A Systems Theory Perspective of Discrete Event Dynamic Systems: The Petri Net Paradigm

Manuel SILVA and Enrique TERUEL
Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior, Universidad de Zaragoza
María de Luna 3, E-50015 Zaragoza, Spain.
e-mail: {msilva,eteruel}@mcps.unizar.es

Abstract— **The intention of this invited paper is to offer a view of Petri net (PN) based formalisms as a conceptual framework for the modeling of discrete event dynamic systems (DEDS), rather than to survey the topic. The introduction of autonomous and interpreted PNs is presented in a way that is close to systems/control engineers. The diversity of interpreted PN formalisms, suited to deal with diverse purposes but sharing basic common principles, turns PNs into a conceptual framework or paradigm for the modeling of DEDS. The paper is concluded by reflecting on a few characteristics of formalisms in general and PN ones in particular.**

I. Introduction

A great number of systems can be naturally viewed as *discrete event dynamic systems (DEDS)*. We emphasize that this is a *view* rather than an inherent characteristic of such systems. For instance, we can view a water tank as a continuous system, containing some amount of water, or we can be simply interested on whether it contains more or less than a certain quantity. In what follows, by a DEDS we mean either the DEDS view of the system or the system which is viewed in this way. Compared to *continuous systems*, either in continuous or discrete time:

- *The state space of a DEDS is discrete*, that is, there is a countable (perhaps infinite) number of distinct states.
- The DEDS evolution is not directly seen as due to time passing; instead, *the state changes are driven by events*. Of course, events happen in time, so time drives the evolution, although *indirectly*, through these events. Depending on whether state changes can occur at any time or only in precise instants, a DEDS is said to be *asynchronous* or *synchronous*.

The DEDS view of systems has been present in systems theory for a long time. It is gaining growing importance as far as the number and complexity of DEDS are ever increasing by the development of computer-based technologies, e.g., in automation, communications, etc. Their complexity and required flexibility to cope with rapid technological and market changes make for the importance of good design and easy operation. *Formal methods* may be helpful in this respect. Some expected benefits of the use of formal methods are:

- A better understanding of the system, which is gained by the obligation imposed by formal methods to think hard on the problem. This better understanding helps in removing incompleteness and contradictions, identifying properties, or discovering potential solutions.
- A sound basis for the development. This is paid off in terms of increased confidence in a design (errors can be detected in early stages, limiting their appearance during operation), correct dimensioning, guided implementation and documentation, re-usability, etc.

The necessity of formal methods is beyond question in many mature engineering disciplines (electrical, mechanical, control, etc.) although it is still in question in emerging disciplines such as software engineering [Hal90], [BH95]. Different formalisms for DEDS are being proposed and experienced, and their links and relative merits are being investigated. DEDS have so many facets that it is expected that they are approached from different angles, providing *complementary* views of systems serving diverse purposes.

A first step in the application of formal methods is to obtain a *formal model* of the system of interest. While in a colloquial or artistic sense a model is a *reference* or standard to be imitated (e.g., in a sculpture or in one's behavior), in the domain of systems theory it is rather a *representation* of some aspects of a system (either existing or being conceived) for the purpose of understanding (analysis) or creation (design). In this latter sense, a model aspires to represent a part of the real world to the degree needed in the application for which it is intended. Some models are *formal*, typically based or rooted on mathematical representations. For instance, a set of differential equations may be a formal model of a given continuous system, suitable for the purpose of analyzing its time response or designing a feedback controller for it.

A *formalism* is a conceptual framework that allows to obtain a kind of formal models of systems. For instance, ordinary differential equations are a formalism for the modeling of the dynamic behavior of continuous systems with lumped parameters. Some examples of formalisms for DEDS with diverse purposes are *sequential/state machines* or *state diagrams* [Boo67], [Lew85] (for functional description), *Markov chains* and *queueing networks* [Kan92], [VN92] (for performance evaluation), *PERT graphs* and *conjunctive/disjunctive graphs* [Fre82], [CC88] (for scheduling), etc.

In view of the long life cycle of a given system (along which it is conceived, analyzed from different perspectives, implemented, and operated) and the diversity of application domains, it seems desirable to have a *family of formalisms* rather than a collection of unrelated or weakly related formalisms. Following Kuhn's definition, a *paradigm* is "the total pattern of perceiving, conceptualizing, acting, validating, and valuing associated with a particular image of reality that prevails in a science or a branch of science". In particular, for us a *modeling paradigm* is a conceptual framework that allows to obtain formalisms from some common concepts and principles, with the consequent economy and coherence, among other benefits. In this paper Petri nets (PN) are seen as a modeling paradigm for DEDS.

The structure of the paper is as follows: In Section II, autonomous PNs are introduced in a way that is close to systems/control engineers. We make emphasis on some features of PN models of DEDS such as the internal representation in terms of local/distributed states, the even-handed treatment of states and transitions, the clear separation of concurrency and non-determinism, or the complementary algebraic-oriented and graph-oriented representations. Then we show in Section III how to incorporate information from the environment to the abstract PN formalism by way of interpretations, leading to a family of related formalisms. Finally, a reflection on some general requirements and judgment of qualities of formalisms for DEDS is made in Section IV, specially looking at PNs.

II. Autonomous PNs as a Formalism for DEDS

In this section we concentrate on the *logic* behavior of a DEDS, that is, the possible states and evolutions of the system disregarding time. Abstracting from time can be done in the case of DEDS where the evolutions are driven by the occurrence of events, in contrast with continuous systems where the system may evolve simply due to time passing.

A. Encountering Place/Transition Net Systems

Models for continuous systems are based on differential or difference equations (corresponding to whether the time is seen as continuous or discrete). Two basic approaches to the modeling are:

- *External* description: The system is regarded as a *black box*, and the direct explicit relation between inputs and outputs is described. For instance, *transfer functions* or *impulse responses* are external descriptions of linear time-invariant continuous systems.
- *Internal* description: Some (minimal number of) variables are identified to describe the *state* of the system, summarizing its dynamic history. Given the initial state and the inputs from then on, the state evolution and the outputs can be computed. Therefore there is an indirect explicit relation between inputs and outputs.

In the case of DEDS, a *regular expression* can be considered as an external description (the output of the system is this whenever the input pattern is that). Although this is adequate in some applications, such as specifying the behavior of an electronic lock, in most cases it is interesting if not necessary to capture the notion of state and somehow describe the internal structure of the system.

Let us have a look at the internal description of continuous systems. The state variables are a (minimal) representation of the past dynamic history of the system, possibly corresponding to magnitudes of the real world system (e.g., the voltage of a node in an electric circuit, the population of a species in an ecosystem, etc.). The selection of the appropriate state variables in each case is part of the modeling process and usually requires knowledge of the domain, ingenuity, and methodology. Since magnitudes are continuous they are *coded* as real numbers. For notational convenience the state variables are collected in a *state vector*, $\mathbf{x}$. Similarly for the inputs or excitation, $\mathbf{u}$. Now the change of the state variables needs to be described. When time is regarded as continuous (and represented by a real variable t), this means describing the time derivatives of the state variables:

$$\dot{\mathbf{x}}(t) = \varphi(\mathbf{x}(t), \mathbf{u}(t), t).$$

When time is regarded as discrete (and represented by an integer variable k, that is, $t = k \cdot \theta$ where θ is the *sampling period*), the *next state* is described:

$$\mathbf{x}(k+1) = \phi(\mathbf{x}(k), \mathbf{u}(k), k).$$

These equations are known as *state equations* of the system. (Another vector equation is used to describe the outputs — measurable real world signals — as a function of the state, inputs, and time, but this is not relevant here.) Some advantages of this state representation are that it is more adequate to cope with complex systems (multivariable, non linear, time varying), has a convenient mathematical representation, and allows to cope with optimization problems. In the case of *time-invariant* systems, i.e., those whose structure/parameters are constant over time, the state equation is reduced to:

$$\mathbf{x}(k+1) = \phi(\mathbf{x}(k), \mathbf{u}(k)),$$

and when the dependence can be assumed to be linear:

$$\mathbf{x}(k+1) = \mathbf{F} \cdot \mathbf{x}(k) + \mathbf{G} \cdot \mathbf{u}(k).$$

We turn now our attention to DEDS. These are systems with non-numerically-valued states, inputs, and outputs. The discrete nature of the states makes their number countable (often finite). The state can be represented symbolically, or it can be coded as a number (typically for implementation purposes). A global representation of the state is useful in some applications, especially when the system is a single entity (e.g., a single queue, a machine that can be idle, working, blocked, or out of service, etc.). The state is represented *globally* in formalisms such as state machines or Markov chains. Nevertheless when the system is composed by several entities that interact, a global representation of the state does not reflect the structure of the modeled system and is usually cumbersome due to the large number of combinations of local states that lead to different global states. In such cases it is better to select a collection of *local* state variables forming a state vector that we denote here by $\mathbf{m}$. Without loss of generality, we assume that the local state variables are *coded*, and range over the naturals. (When the number of states is finite even binary state variables can be taken.) Again we note that the selection of variables is a crucial modeling task usually requiring knowledge of the domain, ingenuity, and methodology. For instance, if a sequential component is identified in the system we can take a variable for each possible state so that when the component is in a given state the value of the corresponding variable is one, and zero otherwise. Or if we find a component of the system that holds items (e.g., a store) it can be represented by a variable whose value is the number of present items, etc. (In general, differently from continuous systems, minimality in the number of state variables is *not* required; actually, from the understanding and implementation points of view, non-minimality — even obvious *redundancies* — may be interesting.)

In a DEDS state changes at discrete points in time, driven by the occurrence of events. In other words, the state does not change simply because time passes, unless this is an event for our system (e.g., a clock). Abstracting from the particular events that drive state changes or state transitions, we assume now that *there are a finite number of atomic state transition patterns*, that we call (individual) *transitions*. Depending on whether these transitions can occur at any time or only in precise instants, a DEDS is said to be *asynchronous* or *synchronous*, and the time is seen as continuous or discrete, respectively. If we abstract from time (i.e., we are only interested in the evolution of the state irrespective of the instant when it happens) we can assume that the "time variable" is discrete, corresponding to the ordering of "instants" at which transitions have occurred. Several individual transitions may occur at the same "instant", e.g., if they are *independent* of each other and it is not known the precise order in which they occurred (in a distributed environment it is not always possible to order events totally). Therefore:

$$\mathbf{m}(k+1) = \delta(\mathbf{m}(k), \mathbf{s}(k)),$$

where in the k-th "instant" the individual transition i has occurred $s_i(k)$ times.

Between two state transitions the state is memorized. Thus, without loss of generality, the function of state change can be broken up in two parts, the *memory* and the *innovation*:

$$\mathbf{m}(k+1) = \mathbf{m}(k) \oplus \Gamma(\mathbf{m}(k), \mathbf{s}(k)),$$

where $\oplus$ is some operator and $\Gamma(\mathbf{m}(k), \mathbf{0})$ must be the neutral element with respect to $\oplus$. We assume now that *the extent of change produced by a transition is fixed*, that it does not depend on the state at which it occurs. Then:

$$\mathbf{m}(k+1) = \mathbf{m}(k) \oplus \Gamma(\mathbf{s}(k)).$$

Since the state is a vector of natural numbers, the innovation produced by a given transition can be represented without loss of generality by a vector of integers, accounting for the difference between the next and the current state when such transition occurs, the *displacement* of the state produced by the transition. The negative entries account for state variables whose value decreases, the positive account for those whose value increases, and the null ones for those whose value is not affected. Let us write the innovations corresponding to the individual transitions as columns of a matrix $\mathbf{C}$: the i-th column, C_i, contains the state change associated to individual transition i. The state change produced by the occurrence of $\mathbf{s}$ (several individual transitions at the same "instant") is the corresponding linear combination of columns of $\mathbf{C}$:

$$\mathbf{m}(k+1) = \mathbf{m}(k) + \mathbf{C} \cdot \mathbf{s}(k).$$

The above equation imposes a limitation to the transitions that can occur at a given state, because the state variables were assumed to range over the naturals: the fixed extent of change associated with the transition must be possible at that state. We assume now that *a transition is enabled to occur at a state if and only if the fixed extent of change associated with the transition is possible at that state*; that is, possibility of the state change is not only necessary *but also sufficient* for the enablement.

A DEDS under the above assumptions (i.e., finite number of atomic individual transitions which are enabled at a state if and only if the fixed extent of change they produce is possible at that state) can be represented by a *vector addition system* [KM69], defined by the initial state and the displacement vectors (in the above notation, $\mathbf{m}(0)$ and the columns of $\mathbf{C}$).

Let us further illustrate the above *logic of enablement*. The natural valued state variables can be considered as *stores/counters* (the actual value is the number of items). The occurrence of a transition *consumes* items from some state variables (those corresponding to the negative entries) and *produces* items in others (for the positive entries). A transition is enabled if and only if there are enough items to remove. If we want to adhere to this interpretation in terms of *consumption/production*, we realize that a zero entry in the innovation vector of a transition may be due to the fact that such transition removes as many items as it produces in some state variable. More generally, the innovation or displacement vectors represent only the *net effect* of the consumption and production. To account for this kind of situations we can separate the positive and negative parts of the innovations: $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$, and require that $\mathbf{m} \geq \mathbf{Pre} \cdot \mathbf{s}$ for $\mathbf{s}$ to be enabled at $\mathbf{m}$. (When there are no *self-loops*, i.e., transitions that remove items from and put items in the same state variable, $\mathbf{C}$ contains all the information.)

This is known in the literature as a *place/transition (Petri) net system* [HC70], [Hac75], [Pet81], [BRA83], [Rei85], [Sil85], [Mur89]. A place/transition net is the fixed or *static* structure:

$$\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle,$$

where P is the set of state variables, T is the set of atomic individual transitions, and $\mathbf{Pre}$ and $\mathbf{Post}$ are $|P| \times |T|$ dimensional matrices whose columns describe the consumption and production associated to the corresponding transitions, respectively; a net together with the initial state $\mathbf{m_0}$ is called a place/transition net system.

There are diverse alternative forms for defining net systems, often computer science oriented. Historical remarks can be found in [Pet81], [Mur89]. The seminal work is Petri's dissertation [Pet62]. His axiomatic view, based on the fundamental notion of causal independence, is exposed in [Pet87].

B. Viewing DEDS through Place/Transition Systems

In systems theory, it is habitual to define a system as a collection of *objects* and their *relations.* Objects are characterized by their *attributes*, some of which are fixed while others are variable. The value of the variable attributes defines, perhaps in a not minimal way, the *state* of the system. We can identify the state variables, P, and the individual transitions, T, as the objects in our system. It can be said that state variables are *passive* objects and transitions are *active*, in the sense that the value of state variables *is changed by* the occurrence of transitions. The consumption/production interrelation can be defined by a relation $F \subseteq (P \times T) \cup (T \times P)$ and a valuation of this relation, $W : F \longrightarrow I\!N^+$, leading to an alternative definition of a net:

$$\mathcal{N} = \langle P, T, F, W \rangle.$$

Let us illustrate the semantics of this weighted flow relation: $(p,t) \in F$ and $(p',t) \in F$ means that t consumes $W(p,t)$ items from p and $W(p',t)$ from p'. As another example, $(t,p) \in F$ and $(t',p) \in F$ means that the value of p can be increased by t or t', etc.

A current, often convenient, technique to represent interrelations is by use of diagrams. The various components of a system are represented by some kind of nodes and connecting lines represent relations between the corresponding components. Diagrams may inform on the physical structure of a system, its computational structure, or both. Diagrammatic representations of continuous systems are, for instance, circuit diagrams, block diagrams, and bond graphs. A DEDS can also be represented by a diagram. For instance, a state diagram represents a state machine by depicting the possible *states* as nodes connected by arrows accounting for the transitions between states; a PERT graph represents the precedence relations (arcs) between *tasks* (nodes); a queueing network represents *queues* and *stations* as nodes and the *routing* of customers by arcs. Notice that some diagrams represent states as nodes (state diagram), events as nodes (PERT graphs), or both (queueing networks — queues are local state variables and stations are state transformers).

The standard representation of place/transition net systems uses two kinds of nodes: circles for the local state variables or *places*, and bars or boxes for the individual transitions. Adhering to the interpretation of items inside stores/counters, the value of a variable is depicted as a number of marks or *tokens* inside the corresponding place. Therefore the global state of the system is represented by the *marking* of the places. The extent of change produced by a transition is indicated by arrows connecting the places and the transition in the direction of token flow, labeled by the number of tokens consumed/produced at the occurrence of a transition (this is a straightforward graphical representation of the *flow* relation F by means of directed arcs and the *weighting* W by means of the labels). Figure 1 shows a sample place/transition system before and

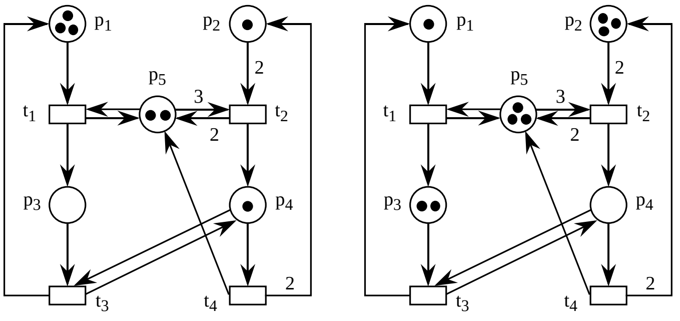

Fig. 1. A place/transition system. Illustration of the evolution of the state by the occurrence of the step $2t_1 + t_4$.

after the occurrence (or *firing*) of the *step* $2t_1 + t_4$ (two occurrences of t_1 and one of t_4). The evolution of the state can be seen as a sort of game, the "token game". Each "move" corresponds to a legal state change (produced by the occurrence of one or more individual transitions) and consists on removing tokens from some places and placing tokens on others (the total number of tokens may be changed in a "move", although usually some token conservation laws can be found).

It is remarkable that with so few objects and rules diverse fundamental phenomena appearing in (concurrent) DEDS can be succinctly captured. The possibility of *independent* (or "simultaneous") occurrence of several individual transitions in a *step* accounts for the *concurrency* between transitions in a very natural way. In particular it implies that every ordering of the concurrent transitions is possible, while the reverse is not always true. (As an example, in Figure 4, t_2 and t_3 can occur in any order, but they are of course not concurrent or independent since they both require the token in place R.) In fact, it is said that *"true concurrency"* is represented, compared to defining concurrency as the possibility of all *interleaved* sequential observations. For instance, in the example of Figure 1, t_1 and t_4 occurred concurrently, even t_1 occurred *self-concurrently.* In a system with transitions only, they would all be concurrent. Places constraint this concurrency establishing *synchronic dependencies* between transitions according to the global structure and initial marking. When it is not possible that everything that is enabled occurs in a step we speak of *conflict.* For instance, in Figure 1 (right) t_1 and t_2 compete for the tokens in p_5. It may not be obvious whether conflicts will arise; sometimes this depends on the order of occurrence of some independent/concurrent transitions. This is known as *confusion.* For instance, in Figure 1 (left) both t_1 and t_4 can occur independently. If t_1 fires first then a conflict between t_3 and t_4 appears, that may or may not be solved in favor of t_4, while if t_4 is fired first such conflict does not appear. From the possibility of representing fundamental phenomena we obtain the potentiality to model virtually every schema in parallel and distributed systems: sequence, alternative, iteration, fork-join, rendez-vous, message-passing, (mutex) semaphores, etc. Subtle or paradoxical behaviors of concurrent systems are reflected in simple net models; for instance, we shall illustrate later when time is incorporated (with Figure 4), the possibility of slowing down a system by speeding up a part of it. In this sense, it is not unusual that PNs are found to clarify subtleties in distributed algorithms or communication protocols, or that they are used to express the semantics of other formalisms.

Place/transition net systems are an *operational formalism* for DEDS, i.e., they state *how* the system works, differently from *denotational formalisms* (e.g., logic-based) that state *what* the system is intended to do. In fact, place/transition diagrams clearly show the computational structure of the modeled system. The *locality* of the state variables and transitions makes possible to respect also the physical structure of the modeled system. Figure 2 shows a net model for a two-machines manufacturing flow line with

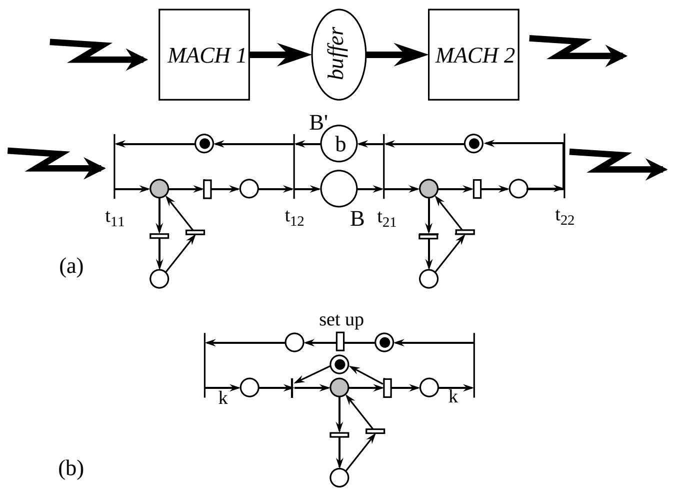

Fig. 2. A producer-buffer-consumer system: (a) Net model. (b) Model of machine accepting batches.

an intermediate store acting as a buffer against the possible disruptions due to failures at machines. It is not difficult to recognize the sub-models describing the machines (operation dependent failures) and the buffer. The firing of transition t_{11} represents the arrival of a part to the first machine while parts exit through t_{22} (merging these transitions with others from models of other subsystems could be done to analyze this flow line embedded in its environment). The buffer is modeled by places B (parts processed by *MACH 1* waiting for *MACH 2*) and B' (unused locations in the buffer), and transitions t_{12} and t_{21} (deposit and withdrawal, respectively). Assume now machines accept *batches* of k parts (e.g., due to the pallet size), although they process parts one by one, and that there is a *set-up time* between two successive batches. Clearly we can substitute the models of the machines by the subnet shown in Figure 2 (b), while the buffer places now contain pallets instead of parts.

C. Abbreviation: High-Level PNs

Assume we find a sort of box structure in matrix $\mathbf{C}$ (strictly speaking we should consider $\mathbf{Pre}$ and $\mathbf{Post}$, but to simplify the discussion we consider that $\mathbf{C}$ contains all the information), for instance:

$$\mathbf{C} = \left(\begin{array}{c|c|c|c} - & + & 0 & 0 \\ \hline 0 & + & - & - \\ \hline 0 & 0 & + & - \\ \hline + & - & 0 & + \end{array} \right)$$

where a "+" box contains positive (and null) entries, a "$-$" box contains negative (and null) entries, and a "0" box contains only null entries. Then we can group or *fold* all the places and transitions according to this box structure, leading to a (smaller) number of (folded) places and transitions. The new places still play the role of state variables, but their values are somehow *structured* (e.g., represented by a vector with as many components as there were original places). The folded transitions still play the role of state transformers, and the change in the place values produced by its occurrence is described by the function that is represented by the corresponding box; each column represents an *occurrence mode* of the folded transition.

By doing this folding we have introduced a *second level* in the structure of the model: in addition to the structure of state variables and transitions, these are now also structured in the form of *data types* and *functions*, which are depicted by labeling places, transitions, and arcs.

Of course the folding process should not be done arbitrarily, but guided by the nature of the modeled system. Actually, one directly obtains the folded model by firstly identifying the relevant data structures in the system to be modeled. A modeling trade-off appears when deciding what to put in the *net structure* and what to put in the *data/functions structure*. As an extreme example, it is always possible to obtain a completely folded model with one place and one transition in a self-loop, and with all the information captured by the data structure and functions.

PN systems that incorporate the possibility of typed state variables are called in the literature *high level Petri nets*, and different ways of specifying the data structures and the corresponding functions have been proposed. The above line of reasoning leads to *colored* PNs [Jen91], although the breakthrough from low to high level PNs was the proposal of *predicate/transition* nets [GL81], which adds predicate logic inscriptions to net objects and relations.

A possible definition of a colored PN represents the *data types* as a finite set of finite *color classes*, $\mathcal{C}$, and the corresponding *color domains* of each place and transition, $cd : P \cup T \to \mathcal{C}$, and the *functions* in each element of the matrices $\mathbf{Pre}$ and $\mathbf{Post}$:

$$\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathcal{C}, cd \rangle.$$

As an extremely simple example, in Figure 3 a colored PN model of a modulo *max* counter is represented. In

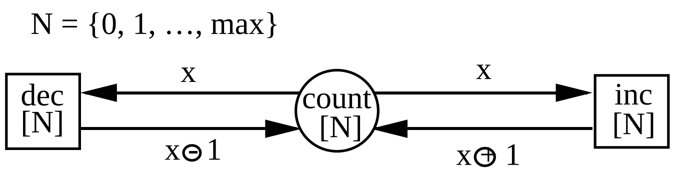

Fig. 3. A colored PN model of a counter.

this case $\mathcal{C}$ is a single color class $N = \{0, 1, \ldots, max\}$, and all the color domains are N; the functions are described by the labels, e.g., when transition *inc* occurs in mode $x \in N$ it removes a token valued x and puts a token valued $x \oplus 1$ (functionally, $\mathbf{Pre}[cont, inc] = identity$ and $\mathbf{Post}[cont, inc] = successor$). A possible initial marking is a token with value 0. If we put several tokens in the initial marking we are modeling several (independent) counters.

It is clear that to produce concise models of real world systems, the ability to deal with data types in order to group similar objects is essential. In particular, this dramatically improves the understanding, specially when there are *symmetries* in the system (e.g., customer classes in a customer/server system). But it is important to note also that this is only an *abbreviation* of the model, crucial as it may be. In fact, it is possible to obtain an equivalent place/transition net system from a high level model by *unfolding* it, not to be confused with *decoloring.* Decoloring is abstracting from the identity of tokens, and can be done whenever the precise identities are not relevant, e.g., they represent individuals of a customer class. It results in a more abstract model than unfolding, which is as detailed as the original high level model (but possibly much more cumbersome).

D. Extensions of the Basic Formalism

Let us come back to the assumptions we made to arrive at place/transition net systems as models for DEDS (the same holds for high level PNs, but we shall mainly phrase the discussion in terms of place/transition for the sake of clarity). We assumed that:

1. The number of state variables and individual transitions is finite,
2. The extent of change caused by a transition is fixed, it does not depend on the state at which it occurs, and
3. A transition is enabled to occur at a state if and only if there are enough tokens to consume.

Obviously these assumptions restrict the kind of system behaviors that can be modeled. Removing these restrictions leads to several *extensions.* Of course such extensions tend to reduce the *tractability* or *analyzability* of the model [Pet81]; in fact, they often lead to the computation power of Turing machines.

Regarding the finiteness of the number of nodes, if colored PNs had color classes of infinite cardinality, e.g., the naturals, they would be finite representations of systems with an infinite number of transitions and state variables — the unfolding would lead to a net with infinite places and transitions. This is somehow similar to *partial* differential equations compared to ordinary differential equations: the distributed state variables can be seen as an infinite number of state variables.

Regarding the fixed extent of state change, *reset nets* [AK77] (the occurrence of a reset transition empties a place), *self-modifying nets* [Val78] (the arc weights are nonhomogeneous linear combinations of place markings), or more generally *nets with marking-dependent arc cardinality* [Cia94] have been proposed.

Regarding the logic of enablement in terms of consumption, the enablement can be broken up in two parts: *concession* (there are enough tokens in the input places) and some additional "guard". Therefore, the conjunction of consumption requirements of a transition is no longer sufficient but only necessary in the logic of enablement. The guards could be chosen to be arbitrary predicates on the state, or they can be somehow restricted. The most popular of all such guards (in fact, the most popular extension) are *inhibitor arcs* [AF73], [Pet81]: the inhibiting place must be empty (or have less tokens than the weight) for the transition to be enabled. Also the complementary notion, *test* arcs, has been proposed [CH93], [LC94], [MR95], where the tested place must have enough tokens for the transition to be enabled. It is important to notice that a test arc differs from a self-loop, because the latter affects concurrency due to the consumption semantics: clearly a "read" operation is not equivalent to a "rewrite" (take and put back) operation in a distributed system!

Another way to modify the logic of enablement is by way of *priorities*: for a transition to be enabled, no transition of higher priority can be enabled. We can either define a (irreflexive antisymmetric transitive partial) relation between pairs of transitions (the priority of t is higher than that of t', etc.) [BK92], or a partial ordering (each transition has some priority level) [Pet81].

Some of the above extensions are theoretically interchangeable, since their computation power is the same. Even more, under some circumstances (typically boundedness of certain state variables) even plain place/transition net systems can simulate extended models. The existence of these connections allows to re-use some results for the analysis. Nevertheless, it is not usually convenient from a modeling point of view to artificially simulate a given extension (typically the resulting model is larger and does not reflect so well the structure of the modeled system).

III. The PN Modeling Paradigm for DEDS

The connection of a formalism and reality is provided by the *interpretation.* In a totally uninterpreted theory there is no meaning associated to the mathematical objects. For instance, the theory of graphs does not assume any particular meaning for its objects (e.g., graph nodes can represent sites, states, actions, etc.). This very abstract setting has some advantages: it is extremely general (so it can be applied in a diversity of domains, with the consequent economy) and precise. A *semi-interpreted* formalism gives a sort of generic meaning to the mathematical objects. For instance, control theory is semi-interpreted: some variables in the differential equations describe the state, others play the role of external inputs or excitation, some of which are control signals while others are perturbations, etc. But the same formulation/equations can describe systems with a very different nature (electrical, mechanical, socioeconomical, etc.). When a given model is completely interpreted, every variable has a precise meaning in terms of the real world system being modeled.

In this sense, autonomous PNs are semi-interpreted: places have the meaning of state variables, transitions are state transformers, and some rules for the dynamic behavior are provided by the logic of enablement/occurrence. We can associate a precise meaning to places and transitions (e.g., this place represents a store, this transition represents the arrival of a part, etc.) in the form of a labeling (with statements) indicating to the human observer the intent of the model. Doing so, the behavior is *not* affected.

But in many situations the association of a meaning to the net objects has stronger implications: if a transition models the end of some activity, there may be temporal constraints for its occurrence once it is enabled; or if two transitions are in conflict, their meaning may imply that there is some constraint on how this conflict should be solved. The behavior of autonomous PNs is independent of time and environment. In this sense their *non-determinism* (notice that we fixed when a transition is *enabled* to occur, but not *when* would it occur, even whether it would occur at all, or *how* a conflict would be solved) can be regarded as a total *abstraction* of time and environment. (This abstraction is even stronger than in the case of stochastic models, where some knowledge, though incomplete, is captured by pdf's — probability distribution functions.) If the constraints associated to the interpretation are taken into account, the non-determinism is reduced (or removed) and the behavior of the model *is* affected, actually restricted. This is why the adjective *interpreted* is usually regarded as synonymous of *non-autonomous* in the PN literature, while in time-invariant continuous systems non-autonomous is synonymous of *forced*, a meaning that fits also very well in our context, although is not conventionally used.

Since similar interpretations are useful in a diversity of application domains, *interpreted extensions* (simply *interpretations* in the sequel) incorporating external constraints, often in terms of time, have been proposed. They lead to different PN based formalisms sharing some basic principles. This is why we speak of a *Petri net paradigm*. Some of these formalisms developed from PNs have become standards, either by their use or by the influence of organisms, as it happens in other areas (e.g., the use of BCMP queueing networks [BCMP75] made them a standard within queueing networks, or LOTOS (Language of Temporal Ordering Specification) [ISO88] is a standardized language, based on process algebra, oriented to open distributed systems).

In the following subsections we will comment on two particular kinds of interpretations. Our selection is motivated by the relevance for automation applications. In the first, constraints on the timing and conflict resolutions are provided, leading to timed/stochastic PNs. These formalisms are used in performance evaluation and optimization, or in scheduling. In the second, the evolution is constrained by external inputs, which is interesting in control. The net model evolves in closed-loop with its environment, which is not modeled (at least at the same degree of detail, only some signals are selected to inform about its state).

A. Timed/Stochastic PNs

One among the very many possible ways to incorporate time in a PN system is by associating it to transitions. This can be done as a *delay*, constraining the amount of time that elapses between the enabling of a transition and its instantaneous occurrence (assuming it is not disabled in the meanwhile by the occurrence of another — conflicting — transition), or as a *duration*, and then the occurrence is in *three phases*: start/activity/end. "True concurrency" leads to *temporal realism* of these models.

Different ways of constraining time lapses are:

- Giving a time interval, or window, as in *time PNs* [Mer74]. The interval may be just a point, and then timing is deterministic, as in *timed PNs* [Ram74].
- In a probabilistic fashion, giving the pdf, as in *stochastic PNs* [Mol82], [ABC84], [ABC$^+$95].
- In a possibilistic fashion, by way of fuzzy sets. In [CVD96] *fuzzy PNs* are overviewed. (In some cases not only the timing but also the marking is fuzzyfied.)

Similarly, different ways of constraining conflict resolution are:

- Giving a fairness constraint. The constraint may be rigid, and then it is deterministic.
- In a probabilistic fashion, as in stochastic PNs.
- In a possibilistic fashion, by way of fuzzy sets.

Defining a sound interpretation in order that the model reflects faithfully the intended behavior is not always an easy job. In the case of stochastic interpretations, [ABB$^+$89] explores different sound possibilities, and it shows that the net structure should be carefully taken into account.

Figure 4 shows a PN model under two different interpretations. The model is extremely simple, yet it reveals some paradoxical behaviors. If we associate a stochastic interpretation, e.g., firing durations are exponentially distributed with means $[s_1, 7, 0.1, 2, 1, 7]$, we can plot the cycle time versus s_1, the mean duration time of t_1 (what reveals that the system can be speeded up — the global cycle time decreases — by slowing down a subsystem, t_1 in this case). If we associate a deterministic duration interpretation, e.g., firing durations are $[1, 2, 1, 2, 1, 3]$, we can compare different scheduling policies by depicting the corresponding Gantt charts (what reveals a well-known phenomenon in optimization: the optimal global behavior may not be reached using local optimization rules, such as *immediate progress* — firing as soon as possible — in this case).

PN based formalisms can be related to other formalisms which are used for similar purposes. For instance,

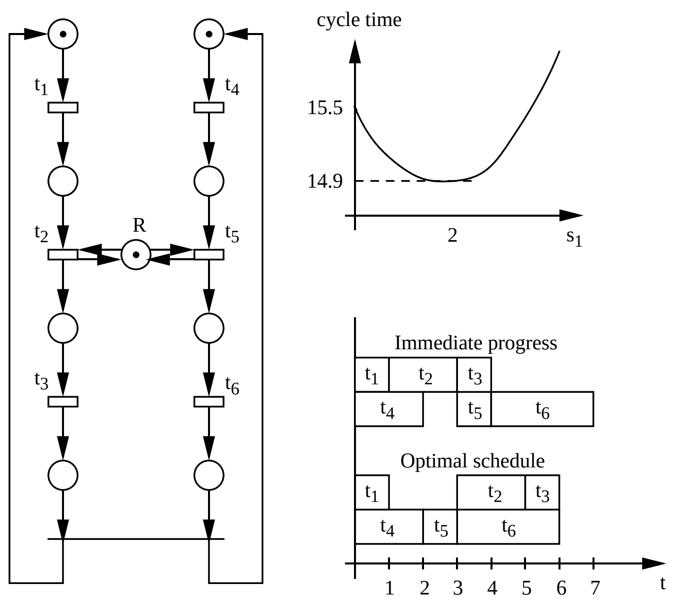

Fig. 4. Different interpretations of a PN. With an stochastic duration time interpretation it is illustrated that slowing down a subsystem may speed up the whole system. With a deterministic duration time interpretation it is illustrated that immediate progress scheduling may slow down the whole system.

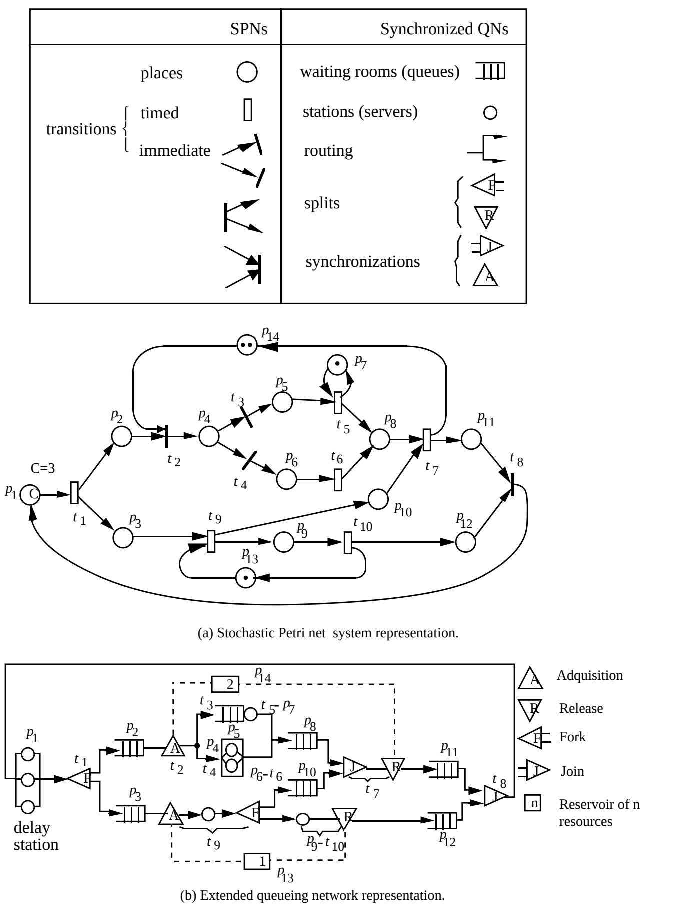

Fig. 5. Stochastic PN and extended queueing network representations of a customer/server system with passive resources and synchronizations.

stochastic PNs are used for performance evaluation, the same as (the diverse formalisms in the family of) queueing networks. With the presented interpretation, transitions clearly correspond to stations (self-concurrency corresponds to multiple servers), and queues are modeled by places. PNs provide a systematic way to introduce synchronization primitives (in this sense Figure 5 is self-explanatory). Moreover, since in a net model not all the places need to play the role of queues (they can model resources, or local states of a station which has been refined for a better description), it is possible to merge the customer/server and functional modeling perspectives. On the other hand, work needs to be done to fully incorporate *service*, *queueing*, and *routing* disciplines into the PN framework, where it is current practice to assume random policies.

As another example, (possibly stochastically) timed PNs can be used for the modeling and solution of scheduling problems. In this case, transitions can be classified as *controllable* or *uncontrollable*. The latter fire as soon as possible, while the former need to be scheduled, e.g., to obtain a good (ideally the best) performance. From this perspective, scheduling is a *performance control* activity. Compared to PERT graphs, which represent only *precedence* constraints between tasks, PNs also allow for the natural representation of *resource* constraints, generalizing conjunctive-disjunctive graphs [CC88], and allow the consideration of *cyclic* behaviors.

Besides the economy and coherence gained by the use of related formalisms, concepts developed for one particular purpose can become meaningful for others. We shall illustrate this phenomenon by two significative examples. In the analysis of timed systems it is usually found that activities take very different amounts of time to be completed, what leads to the appearance of states with very different steady state probabilities. States with negligible probabilities could be disregarded in many analyses to reduce the computational complexity. To facilitate this simplification, a time-scale notion was introduced in stochastic PNs by the definition of *immediate* transitions [ABC84], which have the additional important benefit of facilitating the description of the *routing*. The distinction of different time scales has been related to the distinction of observable and non-observable transitions in the framework of logic analysis [SA92], laying a bridge for the cooperation from both views of the problem.

As a second example, it is conventional to describe (wide range) quantized signals by continuous signals. A well-known example is the description of the populations in predator-prey systems (e.g., goats and wolves living in an island) by positive real functions over time as an approximation of the more realistic natural functions. A *fluidization* of this kind has been used in approximate performance analysis techniques but it is also interesting to extend the PN modeling paradigm to cope with some continuous and *hybrid* systems (see, for instance, [LAD91], [Ols93], [TK93], [DA94]).

Regarding the matter of standardization, there is still a wide diversity of timed/stochastic PN formalisms. Nevertheless the use of *generalized stochastic PNs* (GSPN) [ABC$^+$95] and their colored extension has made them a *de facto* standard or reference model for performance modeling and evaluation. For scheduling applications, in most cases (deterministically) timed PNs [Ram74] where all transitions are controllable are used.

B. Marking Diagrams

In order to use PNs in automation it is needed to connect the net model (acting as a controller) to the plant being controlled. This implies that the evolution shall be somehow governed by *inputs* and reflected by *outputs*. *State diagrams* [Boo67], [Lew85] have associated inputs and outputs. We call a PN model with a similar association of inputs and outputs a *marking diagram*, as a natural — although not generally used — name, provided that in PNs the state is called marking. Models of this kind can be found in [DB76], [Sil85], [MKMH86], [Sil89], [DHP$^+$93], [ZD93].

Inputs (either in the form of external events or logic conditions) are associated to transitions in the form of *guards*. They affect the evolution: a transition *must* occur whenever it is enabled and the corresponding guard is true (provided contingent conflicts are solved). Outputs or actions can be associated to both places and transitions. In the former case some action is produced while the place is (sufficiently) marked (e.g., while train in critical section, represented by a corresponding place marked, red light on). In the latter some signal is produced at the occurrence of the transition (e.g., start a timer, step a counter, etc.). Actions can be further conditioned by external conditions.

In local control, net conflicts are typically solved by the corresponding guards. Otherwise, especially in coordination level control, the occurrence of controllable transitions is decided by consulting some *external oracle* (e.g., a *knowledge-based* scheduler) [VCA$^+$88], [MMS$^+$89], [VC93].

Marking diagrams allow for concise and natural representation of concurrency and sequencing compared to state diagrams and relay ladder logic diagrams, respectively. PN based controllers are available [DB76], [MKMH86]. *Grafcet*, an International Standard since 1987, is another tool for the specification of logic controllers which is essentially a subclass of interpreted PNs [DA92], [Dav95].

C. Consequences for the Analysis

Interpretations *restrict* the behavior of the underlying autonomous model, so they must be taken into account for the analysis. On the one hand, this may become extremely complicated in some cases because the notion of state must be enlarged, e.g., time PNs [BD91]. On the other hand, performing analysis of the autonomous system is only conclusive for some particular properties and subclasses of interpreted systems [Sil85]. For instance, the autonomous system in Figure 6 (a) is not bounded unless the interpretation ensures that t' fires as often as t; the autonomous system in (b) deadlocks (fire t twice) unless the interpretation precises that the conflict is resolved in alternating fashion; in (c), if t takes always more time to fire than t' then the system will not return to the initial marking and t'' will die, although the autonomous model is live and reversible.

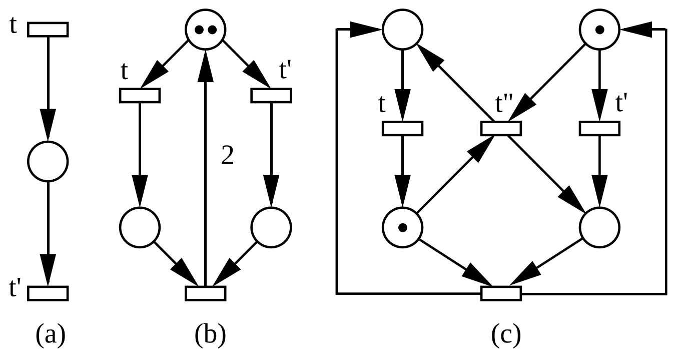

Fig. 6. The interpretation affects qualitative properties.

All in all, the shared structure allows to re-use *structural objects* and *relations* since the only modifications are in the occurrence rule. Therefore different PN based formalisms can be viewed as members of a *family* where the relationships lead to both economy and coherence. Although for each purpose or degree of detail the adequate formalism would be chosen from the family, the transformation from one formalism to another could be sound, if not formal or even automatic. The use of a single family of formalisms for such a diverse range of problems is not only beneficial from the point of view of communication and re-utilization of results. It has proven to lead also to a *synergic* situation where the concepts and techniques developed in one area help in the solution of open problems in another one [Sil93], [SC95]. For instance, the computability of the *visit ratios* (relative occurrence transitions) in stochastic net models opened the way to discover the so called *rank theorems* [CCS91], [TS96], which characterize in polynomial time important logical properties. As another example, symmetry detection at the logical level is a fundamental step towards efficient performance evaluation of stochastic colored PNs [CDFH93].

IV. On the Quality of Formalisms

In this section we reflect on some characteristics of formalisms, making reference to PNs. Our intention is not to be normative or classificatory, but rather to give some broad criteria for judgment.

A. On the Number of Concepts

Minimality in the number of primitives is usually a must in the construction of a conceptual framework. This raises a tradeoff between the engineering and scientific perspectives: while engineers appreciate a rich ontology with different concepts suited for different purposes, scientists look for basic and deep underlying notions. Take for instance the two models in Figure 5. The extended queueing network introduces a variety of specific primitives to handle synchronization and resource constraints, although some situations are quite similar, e.g., passive resources and customers reside in different kinds of nodes and different nodes are used to model a join and a resource acquisition. In the PN model the same phenomena are represented with a few basic primitives, e.g., a customer or a resource are tokens in places, so both joins or resource acquisitions are modeled in the same way.

The diversity and specificity of primitives may be convenient to develop concise and elegant models, but it tends to difficult formal reasoning and theory construction. An ideal solution to conciliate reasoning capabilities and practical expressivity consists on having a minimal number of basic primitives in terms of which richer derived primitives can be constructed. In this sense, the basic PN formalism is quite spare: only two simple and somehow orthogonal primitives are identified: one active (transitions) and one passive (places). These basic primitives are connected in alternation to form nets. The fact that the two primitives can be rated as active and passive does not condition the modeling of what could be identified at a higher level as active and passive *subsystems/components* (e.g., programs and data structures). Actually, these are represented in the same way: *by means of nets*. For example, in Figure 2 the machines are active components while the buffer is passive, but they are all represented as subnets even though they play very different roles. When richer primitives are required, high level PNs (Subsection II-C) can be used as an abbreviation of place/transition nets, and reversible transformations (folding/unfolding) exist to go from one to the other. As another example of abbreviation, in the case of bounded systems the use of extensions such as inhibitor arcs can simplify a plain place/transition model.

Up to this point, the number of primitives has been regarded mainly as a matter of modeling convenience. But this is not the only implication of the number of primitives. In some cases, having too few primitives prevents the representation of certain situations and features. This self-imposed limitation leads usually to stronger analysis results, at the price of loosing theoretical modeling power.

In spite of the reduced number and simplicity of primitives, place/transition systems are rather general, in the sense that they are able to describe complex and subtle phenomena, as it was illustrated in Subsections II-B and III-A. But there are systems that cannot be represented, or properties of systems that cannot be captured, what requires the introduction of (interpreted) extensions in some situations (Subsection II-D and Section III).

B. On Semantics

The semantics of a formalism must be clear and unambiguous. Operational semantics describe *how* the system works in a way close to a conceivable implementation. A semantics is denotational when it declares *what* is done, in terms of equations, expressions, etc. for instance in a logic specification. Denotational semantics are more abstract and so more independent of implementation.

PNs are typical operational formalisms. They describe the system in terms of (local) states, state changes (transitions), and the evolution rule. These objects are semantically rich in the sense that they allow to represent concurrency, synchronic relations, conflict, etc. as illustrated in Subsection II-B. In particular, the representation of concurrency by means of *causal independence* of transition occurrences is more faithful than in the case of interleaving semantics, where concurrency is expressed in terms of non-determinism. The separation of concurrency from non-determinism leads to *temporal realism* in timed interpretations, what is crucial for performance evaluation or scheduling.

Contemplating non-determinism is important in a semantics of a formalism for DEDS, specially in the case of parallel and distributed systems, because it recognizes the *lack of observability* of the global state of the system. In autonomous PNs non-determinism means completely abstracting from the scheduling of transition occurrences. In other words, it is not fixed how conflicts are solved (as in non-deterministic automata), even it is not fixed when enabled transitions occur. This abstraction of the scheduling gives freedom for the implementation (as it happens in Ada with the *select* statement).

Let us briefly comment on the semantics of the *logic* of enablement/occurrence of autonomous PNs. Many bridges have been laid between PNs and logic. For instance, first-order predicate logic was used as a model for developing a net theoretic formalism dealing with individuals (distinguishable tokens) and their properties and relations in predicate/transition nets [GL81]. In [MZ88], [PM89] predicate/transition nets are used for the formal modeling of some logic programs. But there is an essential difference between predicate logic and PNs due to the inability of classical logic to handle resources [VC93]. *Linear logic* [Gir87] has been proposed to fill this gap. In classical logic, by the *weakening axiom*, from 'A implies B' it can be deduced that 'A and C implies B'. So, if A, B, and C were interpreted as resources, and implication was interpreted as a resource transformation, then resources (of type C) might disappear without being used. However, in linear logic, from 'A implies B' it can only be deduced that 'A and C implies B and C'. Similarly, the *contraction axiom* (if 'A implies B' then 'A and A implies B') is not valid in linear logic. In [EW94] it is shown how linear logic may serve as a specification logic for PNs.

In [EW94] PNs are used also to clarify some notions of linear logic. In fact, the clear and intuitive operational semantics of PNs together with their generality have made them a kind of framework of reference for expressing the semantics of other formalisms (e.g., [Tau89]). In [Old91] three complementary formalisms are used to describe concurrent processes at different abstraction levels, and their semantic links are explored; logic formulas of temporal or ordinary predicate logic specify the bahavior, process terms are used as an abstract concurrent programming language that stresses compositionality, and place/transiton nets are chosen to operationally describe processes as concurrent and interacting machines.

C. On Locality and Structuration

When modeling large systems, composed by many different interconnected subsystems, it is essential that the model reflects such structure (e.g., the physical structure of a complex mechanical device or a manufacturing plant; notice that most often we deal with *man-made* systems, which have some kind of meaningful structure — perhaps not unique — from their design). Otherwise, modifying a subsystem or adding a new one may force to largely (sometimes completely) re-build the model, and it is not possible to re-use subsystem models in different situations. Complex systems typically have a long life cycle, so they require indeed frequent such modifications, additions, and re-uses.

The locality of places and transitions is central in PNs. It appears as the starting point for the *structuration* of net models. For instance, it is possible to *refine* a place or transition to give a more detailed description, or to *compose* two *modules* by identification of shared transitions or places. Refinements and modularity can be either based on states or actions, thanks to their treatment on equal footing.

It must be noted at this point that the basic PN formalism *does not force* to structure models. The fact that somehow PNs are *structure based* does not mean that they are *structured.* In other words, a PN model is a static structure (places, transitions, arcs) on which a behavior is modeled by "playing the token game". But appart from this structure (which is very important, e.g., for the analysis as we shall comment later) the net model is "flat", there is no *explicit* structuration in the form of modularity, hierarchy, etc., so it can grow disorderly.

Structuration can be achieved either by introducing construction operators (following the approach of process algebras [Mil89] and defining some algebraic structure [BK95]), or by following some pragmatical methodology (e.g., [VMS88], [CBG91], [ZD93] in the context of manufacturing systems). For systematic construction it is essential to investigate the fundamental notions of *composition* and *refinement*, both based on the locality principle. Moreover it is important to consider *hierarchical* and *layered* systems, where a collection of inter-related models are used to describe with different degrees of detail the same system or parts of it. This has been a major topic in PN research [BGV91], [BD92], [PRS92], [Feh93].

It is a common trend in systems theory to study restricted *subclasses* of systems or formalisms where the greater tractability is paid for by the lower expressive power. For instance, linear time-invariant dynamic systems are a well studied (and practically relevant) subclass of continuous systems. In the case of PNs, thanks to the clarity and depth of the basic notions, it has been possible to characterize specially tractable system behaviors in terms of simple local syntactical constraints, leading to a sort of *taxonomy* of systems. In several cases the studied subclasses allow for, or are meant to, systematic construction. Some well studied subclasses are marked graphs and other deterministic systems [BCOQ92], [TCS97], free choice models [CCS91], [DE95], [TS96], macroplace/macrotransition [DJS92], modules synchronized via rendez-vous [DDPS82], [Don94], or restrictedly sharing resources [ZD93], [ECM95], or cooperating via message passing [Sou93], [TSCC95].

D. On Methodological Support

Maturity of an engineering discipline requires not only formalisms, but also some associated *methodological support* (e.g., in terms of synthesis procedures, analysis techniques, tools, etc.) and the existence of *expertise* (e.g., in terms of catalogs of models and methods for specific domains, standards, etc.).

Figuratively, a formalism is a car that needs a driver (methods and tools) in order to successfully reach the destination (solving problems). For example, to obtain and solve differential equations models of complex continuous dynamic systems the aid of modeling methodologies (e.g., bond graphs [Tho90], system dynamics [Coy77], etc.) and supporting tools are essential. In this subsection we briefly comment on the support given by PNs for the modeling, analysis, implementation, and documentation of DEDS.

Model building is mainly a creative, thus difficult to automatize, task. Nevertheless for a given kind of systems and problems it may prove useful to somehow limit creativity, either to help in the automatic generation of the model from a domain-oriented description of the problem or to facilitate the subsequent analysis. The considerations we made regarding structuration in Subsection IV-C are relevant at this point. Additionally, when considering high level models, a methodology for the data representation, integrating modern software engineering concepts like *algebraic specification* and *object-orientation*, is required [BDM88], [BB91], [Sib94], [van94], [VM94].

The *analysis* of PN models has been largely investigated. Virtually all the already cited books and surveys on PNs cover analysis issues, both of autonomous or interpreted models. In fact, analyzability is frequently pointed out as one of the major comparative advantages of net models. Broadly speaking, (qualitative and quantitative) analysis methods of PN models can be classified as *behavioral*, *net-driven*, or *structural.* Behavioral methods are based on some description of the state space, typically in terms of a (partial) enumeration of the reachable states (e.g., reachability graph, underlying Markov chain, etc.). While they are often conclusive, they are computationally expensive or non feasible due to the *state space explosion problem.* To alleviate this problem, net-driven techniques have been proposed to reduce the number of states for a given analysis (e.g., taking advantage from symmetries, reducing the model at the net level, applying a "divide and conquer" approach identifying appropriate structural components, etc.). In some cases, specially when some net subclasses are considered, the analysis can be done reasoning only at the net level (i.e., structurally), typically by a combination of linear algebra/convex geometry and graph theory. Besides the eventual efficiency achieved by structural methods, they have the advantage of providing a deeper understanding and giving results which are valid for a class of models rather than a single one (because the initial marking is regarded as a *parameter*). As it was pointed out in Subsection III-C the coherence between formalisms for different purposes results in a synergic interleaving of concepts and analysis techniques [Sil93], [SC95].

The availability of analysis results allows the designer to apply a *try and error* design methodology: after the model is built, the analysis techniques are applied; if the results are not satisfactory then the model is changed and the cycle is repeated. An alternative approach is to build models which are *correct by construction*. To date this has been more successfully achieved for top-down methodologies ([Val79], [SM82], reversed reduction rules of [Ber86]). Correctness by construction is also one of the aims of the theory of *supervisory control*, a control theory for DEDS (see [RW89] for a tutorial survey). In [HKG95] a rather complete overview of the use of PNs for supervisory control is given. The expected benefit from the use of PNs comes from the clear and sound definition of the state space through a structure that reflects the locality principle and that can be exploited using (integer) linear algebraic or graph theoretic techniques.

When using nets for design purposes, once a suitable model has been obtained (e.g., the model for a controller, a model for simulation or prototyping) it has to be implemented. Basically an *implementation* is a device, usually a programmed computer system, which emulates the behavior expressed by the model. The implementation is affected by the selected formalism (low or high level, different interpretations of the firing rule), the algorithmic approach (interpreted, where the net model is a data structure, or compiled, where a program is obtained from the given net; centralized or parallel/distributed schemas), and the computer architecture (high or low level programming language; single or multi processor). It is quite apparent from the above that PNs, when used as a specification for implementation, leave significant freedom.

Regarding the support provided for *documentation* purposes, it is widely recognized by the clear and intuitive semantics together with the *graphical* representation of net models make them a valuable tool for the *dialog* between different people through different stages of the design and operation (even acting as a kind of "blue print" or contract), and for the visualization and animation in *monitoring*. (In large models, the graphical representation requires some kind of modularity and hierarchy to be manageable.)

Complete methodological support requires, in addition to conceptual developments, their integration, computer implementation, and some degree of standarization, possibly tailored for specific application domains. Regarding computer support, this is essential to aid in the construction of models (editing facilities, syntax checks, libraries, etc.), their analysis (the designer is not required to know in detail analysis techniques, and they are reliably and efficiently applied; quick interaction helps in the understanding and shortens the time lapses in try and error iterations), implementation (automatic code generation, prototyping, etc.), and operation (e.g., in computer integrated manufacturing). In this respect, much work needs to be done yet to meet industrial requirements, although significant achievements are available (a good pointer is `http://www.daimi.aau.dk/PetriNets/`, the Web page on PNs maintained by DAIMI, Aarhus University).

Significant expertise exists in the application of PNs to diverse fields, in particular to the design and operation of manufacturing systems. Some recent survey or tutorial publications on the topic are [SV89], [DA92], [VN92], [DHP+93], [DAJ94], [ZZ94], [ST96].

To conclude, we believe that PNs are an adequate conceptual framework or paradigm for the operational description of DEDS. Nevertheless, we do not believe that it is always possible to select a single formalism, or family of them, to deal in a reasonable way with every aspect of every DEDS. The complexity and variety of systems suggest instead the interest of having multi-paradigm environments, where the existence of sound and efficient bridges between different paradigms becomes a major issue.

Acknowledgement

We are grateful to Javier Aracil from the University of Sevilla and to our colleagues from the University of Zaragoza Javier Campos, José Manuel Colom, and Laura Recalde, who made very helpful comments and suggestions after reading earlier versions of this paper.

References

[ABB+89] M. Ajmone-Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Trans. on Software Engineering*, 15(7):832–846, 1989.

[ABC84] M. Ajmone-Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Trans. on Computer Systems*, 2(2):93–122, 1984.

[ABC+95] M. Ajmone-Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley, 1995.

[AF73] T. Agerwala and M. Flynn. Comments on capabilities, limitations and "correctness" of Petri nets. *Computer Architecture News*, 2(4), 1973.

[Ajm93] M. Ajmone Marsan, editor. *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*. Springer, 1993.

[AK77] T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3:85–104, 1977.

[BB91] M. Baldassari and G. Bruno. Protob: An object oriented methodology for developing discrete event dynamic systems. *Computer Languages*, 16(1):39–63, 1991.

[BCMP75] F. Baskett, K. M. Chandy, R. R. Muntz, and F. Palacios. Open, closed and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, 1975.

[BCOQ92] F. Baccelli, G. Cohen, G. J. Olsder, and J. P. Quadrat. *Synchronization and Linearity*. Wiley, 1992.

[BD91] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Software Engineering*, 17(3):259–273, 1991.

[BD92] L. Bernardinello and F. DeCindio. A survey of basic net models and modular net classes. In Rozenberg [Roz92], pages 304–351.

[BDM88] E. Battiston, F. DeCindio, and G. Mauri. OBJSA nets: A class of high-level Petri nets having objects as domains. In Rozenberg [Roz88], pages 20–43.

[Ber86] G. Berthelot. Checking properties of nets using transformations. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 1986.

[BGV91] W. Brauer, R. Gold, and W. Vogler. A survey of behaviour and equivalence preserving refinements of Petri nets. In Rozenberg [Roz91], pages 1–46.

[BH95] J. P. Bowen and M. G. Hinchey. Seven more myths of formal methods. *IEEE Software*, 12(4):34–41, 1995.

[BK92] E. Best and M. Koutny. Petri net semantics of priority systems. *Theoretical Computer Science*, 96:175–215, 1992.

[BK95] E. Best and M. Koutny. A refined view of the box algebra. In De Michelis and Diaz [DD95], pages 1–20.

[Boo67] T. L. Booth. *Sequential Machines and Automata Theory*. Wiley, 1967.

[BRA83] G. W. BRAMS. *Réseaux de Petri: Théorie et Pratique*. Masson, 1983.

[BRR87] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Central Models and their Properties. Advances in Petri Nets 1986, Part I*, volume 254 of *Lecture Notes in Computer Science*. Springer, 1987.

[CBG91] D. Cruette, J. P. Bourey, and J. C. Gentina. Hierarchical specification and validation of operating sequences in the context of FMSs. *Computer-Integrated Manufacturing*, 4(3):140–155, 1991.

[CC88] J. Carlier and P. Chretienne. *Problèmes d'Ordonnancement. Modélisation, complexité et algorithmes*. Masson, 1988.

[CCS91] J. Campos, G. Chiola, and M. Silva. Properties and performance bounds for closed free choice synchronized monoclass queueing networks. *IEEE Trans. on Automatic Control*, 36(12):1368–1382, 1991.

[CDFH93] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Trans. on Computers*, 42(11), 1993.

[CH93] S. Christensen and N. D. Hansen. Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs. In Ajmone Marsan [Ajm93], pages 186–205.

[Cia94] G. Ciardo. Petri nets with marking-dependent arc cardinality: Properties and analysis. In Valette [Val94], pages 179–198.

[Coy77] R. G. Coyle. *Management System Dynamics*. Wiley, 1977.

[CVD96] J. Cardoso, R. Valette, and D. Dubois. Fuzzy Petri nets: An overview. In 13^{th} *IFAC World Congress*, San Francisco, CA, USA, July 1996. To appear.

[DA92] R. David and H. Alla. *Petri Nets and Grafcet*. Prentice-Hall, 1992.

[DA94] R. David and H. Alla. Petri nets for modeling of dynamic systems — a survey. *Automatica*, 30(2):175–202, 1994.

[DAJ94] A. Desrochers and R. Y. Al-Jaar. *Applications of Petri Nets in Manufacturing Systems*. IEEE Press, 1994.

[Dav95] R. David. Grafcet: A powerful tool for specification of logic controllers. *IEEE Trans. on Control Systems Technology*, 3(3):253–268, 1995.

[DB76] E. Daclin and M. Blanchard. *Synthèse des Systémes Logiques*. Cepadues, 1976.

[DD95] G. De Michelis and M. Diaz, editors. *Application and Theory of Petri Nets 1995*, volume 935 of *Lecture Notes in Computer Science*. Springer, 1995.

[DDPS82] F. DeCindio, G. DeMichelis, L. Pomello, and C. Simone. Superposed automata nets. In Girault and Reisig [GR82].

[DE95] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.

[Des89] A. Desrochers, editor. *Modeling and Control of Automated Manufacturing Systems*. IEEE Computer Society Press, 1989.

[DHP$^+$93] F. Dicesare, G. Harhalakis, J. M. Proth, M. Silva, and F. B. Vernadat. *Practice of Petri Nets in Manufacturing*. Chapman & Hall, 1993.

[DJS92] A. Desrochers, H. Jungnitz, and M. Silva. An approximation method for the performance analysis of manufacturing systems based on GSPNs. In *Procs.* 3^{rd} *Int. Conf. on Computer Integrated Manufacturing and Automation Technology (CIMAT '92)*, pages 46–55. IEEE Computer Society Press, 1992.

[Don94] S. Donatelli. Superposed generalized stochastic Petri nets: Definition and efficient solution. In Valette [Val94], pages 258–277.

[ECM95] J. Ezpeleta, J. M. Colom, and J. Martínez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation*, 11(2):173–184, 1995.

[EW94] U. Engberg and G. Winskel. Linear logic on Petri nets. In *A Decade of Concurrency. Reflections and Perspectives*, volume 803 of *Lecture Notes in Computer Science*, pages 176–229. Springer, 1994.

[Feh93] R. Fehling. A concept of hierarchical Petri nets with building blocks. In Rozenberg [Roz93], pages 148–168.

[Fre82] S. French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Ellis-Horwood, 1982.

[Gir87] J. Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[GL81] H. J. Genrich and K. Lautenbach. System modeling with high level Petri nets. *Theoretical Computer Science*, 13:109–136, 1981.

[GR82] C. Girault and W. Reisig, editors. *Application and Theory of Petri Nets*. Springer, 1982.

[Hac75] M. H. T. Hack. *Decidability Questions for Petri Nets*. PhD thesis, M.I.T., Cambridge, MA, USA, December 1975. Also Tech. Report 161, Lab. for Computer Science, June 1976.

[Hal90] J. A. Hall. Seven myths of formal methods. *IEEE Software*, 7(5):11–19, 1990.

[HC70] A. W. Holt and F. Commoner. Events and conditions. *Applied Data Research*, 1970.

[Her91] U. Herzog. Performance evaluation and formal description. In *Proc. IEEE Conf. CompEuro 91*, pages 750–756, Bologna, Italy, 1991.

[HKG95] L. E. Holloway, B. H. Krogh, and A. Giua. Petri nets for the control of discrete event systems: A tutorial survey. In *Supervisory Control of Discrete Event Systems*. Laboratoire d'Automatique de Grenoble, INPG, September 1995.

[Ho89] L. Ho, editor. Special issue on discrete event systems. *Proceedings of the IEEE*, 77(1), 1989.

[ISO88] LOTOS: A formal description technique based on the temporal ordering of observational behaviour. Technical Report DIS 8807, I.S.O. — Information Processing Systems — Open Systems Interconnection, 1988.

[Jen91] K. Jensen. Coloured Petri nets: A high level language for system design and analysis. In Rozenberg [Roz91], pages 342–416. Collected in [JR91].

[JR91] K. Jensen and G. Rozenberg, editors. *High-level Petri Nets*. Springer, 1991.

[Kan92] K. Kant. *Introduction to Computer System Performance Evaluation*. McGraw-Hill, 1992.

[KM69] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal on Computer Systems Science*, 3:147–195, 1969.

[LAD91] J. Le Bail, H. Alla, and R. David. Hybrid Petri nets. In *European Control Conference (ECC '91)*, pages 1472–1477, Grenoble, France, 1991.

[LC94] C. Lakos and S. Christensen. A general systematic approach to arc extensions for coloured Petri nets. In Valette [Val94], pages 338–357.

[Lew85] D. Lewin. *Design of Logic Systems*. Van Nostrand Reinhold, 1985.

[Mer74] P. Merlin. *A study of the Recoverability of Computer Systems*. PhD thesis, Univ. California, Irvine, CA, USA, 1974.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[MKMH86] T. Murata, N. Komoda, K. Matsumoto, and K. Haruna. A Petri net based controller for flexible and maintainable sequence control and its applications in factory automation. *IEEE Trans. on Industrial Electronics*, 33(1):1–8, 1986. Reprinted in [Des89].

[MMS$^+$89] J. Martínez, P. Muro, M. Silva, S. F. Smith, and J. L. Villarroel. Merging artificial intelligence techniques and Petri nets for real time scheduling and control of production systems. In R. Huber et al., editors, *Artificial Intelligence in Scientific Computation*, pages 307–313. Scientific Publishing Co., 1989.

[Mol82] M. K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Trans. on Computers*, 31(9):913–917, 1982.

[MR95] U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32:545–596, 1995.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[MZ88] T. Murata and D. Zhang. A predicate-transition net model for parallel interpretation of logic programs. *IEEE Trans. on Software Engineering*, 14(4):481–497, 1988.

[Old91] E. R. Olderog. *Nets, Terms and Formulas*, volume 23 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1991.

[Ols93] G. J. Olsder. Synchronized continuous flow systems. In S. Balemi, P. Kozak, and R. Smedinga, editors, *Discrete Event Systems: Modeling and Control*, pages 113–124. Birkhauser, 1993.

[Pet62] C. A. Petri. *Kommunication mit Automaten.* PhD thesis, Institut für Instrumentelle Mathematik, Univ. Bonn, 1962.

[Pet81] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, 1981.

[Pet87] C. A. Petri. Concurrency theory. In Brauer et al. [BRR87], pages 4–24.

[PM89] G. Peterka and T. Murata. Proof procedure and answer extraction in Petri net model of logic programs. *IEEE Trans. on Software Engineering*, 15(2):209–217, 1989.

[PRS92] L. Pomello, G. Rozenberg, and C. Simone. A survey of equivalence notions for net based systems. In Rozenberg [Roz92], pages 410–472.

[Ram74] C. Ramchandani. Analysis of asynchronous concurrent systems by Petri nets. Technical Report Project MAC, TR-120, M.I.T., Cambridge, MA, USA, 1974.

[Rei85] W. Reisig. *Petri Nets. An Introduction.* EATCS Monographs on Theoretical Computer Science. Springer, 1985.

[Roz88] G. Rozenberg, editor. *Advances in Petri Nets 1988*, volume 340 of *Lecture Notes in Computer Science*. Springer, 1988.

[Roz91] G. Rozenberg, editor. *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*. Springer, 1991.

[Roz92] G. Rozenberg, editor. *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*. Springer, 1992.

[Roz93] G. Rozenberg, editor. *Advances in Petri Nets 1993*, volume 674 of *Lecture Notes in Computer Science*. Springer, 1993.

[RW89] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. In *Proceedings of the IEEE* [Ho89], pages 81–98.

[SA92] C. Simone and M. Ajmone-Marsan. The application of EB-equivalence rules to the structural reduction of GSPN models. *Journal of Parallel and Distributed Computing*, 15(3):296–302, 1992.

[SC95] M. Silva and J. Campos. Structural performance analysis of stochastic Petri nets. In *IEEE IPDS '95*, pages 61–70. IEEE Computer Society Press, 1995.

[Sib94] C. Sibertin-Blanc. Cooperative nets. In Valette [Val94], pages 377–396.

[Sil85] M. Silva. *Las Redes de Petri: en la Automática y la Informática.* AC, 1985.

[Sil89] M. Silva. Logic controllers. In *IFAC Symposium on Low Cost Automation (vol. II)*, pages 157–166, Milano, Italy, November 1989.

[Sil93] M. Silva. Interleaving functional and performance structural analysis of net models. In Ajmone Marsan [Ajm93], pages 17–23.

[SM82] I. Suzuki and T. Murata. Stepwise refinement of transitions and places. In Girault and Reisig [GR82], pages 136–141.

[Sou93] M. Y. Souissi. Deterministic systems of sequential processes: A class of structured Petri nets. In Rozenberg [Roz93], pages 406–426.

[ST96] M. Silva and E. Teruel. Petri nets for the design and operation of manufacturing systems. In *Procs. 5th Int. Conf. on Computer Integrated Manufacturing and Automation Technology (CIMAT '96).* IEEE-Computer Society Press, 1996.

[SV89] M. Silva and R. Valette. Petri nets and flexible manufacturing. In G. Rozenberg, editor, *Advances in Petri Nets 1989*, volume 424 of *Lecture Notes in Computer Science*, pages 374–417. Springer, 1989.

[Tau89] D. Taubner. *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets*, volume 369 of *Lecture Notes in Computer Science.* Springer, 1989.

[TCS97] E. Teruel, J. M. Colom, and M. Silva. Choice-free Petri nets: A model for deterministic concurrent systems with bulk services and arrivals. *IEEE Trans. on Systems, Man, and Cybernetics*, 1997. To appear.

[Tho90] J. U. Thoma. *Simulation by Bondgraphs. Introduction to a Graphical Method.* Springer, 1990.

[TK93] K. Trivedi and V. G. Kulkarni. FSPNs: Fluid stochastic Petri nets. In Ajmone Marsan [Ajm93], pages 24–31.

[TS96] E. Teruel and M. Silva. Structure theory of equal conflict systems. *Theoretical Computer Science*, 153(1-2):271–300, 1996.

[TSCC95] E. Teruel, M. Silva, J. M. Colom, and J. Campos. Functional and performance analysis of cooperating sequential processes. In F. Baccelli, A. Jean-Marie, and I. Mitrani, editors, *Quantitative Methods in Parallel Systems*, pages 52–65. Springer, 1995.

[Val78] R. Valk. On the computational power of extended Petri nets. In *Proc. 7th Sypm.Mathematical foundations of Computer Science*, volume 64 of *Lecture Notes in Computer Science*, pages 527–535. Springer, 1978.

[Val79] R. Valette. Analysis of Petri nets by stepwise refinements. *Journal of Computer and System Sciences*, 18(1):35–46, 1979.

[Val94] R. Valette, editor. *Application and Theory of Petri Nets 1994*, volume 815 of *Lecture Notes in Computer Science.* Springer, 1994.

[van94] K. M. van Hee. *Information Systems Engineering: A Formal Approach.* Cambridge University Press, 1994.

[VC93] R. Valette and M. Courvoisier. Petri nets and artificial intelligence. In R. Zurawski and T. Dillon, editors, *Modern Tools for Manufacturing Systems*, pages 385–405. Elsevier, 1993.

[VCA$^+$88] R. Valette, J. Cardoso, H. Atabakhche, M. Courvoisier, and T. Lemaire. Petri nets and production rules for decision levels in FMS control. In *Procs. IMACS 1988, 12th World Congress on Scientific Computation*, pages 522–524, 1988.

[VM94] J. L. Villarroel and P. Muro. Using Petri net models at the coordination level for manufacturing systems control. *Robotics and Computer-Integrated Manufacturing*, 11(1):41–50, 1994.

[VMS88] J. L. Villarroel, J. Martínez, and M. Silva. GRAMAN: A graphic system for manufacturing system design. In S. Tzafestas, A. Eisinberg, and L. Carotenuto, editors, *IMACS Symp. on System Modelling and Simulation*, pages 311–316. Elsevier, 1988.

[VN92] N. Viswanadham and Y. Narahari. *Performance Modeling of Automated Manufacturing Systems.* Prentice-Hall, 1992.

[ZD93] M. C. Zhou and F. DiCesare. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems.* Kluwer Academic Publishers, 1993.

[ZZ94] R. Zurawski and M. C. Zhou, editors. Special issue on Petri nets in manufacturing. *IEEE Trans. on Industrial Electronics*, 41(6), 1994.