# Petri nets for the control of discrete event systems

Alessandro Giua, Carla Seatzu

DIEE, University of Cagliari, Italy (email: {giua,seatzu}@diee.unica.it)
LSIS, Aix-Marseille University, France

**Abstract**

The interest in Petri nets has grown within the automatic control community in parallel with the development of the theory of discrete event systems. In this article our goal is that of giving a flavor of the features that make Petri nets a good model for discrete event systems and of pointing out the main areas where Petri nets have offered the most significant contributions.

Published as:
A. Giua, C. Seatzu, "Petri nets for the control of discrete event systems," *Software & Systems Modeling*, Vol. 14, No. 2, pp. 693701, 2015. DOI: 10.1007/s10270-014-0425-1.

# 1 Introduction

The object of the study of traditional control theory have been *time-driven systems*, i.e., systems of continuous and synchronous discrete variables, modeled by differential or difference equations. However, as the scope of control theory is being extended into the domains of manufacturing, robotics, computer and communication networks, and so on, there is an increasing need for different models, capable of describing systems that evolve in accordance with the abrupt occurrence, at possibly unknown irregular intervals, of physical events. Such systems, whose states have logical or symbolic, rather than numerical, values that change in response to events which may also be described in nonnumerical terms, are called *discrete event systems* (DESs) and the corresponding models are called *discrete event models* (Cassandras and Lafortune, 2008).

These systems require control and coordination to ensure the orderly flow of events. As controlled (or potentially controllable) dynamic systems, discrete event systems qualify as a proper subject for control theory. Hence a fundamental issue arises: we need classes of formal models that are capable of capturing the essential features of discrete, asynchronous and possibly nondeterministic systems and that are endowed with efficient mathematical tools for analysis and control.

We claim that Petri nets (PNs) are a powerful discrete event model and, in fact, the interest for this model has grown, within the automatic control community, in parallel with the development of the theory of discrete event systems (Holloway *et al.*, 1997; Cassandras and Lafortune, 2008; Seatzu *et al.*, 2012). In this article the goal is not that of providing a comprehensive survey of the research in this area, but rather that of giving a flavor of the features that make Petri nets a good model for systems theory.

Let us first point out what are the two main challenges in the application of system theory approaches to discrete event systems.

- A first problem is due the *absence of reference models*. Unlike classical control theory where there are well accepted models, such input-output or state variables models that can be used in different contexts (analysis, control, etc.), in the DES domain we still find a series of different modeling formalisms, each one used to solve a series of particular problems. Examples of these modeling formalisms include finite state automata, formal languages, Petri nets, max-plus algebra, predicate algebra, semimarkovian processes, queuing networks, and so on.
- A second problem, that has so far been the main obstacle to the transfer of theoretical results to real word applications, is the so called *state space explosion* that originates from the combinatorial nature of DESs. Consider, as an example, a system composed by $k$ modules each one having a state space of cardinality $n$: the cardinality of the state space of the overall system will be $N \leq n^k$ and the upper bound is strict in the sense that in many cases it is attainable. This exponential growth of the state space size with the number of composed systems makes all those approaches that are based on an exhaustive exploration of the state space of little practical interest in many applications.

The success of Petri nets as a discrete event model can be explained observing that they provide an answer (albeit partial) to both problems.

## 2 The Petri net family

Petri nets can be considered as a reference model because they represent a general paradigm that gives rise to a large family of powerful models (Murata, 1989; David and Alla, 2005). *Place/transition nets* can be used to describe logical systems — possibly with an infinite state space — providing several primitives for modeling sequentiality, concurrency, rendez-vous, choice, resource allocation, etc. *Labeled nets* are formal languages generators that can also be used to study problems related to observability and identification: the class of Petri net languages are (analogously to context-free languages) a superset of regular languages and a subset of context-sensitive languages but the introduction of inhibitor arcs extends the modeling power of Petri nets to that of Turing machines. Models of *timed nets* have also been defined: if the timing structure is deterministic we get nets that can describe, among others, max-plus algebra models, while deterministic nets generalize semi-markovian processes and queueing networks. Finally, more recently *continuous* and *hybrid Petri nets* combining both event-driven and timed-driven dynamics have also be defined. Although some of these models require their own analysis techniques that are not applicable to all the family, there exist other approaches such as those based on reachability, state equation, incidence matrix analysis that are applicable to all models, thus providing a common unifying framework.

## 3 Coping with the state space explosion

Petri nets provide several features to alleviate the obstacle created by the state space explosion.

The modular synthesis of complex models is done through operators — such as the *synchronous product* also called *concurrent composition operator* — that work on the net structure and not on the reachability space (Peterson, 1981; Giua, 2012). This is an enormous advantage, because the net structure typically grows linearly with the number of composed modules, as opposed to the state space that grows exponentially.

**Example 3.1** *Consider a production line composed by a series of $k+1$ workstations, denoted $W_0, W_1, \ldots, W_k$ interconnected though $k$ buffers of capacity $n$, denoted $B_1, B_2, \ldots, B_k$ as shown in Fig. 1.(a). The first workstation takes parts from a buffer of infinite capacity (not modeled) and the last workstation puts produced parts in a buffer of infinite capacity (also not modeled).*

*Each workstation is initially in the idle state; when loaded with a part it goes in the working state and after being unloaded goes back to the idle state: the model of the generic workstation $W_i$ is shown in Fig. 1.(b). Each buffer has initially $n$ empty slots; when a new part arrives one slot is occupied, while when a part leaves one slot is freed: the model of the generic buffer $B_i$ is also shown in Fig. 1.(b).*

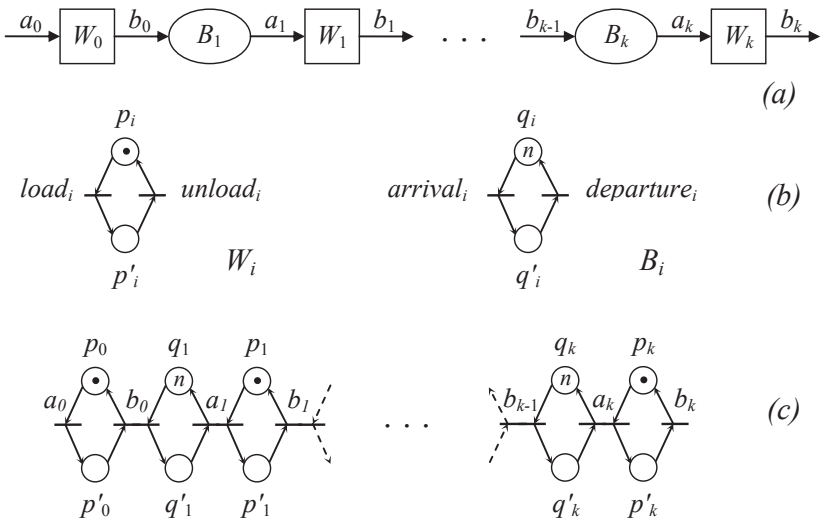


Figure 1: Modular synthesis of a production line in Example 3.1.

*In the production line, due to its topology, there is a synchronization between workstations and buffers. In fact, workstation $W_i$ is loaded with parts taken from buffer $B_i$ (for $i = 1, 2, \ldots, k$): this means that event $load_i$ coincides with event $departure_i$ and we denote this synchronized event $a_i$. Dually, when workstation $W_i$ is unloaded the processed part is put in buffer $B_{i+1}$ (for $i = 0, 1, \ldots, k-1$): this means that event $unload_i$ coincides with event $arrival_{i+1}$ and we denote this synchronized event $b_i$. The overall model of the production line can thus be represented by the net in Fig. 1.(c), obtained by the modules in Fig. 1.(b) through the concurrent composition operator after suitable relabeling of the transitions.*

*Two comments concerning the model of the overall system are in order.*

- *This net has $2(2k+1)$ places and $2k$ transitions hence its structure grows linearly with the number $k$ of composed modules. On the contrary, its state space has cardinality $2^{k+1}(n+1)^k$, being 2 (resp., $n+1$) the cardinality of each machine (resp., buffer) module.*
- *In the overall net it is still possible to distinguish the structure of each single module. This is due to a special feature of Petri nets: locality of states (described by the token distribution in places) and events (each transition modifies the token count of a subset of places).* ■

Some PN analysis techniques, such as those based on the reachability graph, require the exhaustive enumeration of the net state space, thus they do not fully exploit the representational advantage of this model. However, there exist other approaches, such as those based on *structural analysis* (Colom and Silva, 1990; Silva *et al.*, 1992; Cabasino *et al.*, 2012), that take into account the net structure. This has made possible the development of efficient techniques for reachability, liveness and deadlock analysis, state estimation, observability and diagnosis. It must be said, however, that these techniques are usually only applicable to particular classes of nets.

To give a flavor of the structural approaches, we discuss a classic result concerning reachability analysis. Given a net $N$ with initial marking $\boldsymbol{m}_0$, we denote the set of its reachable markings $R(N, \boldsymbol{m}_0)$. There exist several algebraic approximations of this set and here we recall the one based on invariant analysis.

**Example 3.2** *Consider the net in Fig. 1.(c) that models the production line discussed in Example 3.1. This net belongs to a special class called* marked graphs *and the following two results hold:*

- *the number of tokens in each elementary directed cycle of the net remains constant for all reachable markings and the corresponding law is called a* marking invariant *of the net);*
- *if no elementary directed cycle of the net is initially empty, then a marking that satisfies all invariants is reachable.*

*In this net there exist $k+1$ elementary directed cycles of the form $p_i a_i p'_i b_i p_i$ (for $i = 0, 1, \ldots, k$). The initial marking shown in the figure assigns one token to each cycle, and we can thus write the marking invariant laws:*

$$m(p_i) + m(p'_i) = m_0(p_i) + m_0(p'_i) = 1.$$

*There also exist $k$ elementary directed cycles of the form $q_i b_i q'_i a_{i+1} q_i$ (for $i = 1, 2, \ldots, k$). The initial marking shown assigns $n$ tokens to each cycle, and we can thus write the marking invariant laws:*

$$m(q_i) + m(q'_i) = m_0(q_i) + m_0(q'_i) = n.$$

*Hence we can write*

$$R(N, \boldsymbol{m}_0) = \{\boldsymbol{m} \in \mathbb{N}^{2(2k+1)} \mid \begin{array}{l} m(p_i) + m(p'_i) = 1 \ (i = 0, 1, \ldots, k), \\ m(q_i) + m(q'_i) = n \ (i = 1, 2, \ldots, k)\} \end{array}$$

*thus characterizing the set of reachable markings as the integer solutions of a system of linear equations, as opposed to enumerating it.* ■

# 4 Comparison with automata

We would also like to briefly comment on the relationship between Petri nets and finite state automata, another model that is often used in the DES domain (Seatzu *et al.*, 2012). Firstly, it is true that similarly to Petri nets automata can be seen as a family of formalisms that encompass logic (finite state and Büchi automata), performance (semimarkov processes, Markov chains) and hybrid models (timed and hybrid automata).

However automata have a smaller modeling power with respect to Petri nets: not only they can only describe finite state systems but they also lack explicit primitives to model behavioral features such as concurrency and rendez-vous. Furthermore, they require the explicit enumeration of the state space and lack computationally efficient algorithms for analysis and synthesis.

This gives credit to our belief that the study of automata — that being a very simple and intuitive model is an integral part of the introductory courses on discrete event systems — should always be complemented with the presentation of Petri nets.

## 5 Petri nets for DESs

In the last two decades an increasing number of researchers from the automatic control community have devoted their effort to the study of Petri nets: for a recent survey see Seatzu *et al.* (2012). Major contributions have appeared in the following areas of discrete event systems.

- *Control*: this is the core problem in systems theory and consists in applying suitable control inputs to a plant to ensure its behavior satisfies a given specification. For Petri nets a typical solution is that of designing a supervisor that disables the firing of some transitions as a feedback law of the plant's observed behavior or state: the controlled system (plant and supervisor) is also called closed-loop system (see Section 6).
- *Marking estimation*: the problem is that of determining efficient ways of reconstructing the state or the firing sequence of a net based on observed events occurrence and/or on partial marking observation (see Section 7).
- *Fault diagnosis*: given a net modeling a system possibly affected by a fault, under the assumption that the occurrence of a fault (modeled by an unobservable transition) cannot be directly observed the problem is that of detecting and identifying the fault from the observed behavior. Among the different approaches presented in this area are we mention the work of Prock (1991), Wu and Hadjicostis (2005), Miyagi and Riascos (2010), Basile *et al.* (2009), Benveniste *et al.* (2003), Dotoli *et al.* (2009), Genc and Lafortune (2007), Cabasino *et al.* (2010), Cabasino *et al.* (2011).
- *Deadlock*: a deadlock represents an anomalous state from which no further evolution is possible. This is an issue that appears in many automation problems and appropriate strategies should be adopted in order to prevent it. We refer to the works of Viswanadham *et al.* (1990), Ezpeleta *et al.* (1995), Chu and Xie (1997), Park and Reveliotis (2000), Iordache *et al.* (2002), Li and Zhou (2004), Reveliotis (2007) and to the recent contribution by Cordone *et al.* (2013).
- *Identification*: this problem consists in determining a Petri net system starting from examples/counterexamples of its language, or from the structure of its reachability (or coverability) graph. See the work of Hiraishi (1992), Badouel and Darondeau (1998), Sreenivas (2002), Bourdeaud'huy and Yim (2004), Cabasino *et al.* (2007).

In the rest of the article we will briefly discuss some successful approaches in the area of control and state estimation.

# 6 Control

The most interesting and original approach to the control of discrete event systems, that has directly or indirectly shaped much of the research in this area, is *Supervisory Control Theory* (SCT), originated by the work of P.J. Ramadge and W.M. Wonham (Ramadge and Wonham, 1989). According to the paradigm of SCT, a discrete event system $G$ is a language generator whose behavior, i.e., language, is denoted $L(G)$. Given a legal language $K$, the basic control problem is to design a supervisor that restricts the closed loop behavior of the plant to $K \cap L(G)$, disabling *controllable* events; the events whose occurrence cannot be disabled are called *uncontrollable*. It is also usually required that the closed loop system satisfies additional qualitative specifications, such as absence of blocking, reversibility, etc. Since Petri nets can be seen as language generators, it is also possible to use them as discrete event models for SCT; in this case it is assumed that some transitions, that we call controllable, can be disabled by an external agent (Holloway *et al.*, 1997; Cassandras and Lafortune, 2008; Giua, 2012).

A similar approach can also be taken when considering the state evolution of a discrete event system, rather than the traces of events it generates. This approach, that we call *state-based*, is particularly attractive when Petri nets are used to represent the plant and was used by several authors, as reviewed in (Holloway *et al.*, 1997). Let us consider a Petri net system $\langle N, \boldsymbol{m}_0 \rangle$ with $m$ places, whose set of reachable markings is $R(N, \boldsymbol{m}_0)$. Assume we are given a set of legal markings $\mathcal{M} \subseteq \mathbb{N}^m$: the basic control problem consisting in designing a supervisor that restricts the reachability set of plant in closed loop to $\mathcal{M} \cap R(N, \boldsymbol{m}_0)$, while satisfying some qualitative properties of interest.

Of particular interest are those Petri net state-based control problems where the set of legal markings $\mathcal{M}$ is expressed by one — or more — linear inequality constraints called Generalized Mutual Exclusion Constraints (GMEC) (Giua *et al.*, 1992). In this case we write $\mathcal{M}(\vec{w}, k) = \{\boldsymbol{m} \in \mathbb{N}^m \mid \vec{w}^T \boldsymbol{m} \leq k\}$ to denote that $\mathcal{M}$ is expressed by the GMEC $(\vec{w}, k)$ with $\vec{w} \in \mathbb{Z}^m, k \in \mathbb{Z}$. Problems of this kind have been considered by several authors and this special structure of the legal set has the advantage that the supervisor for this class of problems takes the form of a place, called *monitor*, which has arcs going to and coming from some transitions of the plant net. The plant and the controller are both described by a net thus providing a PN model of the closed-loop system for further analysis. Moreover the synthesis is not computationally demanding since it involves only a matrix multiplication. See Iordache and Antsaklis (2005) for a recent survey.

**Example 6.1** *Consider the net in Fig. 2.(a) — ignoring the dashed place $p_m$ and its arcs — representing the parallel execution of $k$ processes. Initially all processes are idle (place $p_1$ marked with $k$ tokens). From the idle state a process can pass to the reading state (place $p_2$ marked) and then to the writing state (place $p_3$ marked), before returning idle.*

*To ensure data integrity we require that at most one process can be writing at a time: this can be expressed by the GMEC $m(p_3) \leq 1$. The control structure that can enforce this constraint is the monitor place $p_m$ shown in in Fig. 2.(a).*

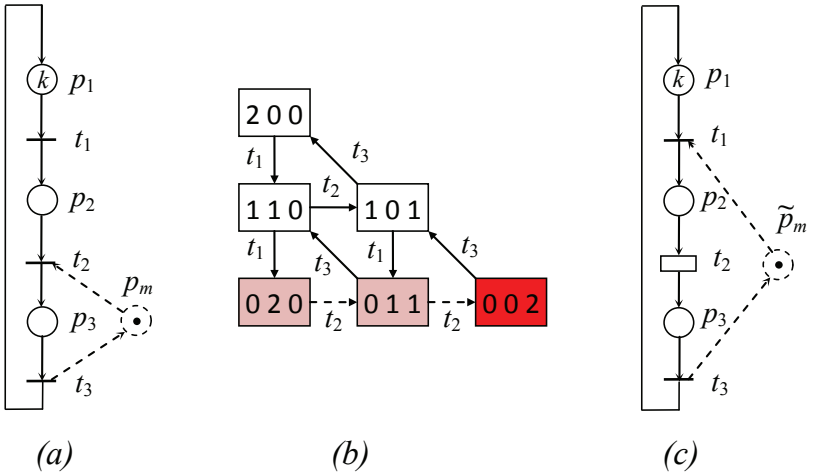


Figure 2: Enforcing a GMEC on a net: (a) plant and monitor assuming all transitions are controllable; (b) reachability graph of the plant for $k = 2$; (c) admissible monitor when transition $t_2$ is not controllable.

*A few comments concerning this control structure.*

- *The monitor is just a new place to be added to the net describing the uncontrolled system (that we call plant) but no additional transition is required.*
- *In Fig. 2.(a) we can recognize: a) the plant: the net without the monitor $p_m$ and its dashed arcs; b) the supervisor: the monitor $p_m$ and its dashed arcs; c) the closed-loop system: the complete net in figure.*
- *Although we do not detail the monitor design procedure — that can be found in (Giua* et al.*, 1992) — we point out that it does not require exploring the net state space. In fact, the monitor does not even depend on parameter $k$.*
- *In the closed-loop net, the monitor determines the marking invariant law $m(p_3)+m(p_m) = 1$ which of course implies the GMEC $m(p_3) \leq 1$.*

*To better understand how the monitor works, consider the reachability graph of the plant (assuming $k = 2$) shown in Fig. 2.(b). The only reachable marking that does not satisfy the GMEC is $[0\ 0\ 2]^T$: in the plant this marking is reachable by the sequence*

$$\begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} [t_1 t_1 t_2\rangle \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} [t_2\rangle \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}.$$

*However, in the closed-loop net after the firing of $t_1t_1t_2$ place $p_m$ is empty and transition $t_2$ is disabled by the supervisor.*

*Supervisory control theory assumes that some transitions may be uncontrollable, in the sense that a control agent may not be able to disable them. Suppose that in the current example transition $t_2$ is uncontrollable: in this case the monitor $p_m$ we have designed is not an admissible supervisor because it disables the uncontrollable transition $t_2$ that is enabled in plant at marking is* $[0\ 1\ 1]^T$. *In this case we need to prevent the plant to reach not only the set of* forbidden markings*, i.e., those markings that do not satisfy the GMEC, but also the set of* weakly forbidden markings*, i.e., those markings that satisfy the GMEC but from which a forbidden marking is reachable by firing a sequence of uncontrollable transitions. The weakly forbidden markings are the light-filled boxes in Fig. 2.(b).*

*Moody* et al. *(1996) and Moody and Antsaklis (1998) have presented an elegant structural algorithm to transform a non-admissible monitor into a more restrictive admissible one. In the case at hand, applying this procedure we get the monitor $\tilde{p}_m$ shown in Fig. 2.(c) where the uncontrollable transition $t_2$ has been represented by an empty box. The new monitor enforces the stronger GMEC $m(p_2) + m(p_3) \leq 1$, exactly preventing the plant from reaching all forbidden and weakly forbidden markings, and thus is an optimal, i.e., minimal restrictive, solution to the given control problem. As a final remark, we mention that when the net contains uncontrollable transitions an optimal solution may not be enforced by a monitor as shown by Giua* et al. *(1992): in this case the procedure of Moody and Antsaklis determines a sub-optimal supervisor.* ■

The use of Petri nets for the control of discrete event systems is still an active research area and we believe that it will continue to remain central during the next decade. In fact, there exist many interesting Petri net analysis tools — as an example, partial order techniques such as *unfolding* — whose applicability to control is still largely unexplored.

# 7 Marking estimation and observability

The problem of estimating the state of a dynamic system has attracted the attention of several researchers in the DES community (Ramadge, 1986; Caines *et al.*, 1988; Kumar *et al.*, 1993; Ozveren and Willsky, 1990; Giua and Seatzu, 2002; Cabasino *et al.*, 2010; Cabasino *et al.*, 2011). It consists in reconstructing the current *state* values of a dynamical system from the knowledge of the current and past values of its external measurable *outputs* and *inputs*. If such a problem admits a solution, the system is said to be *observable*.

We remark that several choices are possible for inputs and outputs.

- In the case of input/output automata — such as Mealy and Moore machines, synchronized Petri nets — the *inputs* are the *events* in the input alphabet. In the case of *autonomous* systems (finite state automata, labeled Petri nest) we obviously have no inputs.
- Possible choice of *outputs* are: *event labels* in the case of Mealy automata; *state labels* in Moore automata; measurements coming from transitions or places sensors on Petri nets.

In the rest of this section we will consider Petri nets with no inputs and where the firing

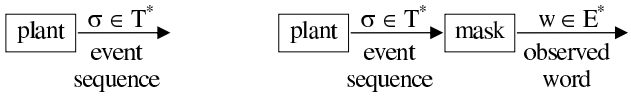


Figure 3: Total observation (left) and partial observation (right).

of transitions can be observed — possibly though an observation mask, as discussed in the following.

A main difference exists between state estimation in time-driven (i.e., continuous) systems and state estimation in DES. While in former case the problem consists in determining at each time instant $t$ an estimate $\chi(t)$ of the actual state $x(t)$, in the DES framework it consists in determining a set of states $\mathcal{C}(w)$ *consistent* with a given observation $w$, i.e., the possible values of the system's state after $w$ has been observed.

In DES we also distinguish between two main approaches to the state estimation problem:

- *Total observation*: transition firings are directly observable (i.e., the net is free-labeled) but the initial state is unknown (Giua and Seatzu, 2002).
- *Partial observation*: transition firings are observed though a mask (i.e, the net is arbitrarily-labeled) but the initial state is usually known (Cabasino *et al.*, 2010; Cabasino *et al.*, 2011).

In the Petri net framework, the above two cases are schematized in Fig. 3 where $T$ denotes the set of transitions of the Petri net plant, $E$ is an alphabet of symbols associated to transitions via a labeling function $L : T \to E \cup \{\varepsilon\}$, and $\varepsilon$ is the empty string. In the case of partial observation we mention that: (a) nondeterminism may arise from the fact that transitions may either be labeled with the empty string, or share the same label in $E$ with other transitions that are concurrently enabled: (b) one may also be interested in reconstructing the set $\mathcal{S}(w)$ of event sequences consistent with observation $w$ (event estimation).

Several are the motivations behind state estimation. The most natural one in a systems theory setting is the need to implement a *state-feedback* or *event-feedback* control as sketched in Fig. 4. In the left-hand side scheme specifications are given in terms of legal states, so the controller computes a control pattern $\gamma(\mathcal{C}(w))$ that enables or disables transitions, based on the set of markings consistent with the observation $w$, so as to guarantee that the state of the system remains *legal*. In the right-hand side scheme specifications are given in terms of legal sequences, so the controller computes a control pattern $\gamma(\mathcal{S}(w))$ that enables or disables transitions, based on the current estimation of the set of sequences consistent with the observation $w$, so as to guarantee that the system only generates *legal* words.

Other motivations for state estimation are: fault diagnosis, monitoring the evolution of a partially observed system, surveillance/intrusion detection problems, testing (i.e., determining the final state after a test), opacity (i.e., guaranteeing that the current or the initial state remains ambiguous), etc.

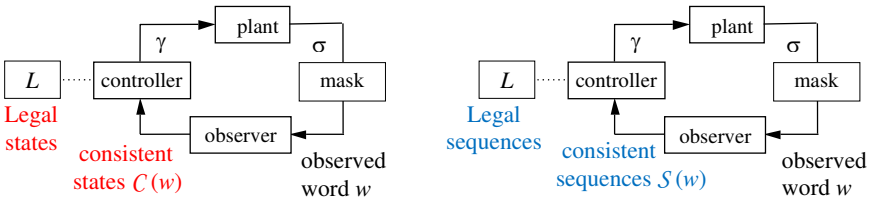


Figure 4: State-feedback control in the case of specifications in terms of legal states (left) and legal sequences (right).

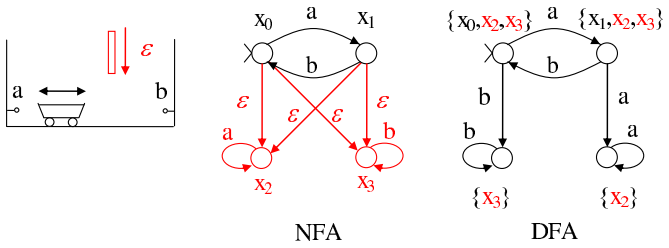


Figure 5: The AGV in Example 7.1.

The problem of state estimation under the assumption of known initial state has been first investigated in the automata framework and can be solved using an automata determinization procedure as shown via the following example.

**Example 7.1** *Consider the AGV shown on the left-hand side of Fig. 5. It moves on a track from left to right and viz. automatically. Two contacts at the end points of the track generate a signal when the AGV touches them. An obstacle may block the path but there is no sensor to detect it thus the occurrence of such an event is labeled with the empty string $\varepsilon$: when the obstacle is present, the AGV will be trapped in one of the two subtracks on its left or on its right.*

*The behavior this system can be described by the nondeterministic finite automaton (NFA) in Fig. 5. State $x_0$ (resp., $x_1$) denotes that the obstacle is not blocking the path and the AGV is moving from right to left (resp., left to right). State $x_2$ (resp., $x_3$) denotes that the obstacle is blocking the path and the AGV is trapped on the left (resp., right) subtrack regardless of the direction of its movement.*

*A state observer may be obtained by simply computing the deterministic finite automaton (DFA) equivalent to the given NFA, i.e., the DFA that generates the same observable language. In the DFA each state obviously corresponds to a set of states of the NFA. The state reached on the DFA after a word $w$ is observed gives the set of states of the NFA consistent with $w$. The observer for the considered example is shown in right-hand side of Fig. 5.* ■

The automata determinization procedure presents the main advantage of generality, indeed it works for any NFA being $\mathcal{L}_{NFA} = \mathcal{L}_{DFA} = \mathcal{L}_{reg}$ where $\mathcal{L}_{NFA}$, $\mathcal{L}_{DFA}$ and $\mathcal{L}_{reg}$ denote the set of languages generated by NFA, DFA, and the set of regular languages, respectively. Unfortunately a determinization procedure cannot be applied to Petri nets in the general case since it holds

$\mathcal{L}_{\text{det}} \subsetneq \mathcal{L}_\lambda$ where $\mathcal{L}_{\text{det}}$ is the set of deterministic PN languages, and $\mathcal{L}_\lambda$ is the set of arbitrary PN languages where nondeterminism is due both to silent events (transitions labeled with the empty string $\varepsilon$) and to undistinguishable events (transitions sharing the same label).

Moreover, the automata determinization procedure presents a series of limitations:

- each set $\mathcal{C}(w)$ must be exhaustively enumerated;
- to compute $\mathcal{C}(w)$ we need to compute $\mathcal{C}(w')$ for all prefixes $w' \preceq w$;
- if the NFA has $n$ states, the DFA can have up to $2^n$ states;
- it does not allow to reconstruct the set $\mathcal{S}(w)$ of consistent sequences.

For the above reasons different techniques have been proposed in the PN literature. In this section we recall one approach suggested by the two authors of this article and their team whose main features can be summarized in the following three items.

- At each step the set of consistent markings is represented by the integer solutions of a linear constraint set thus one needs not exhaustively enumerate all consistent markings.
- The linear constraint set depends on some parameters (the so-called basis markings) that can be recursively computed each time a new event is observed.
- We pose some structural constraints but the same procedure works for bounded and unbounded nets.

Four are the main assumptions:

- the structure of the net $N$ is known;
- the initial marking $M_0$ is known;
- the net is labeled: when a sequence $\sigma \in T^*$ fires we observe the word $w = L(\sigma) \in E^*$;
- the net obtained from $N$ removing all transitions that are observable is acyclic.

The solution we propose is based on two notions of *justifications* and *basis markings* that make it possible to avoid the exploration of all the reachability set, but only require to explore the smaller basis marking set.

In the following we denote $T_u$ the set of unobservable transitions, and $T_o$ the set of observable transitions, that may obviously also be undistinguishable. The set of justifications of the observed word $w \in E^*$ is

$$\mathcal{J}(w) = \{(\sigma_o, \sigma_u), (\sigma'_o, \sigma'_u), \ldots\}$$

where in each couple

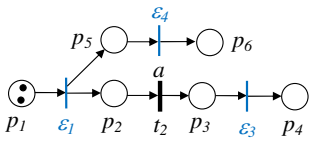


Figure 6: The labeled Petri net system in Example 7.2.

- sequence $\sigma_o \in T_o^*$ is such that $L(\sigma) = w$,
- sequence $\sigma_u \in T_u^*$ (called *justification*) is a sequence of unobservable transitions that must be interleaved with $\sigma_o$ to produce a firable sequence and whose firing vector $\pi(\sigma_u)$ is minimal.

**Example 7.2** *Consider the labeled Petri net system in Fig. 6 where $t_2$, labeled $a$, is the only observable transition.*

*If $a$ is observed, it holds $\mathcal{J}(a) = \{(t_2, \varepsilon_1)\}$: in fact, we can infer that $\varepsilon_1$ must have fired to enable $t_2$. Note that also $\varepsilon_3$ and $\varepsilon_4$ may have fired, but their firing is not necessary to "justify" the firing of $t_2$.* ■

For each couple $(\sigma_o, \sigma_u) \in \mathcal{J}(w)$, the marking[1]

$$\boldsymbol{m}^b = \boldsymbol{m}_0 + \boldsymbol{C}_o \cdot \boldsymbol{\sigma}_o + \boldsymbol{C}_u \cdot \boldsymbol{\sigma}_u,$$

i.e., the marking reached firing $\sigma_o$ interleaved with the minimal justification $\sigma_u$, is called *basis marking* and the firing vector $\boldsymbol{\sigma}_u$ is called its *j-vector* (or *justification-vector*).

The set $\mathcal{M}(w)$ is the set of pairs (basis marking - relative j-vector) that are consistent with $w \in E^*$, and $\mathcal{M}^b(w)$ is the set of basis markings that are consistent with $w \in E^*$.

**Example 7.3** *Consider again the Petri net system in Fig. 6. If $a$ is observed, it holds*

$$\mathcal{M}(a) = \{(\boldsymbol{m}_1^b, \boldsymbol{\sigma}_{u,1})\} = \{([1\ 0\ 1\ 0\ 1\ 0]^T, [1\ 0\ 0]^T)\}$$

*where $\boldsymbol{\sigma}_{u,1} = [\varepsilon_1\ \varepsilon_3\ \varepsilon_4]^T = [1\ 0\ 0]^T$.* ■

The following main result has been proved in (Cabasino *et al.*, 2011). It shows how the set of consistent markings can be written in linear integer terms.

**Theorem 7.4** *Let us consider a net system $\langle N, \boldsymbol{m}_0 \rangle$. Assume that the net obtained removing all transitions that are observable (the unobservable subnet) is acyclic. For any $w \in L^*$ it holds that*

$$\begin{aligned} \mathcal{C}(w) &= \bigcup_{\boldsymbol{m}^b \in \mathcal{M}^b(w)} R(N_u, \boldsymbol{m}^b) \\ &= \bigcup_{\boldsymbol{m}^b \in \mathcal{M}^b(w)} \{\boldsymbol{m} \in \mathbb{N}^m \mid (\exists \boldsymbol{y} \geq \boldsymbol{0})\ \boldsymbol{m} = \boldsymbol{m}^b + \boldsymbol{C}_u \cdot \boldsymbol{y}\}. \end{aligned}$$

[1] Here $\boldsymbol{C}_o$ and $\boldsymbol{C}_u$ denote the incidence matrix of the observable and unobservable subnet, while $\boldsymbol{\sigma}$ denotes the firing vector (or Parikh vector) of sequence $\sigma$.

The set $\mathcal{M}(w)$ and $\mathcal{M}^b(w)$ can be recursively computed. Moreover, for bounded nets this can be done off-line computing a *Basis Reachability Graph* with as many nodes as the number of basis markings (Cabasino *et al.*, 2011).

This is a significant advantage with respect to automata based approaches since the set of basis markings is always a subset (usually a strict subset) of the set of reachable markings. In particular, there exist nets where the size of the reachability graph is exponential in some net parameters, while the set of basis marking is constant or grows linearly (Cabasino *et al.*, 2011).

## 8 Conclusions

Petri nets play an important role in the area of systems theory and in particular in the domain of discrete event systems. In this article we have highlighted by means of several examples what are the main features that make Petri nets such a useful model and what are the advantages that they can offer with respect to other formalisms, focusing in particular on the two fundamental issues of control design and state estimation.

## References


Badouel, E. and P. Darondeau (1998). Theory of regions. *Lecture Notes in Computer Science* **1491**, 529–586.

Basile, F., P. Chiacchio and G. De Tommasi (2009). An efficient approach for online diagnosis of discrete event systems. *IEEE Trans. on Automatic Control* **54**(4), 748–759.

Benveniste, A., E. Fabre, S. Haar and C. Jard (2003). Diagnosis of asynchronous discrete event systems: A net unfolding approach. *IEEE Trans. on Automatic Control* **48**(5), 714–727.

Bourdeaud'huy, T. and P. Yim (2004). Synthèse de rèseaux de Petri à partir d'exigences. In: *Actes de la 5me conf. francophone de Modélisation et Simulation*. Nantes, France.

Cabasino, M. P., A. Giua and C. Seatzu (2010). Fault detection for discrete event systems using Petri nets with unobservable transitions. *Automatica* **46**(9), 1531–1539.

Cabasino, M.P., A. Giua and C. Seatzu (2007). Identification of Petri nets from knowledge of their languages. *Discrete Event Dynamic Systems: Theory and Applications* **17**(4), 447–474.

Cabasino, M.P., A. Giua and C. Seatzu (2012). Structural analysis of Petri nets. In: *Control of Discrete-Event Systems. Automata and Petri Net Perspectives*. Vol. 433 of *Lecture Notes in Control and Information Science*. Springer. pp. 213–234.

Cabasino, M.P., A. Giua, M. Pocci and C. Seatzu (2011). Discrete event diagnosis using labeled Petri nets. an application to manufacturing systems. *Control Engineering Practice* **19**(9), 989–1001.

Caines, P.E., R. Greiner and S. Wang (1988). Dynamical logic observers for finite automata. In: *Proc. 27th IEEE Conf. on Decision and Control*. Austin, Texas.

Cassandras, C.C. and S. Lafortune (2008). *Introduction to Discrete Event Systems - Second Edition*. Springer.

Chu, F. and X.L. Xie (1997). Deadlock analysis of Petri nets using siphons and mathematical programming. *IEEE Trans. on Robotics and Automation* **13**(6), 793–804.

Colom, J.M. and M. Silva (1990). Improving the linearly based characterization of P/T nets. Vol. 483 of *Lecture Notes in Computer Science*. Springer Verlag.

Cordone, R., A. Nazeem, L. Piroddi and S. Reveliotis (2013). Designing optimal deadlock avoidance policies for sequential resource allocation systems through classification theory: existence results and customized algorithms. *IEEE Trans. on Automatic Control* **58**(11), 1–16.

David, R. and H. Alla (2005). *Discrete, continous and hybrid Petri nets*. Springer, Berlin, Heidelberg.

Dotoli, M., M.P. Fanti, A.M. Mangini and W. Ukovich (2009). On-line fault detection of discrete event systems by Petri nets and integer linear programming. *Automatica* **45**(11), 2665–2672.

Ezpeleta, J., J.M. Colom and J. Martinez (1995). A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation* **11**(2), 173–184.

Genc, S. and S. Lafortune (2007). Distributed diagnosis of place-bordered Petri nets. *IEEE Trans. on Automation Science and Engineering* **4**(2), 206–219.

Giua, A. (2012). Supervisory control of Petri nets with language specifications. In: *Control of Discrete-Event Systems. Automata and Petri Net Perspectives*. Vol. 433 of *Lecture Notes in Control and Information Science*. Springer. pp. 235–256.

Giua, A. and C. Seatzu (2002). Observability of place/transition nets. *IEEE Trans. on Automatic Control* **47**(9), 1424–1437.

Giua, A., F. DiCesare and M. Silva (1992). Generalized mutual exclusion constraints on nets with uncontrollable transitions. In: *Proc. 1992 IEEE Int. Conf. on Systems, Man and Cybernetics*. Chicago, IL, USA.

Hiraishi, K. (1992). Construction of a class of safe Petri nets by presenting firing sequences. *Lecture Notes in Computer Science* **616**, 244–262.

Holloway, L. E., B. H. Krogh and A. Giua (1997). A survey of Petri net methods for controlled discrete eventsystems. *Discrete Event Dynamic Systems* **7**(2), 151–190.

Iordache, M. and P. Antsaklis (2005). A survey on the supervision of Petri nets. In: *Petri nets 2005*. Miami, FL, USA.

Iordache, M.V., J.O. Moody and P.J. Antsaklis (2002). Synthesis of Deadlock Prevention Supervisors using Petri Nets. *IEEE Trans. on Robotics and Automation* **18**(1), 59–68.

Kumar, R., V. Garg and S.I. Markus (1993). Predicates and predicate transformers for supervisory control of discrete event dynamical systems. *IEEE Trans. on Automatic Control* **38**(2), 232–247.

Li, Z.W. and M. Zhou (2004). Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems. *IEEE Trans. Systems, Man and Cybernetics - Part A* **34**(1), 38–51.

Miyagi, P.E. and L.A.M. Riascos (2010). Modeling and analysis of fault-tolerant systems for machining operations based on Petri nets. *Control Engineering Practice* **14**(4), 397–408.

Moody, J.O. and P.J. Antsaklis (1998). *Supervisory control of discrete event systems using Petri nets*. Kluwer Academic Publishers.

Moody, J.O., K. Yamalidou, M.D. Lemmon and P.J. Antsaklis (1996). Feedback control of Petri nets based on place invariants. *Automatica* **32**(1), 15–28.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings IEEE* **77**, 541–580.

Ozveren, C.M. and A.S. Willsky (1990). Observability of discrete event dynamic systems. *IEEE Trans. on Automatic Control* **35**(7), 797–806.

Park, J. and S.A. Reveliotis (2000). Algebraic Synthesis of Efficient Deadlock Avoidance Policies for Sequential Resource Allocation Systems. *IEEE Trans. on Robotics and Automation* **16**(2), 190–195.

Peterson, J.L. (1981). *Petri Net Theory and Modelling of Systems*. Prentice Hall, Englelwood Cliffs, NJ.

Prock, J. (1991). A new tecnique for fault detection using Petri nets. *Automatica* **27**(2), 239–245.

Ramadge, P.J. (1986). Observability of discrete-event systems. In: *Proc. 25th IEEE Conf. on Decision and Control*. Athens, Greece.

Ramadge, P.J. and W.M. Wonham (1989). The control of discrete event systems. *Proceedings of the IEEE* **77**(1), 81–98.

Reveliotis, S.A. (2007). Implicit Siphon Control and its Role in the Liveness Enforcing Supervision of Sequential Resource Allocation Systems. *IEEE Trans. Systems, Man and Cybernetics - Part A* **37**(3), 319–328.

Seatzu, C., M. Silva and J.H. van Schuppen (Ed.) (2012). Control of discrete event systems. Automata and Petri net perspectives. Vol. 433 of *Lecture Notes in Control and Information Science*. Springer.

Silva, M., J.M. Colom and J. Campos (1992). Linear algebraic techniques for the analysis of Petri nets. In: *In: Recent Advances in Mathematical Theory of Systems, Control, Networks, and Signal Processing II*. Mita Press. pp. 35–42.

Sreenivas, R.S. (2002). On minimal representations of Petri net languages. In: *Proc. IFAC WODES'02: 6th Work. on Discrete Event Systems*. Zaragoza, Spain.

Viswanadham, N., Y. Narahari and T.L. Johnson (1990). Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models. *IEEE Trans. on Robotics and Automation* **6**(6), 713–723.

Wu, Y. and C.N. Hadjicostis (2005). Algebraic approaches for fault identification in discrete-event systems. *IEEE Trans. on Robotics and Automation* **50**(12), 2048–2053.