model for optimization and control," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 4, pp. 382–399, 2000.

[14] J. Júlvez, L. Recalde, and M. Silva, "Steady state performance evaluation of continuous mono-T-semiflow Petri nets," *Automatica*, vol. 41, pp. 605–616, May 2005.

[15] J. Júlvez and S. G. Oliver, "fnyzer: A Python Package for the Analysis of Flexible Nets," in *18th International Conference on Computational Methods in Systems Biology*, vol. 12314 of *Lecture Notes in Computer Science*, pp. 349–355, Springer, 2020.