DEDS Along Their Life-Cycle: Interpreted Extensions of Petri Nets

Manuel Silva and Enrique Teruel

Abstract—
The diversity of interpreted Petri net (PN) formalisms, suited to deal with diverse purposes but sharing basic common principles, turns PN into a conceptual framework or paradigm for the modeling of DEDS. They can be used for the modeling, analysis, implementation, and control of these systems, sometimes with advantage, and with the additional benefit of improving the communication between stages of the life cycle. The utilization of Petri nets in several of these stages is illustrated in the paper.

I. Introduction

The development of computer-based technologies, e.g., in automation, communications, etc. makes for the growing importance of discrete event dynamic systems (DEDS). In order to cope with rapid technological and market changes it is essential that suitable techniques and methods for the design and operation of complex and changing DEDS are available. *Formal methods* may be helpful in this respect. They can provide a better understanding of the system, what helps in removing incompleteness and contradictions, identifying properties, or discovering potential solutions. At the same time, they can support the development helping to detect errors in early stages, giving assessment on the dimensioning of the system, guiding, or even automating, the implementation and documentation, etc.

A first step in the application of formal methods is to obtain *formal models* of our systems, typically based on mathematical theories. The conceptual framework that allows one to obtain a kind of formal models of systems is called a *formalism*. Different formalisms for DEDS are being proposed and experienced, and their links and relative merits are being investigated. Some well-known examples are *state diagrams* for functional description, *Markov chains* and *queuing networks* for performance evaluation, *PERT graphs* and *conjunctive/disjunctive graphs* for scheduling, etc.

This diversity arises naturally from the diversity of both the kind of DEDS we are interested in and the kind of properties or purposes we have in mind; DEDS have so many facets that it is expected that they are approached from different angles, providing *complementary* views of systems serving diverse purposes. But, at the same time, 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, makes desirable to have a *family of formalisms* rather than a collection of unrelated or weakly related formalisms. For us, a *modeling paradigm* is a conceptual framework that allows one to obtain formalisms from some common concepts and principles, with the consequent economy and coherence, among other benefits. So, we have formal models of precise systems which are expressed within a given modeling formalism which is the appropriate one within a modeling paradigm. Here, Petri nets (PN) are seen as a modeling paradigm for DEDS: Many formalisms for DEDS, in particular all those mentioned above, can be viewed as PN extended through appropriate interpretations.

The structure of the paper is as follows: Section II introduces the idea of interpreted extensions applied to directed graphs, making emphasis on formalisms for DEDS. In Sections III and IV Petri nets are introduced from state equation concepts for DEDS, and they are viewed as interpreted graphs. Some interpreted extensions of Petri nets, tailored for different phases of the life-cycle of a system and different purposes, are described in Section V. Finally, Section VI remarks that the different formalisms constitute a paradigm for DEDS, where more interpreted extensions have been defined in the literature, leading to a synergy in the development of the theory for all these different formalisms.

II. Some DEDS formalisms as interpreted graphs

(Valued) Binary relations on finite sets (sites, states,...) can be represented by *(valued) directed graphs*. The graphical representations are built using

The authors are with the Departamento de Informática e Ingeniería de Sistemas, Centro Politécnico Superior, Universidad de Zaragoza, María de Luna 3, E-50015 Zaragoza, Spain. E-mail: {silva,eteruel}@posta.unizar.es .

nodes (which represent entities) and arcs (which depict the relations). In the sequel only directed graphs, such as the one shown in Figure 1, will be considered (undirected graphs are an abbreviation when the relation is symmetric). Two classic matrix representations of a bi-

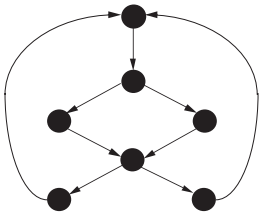

Fig. 1. A directed graph.

nary relation (or its graph) are the *adjacency* (or node to node) and the *incidence* (or node to arc) matrices.

Graphs have been extensively used in an extremely large quantity of applications domains for modeling and problem solving (e.g., electric circuits, power transport and telecommunication networks, layout problems, ...). The connection of a formalism and (some piece of) reality is provided by the *interpretation*. In a totally uninterpreted theory there is no meaning associated with 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.

When the graph-based model describes a dynamic system, the original non-determinism (absence of information about the precise behavior) is *reduced* through an *extension of the interpretation* in which a specification is given for the relationships among the model and its *environment* (defining input and output signals; introducing some explicit notion of time; ...). This allows to precisely characterize *when* and *how* the system evolves. The non-determinism reduction can lead, in extreme cases, to deterministic systems (as in deterministic flow graphs, or in deterministically timed marked graphs). In other cases, some non-determinism remains in the behavior of the model (e.g., when a routing of parts or a mix of production is specified probabilistically).

Along the life-cycle of a system, many formalisms can be used, in order to build models, each one dealing with some different analysis or synthesis purposes. Among many other classical formalisms are:

- *State Diagrams (SD)* [1], [2], that describe the functional behavior of sequentialized switching systems in relation with some environment, through input and output signals;
- *Continuous Time Markov Chains (CTMC)*, and *Queueing Networks (QN)* [3],[4] for performance evaluation;
- *Program Evaluation and Review Technique (PERT)* graphs and *conjunctive/disjunctive graphs* [5], [6] for project scheduling.

All the above formalisms can be "viewed" as graphs provided with appropriate interpretations. Of course, things can be viewed just the opposite way: Formalisms (SD, CTMC, QN, PERT,...) can be provided with graph representations. We will take here the first approach, trying to stress the existence of some "driving forces" in the process, and observing that the logic of the underlying interpretations can be quite different. We will restrict in the sequel to formalisms tailored to describe DEDS. In modeling DEDS it is frequently convenient to give some explicit representation to its state.

The first two interpretations we consider, namely SD and CTMC, provide a (directed) graph (e.g., the one in Figure 1) with two different interpretations sharing, nevertheless, the fact that a *global* state variable takes as many values as nodes in the graph.

In SD, *input signals* allow to define events and logical conditions in order to drive the evolution of the model. Time is not explicitely defined. Moreover, the model is equiped with *output signals* that are expected to influence the behavior of its environment: the model and the environment are in closed-loop (feedback). In the example of Figure 2, outputs are inconditionnally asigned to states (Moore-finite automaton). The model represents a go and back controller for the two wagons system: if they are initially over A and C, and M is pushed, both wagons go rigth till B and D, respectively. When both are at the extreme right, they start simultaneously the way back till A and C.

In a CTMC, such as that in Figure 3, nodes are global states (the same as before), but no input or

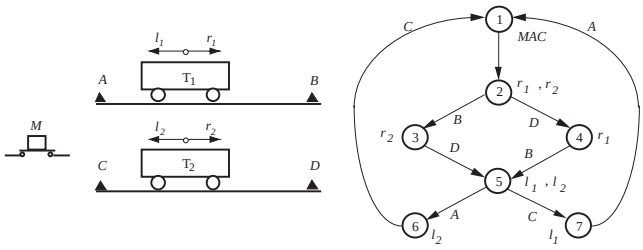

Fig. 2. A state diagram.

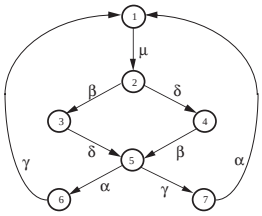

Fig. 3. A continuous-time Markov chain.

output signals are considered. The evolution of the model is now directed by a time-driven specification of events, through some rates (e.g., α represent the rate of occurence of A, what in average will last for $1/\alpha$ time units, provided negative-exponential pdf's). This model is not in closed-loop controlling a plant, but it is a time-driven model of the expected behavior. With it we can compute performance evaluation figures (for example, the average cycle time is:

$$\frac{1}{\mu}+\frac{1}{\beta}+\frac{1}{\delta}-\frac{1}{\beta+\delta}+\frac{1}{\alpha}+\frac{1}{\gamma}-\frac{1}{\alpha+\gamma}$$

time units; the probability vector of states, Π, can be computed as

$$\begin{aligned}\Pi\cdot Q &= \mathbf{0}\\ \Pi\cdot\mathbf{1} &= 1\end{aligned}$$

where Q is the classical *infinitesimal generator matrix* associated to the CTMC).

The same graph in Figure 1 can be interpreted in many other ways. In Figure 4 it is interpreted as a Gordon-Newell QN: nodes are viewed as *stations* provided with their input queues. If service durations are defined by means of negative exponential pdf's, the state is not defined by a global state variable (as in CTMC), but by the collection of the local states in queues, represented by the existing number of customers in each. The model evolves according to a customer/server perspective, following a production/consumption logic in which choices (routing) and

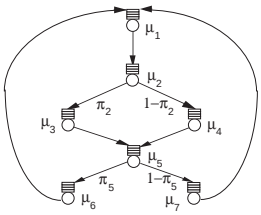

Fig. 4. A Gordon-Newell QN.

attributions (merging of flows) are based on an OR logic (e.g., from station 2 customers go to station 3 OR to station 4; customers arrive to station 1 from stations 6 OR 7). A consequence of this OR/OR logic in Gordon-Newell QN is that customers identity is preserved, while assembly (rendez-vous like) operations cannot be expressed. Routing is probabilistically specified (e.g., going from station 2 to station 3 is done with probability π_2).

In Figure 5 the graph in Figure 1 is now interpreted

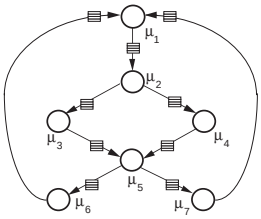

Fig. 5. A Fork/Join queueing network with blocking.

as a Fork/Join QN with Blocking. It shares with the previous queueing interpretation the customer/server view and a distributed state. Nevertheless, now:

- the nodes represent the servers, the arcs support the idea of queues;
- under negative exponential pdf in servers, the state is defined also by means of the state of queues (what means that the state is now defined by some arc attributes, not by node attributes);
- the model is choice and attribution-free.

Two arcs (queues) arriving at a station mean that two customers should synchronize (rendez-vous; assembly) in order to proceed to the service. Analogously, when two arcs leave a station, a disassembly operation takes place giving different disaggregated customers (and resources) to different queues. In other words, now the logic is also a production/consumption logic but of the AND type. With the above AND/AND logic customers are created and destroyed (what is the case in assembly/disassembly systems), but the formalism is decision-free.

Many other different interpretations can be given to graphs. For example, in PERTs also an AND/AND logic is provided, together with a different time interpretation (concerning now the duration of activities). A PERT represents the precedence relations (arcs) between tasks (nodes). PERTs are essentially *action oriented* formalisms, while SD or CTMC are *state oriented* formalisms. More complex interpretations are obtained when several kind of nodes or arcs are taken into consideration. For example, *Conjunctive/Disjunctive* graphs generalize PERTs adding a new class of undirected arcs. The activities represented by the nodes related by this new class of arcs are in mutual exclusion.

Petri nets, as it will be presented next, provide a (valued) bipartite-graph based framework for modeling DEDS in which all the above underlying logics for the behaviour of DEDS are generalized in a straigthforward way, leading to formalisms with more (theoretical and pragmatical) descriptive power. Petri nets behave according to a rather general production/consumption logic. They consider both states and actions on equal footing, and allow to combine both OR and AND logics under arbitrary interleavings.

III. Autonomous PN 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 and constraints from the environment. 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 usually evolves simply due to time passing.

DEDS 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, such as that of SD and CTMC, is useful in some applications, specially 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.). 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 — even they might be chosen to be binary, whenever the number of states is finite. 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. (The actual selection of variables is a crucial modeling task usually requiring knowledge of the domain, ingenuity, and methodology, but here we shall not consider this matter further.)

In a DEDS state changes at discrete points in time, driven by the occurrence of (external or internal) 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 global 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., if 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 with 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* [7], 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 in $\mathbf{C}$) 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* [8], [9], [10], [11], [12], [13], [14]. 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.

Petri nets are a family of formalisms rather than a single one. Dealing with the logic behavior alone, many extensions (e.g., nets with inhibitor arcs, self-modifying nets, etc.) and abbreviations (e.g., colored nets) of the place/transition formalism have been proposed. Since for our present purpose the place/transition is more than enough, we refer the interested reader to [15] where some reflections on and pointers to such extensions can be found.

The above introduction to net systems is intended to reflect their coherence with (discrete time) continuous systems (a more detailed presentation, making explicit this coherence, can be found in [15]). There are diverse alternative forms for approaching net systems, often computer science oriented. The seminal work is Petri's dissertation [16]. The concern of Carl Adam Petri are organizational problems, of importance to broad classes of phenomena. The relevance of net theory is recognized in systems theory [17] going beyond the limits of particular application domains. In fact, Petri declares the attempt to provide a common basis for physical and computational ways of thinking [18]. Quoting from [19]: "nets, just like groups, or vector spaces, or graphs, have many different practical applications, which are based on just as many different interpretations. This is so because they all are fairly general mathematical constructs, which were built and are studied for practical purposes." His axiomatic view is based on the fundamental notion of *concurrency*, the

"binary relation of contemporality of world points". He then proceeds through *occurrence nets*, which represent pieces of the history of interactions within a system. *Condition/Event systems* are obtained by a *morphism* that folds occurrence nets, mapping successive occurrences of states or transitions into single conditions or events, respectively. This basic model was generalized or adapted by other researchers. Historical remarks can be found in [10], [14], [20]. A completely different approach aims to obtain a net model starting from a finite automata based description of the system. This *synthesis approach* introduces nets as compact representations or realizations of systems. The seminal work in this direction is [21], and recent achievements are surveyed in [22].

IV. The graph view of autonomous PN

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.

As it was shown in Section II, a current, often convenient, technique to represent interrelations is by use of diagrams, or graphs. 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). 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 in 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).

Figure 6 shows a sample place/transition system. It

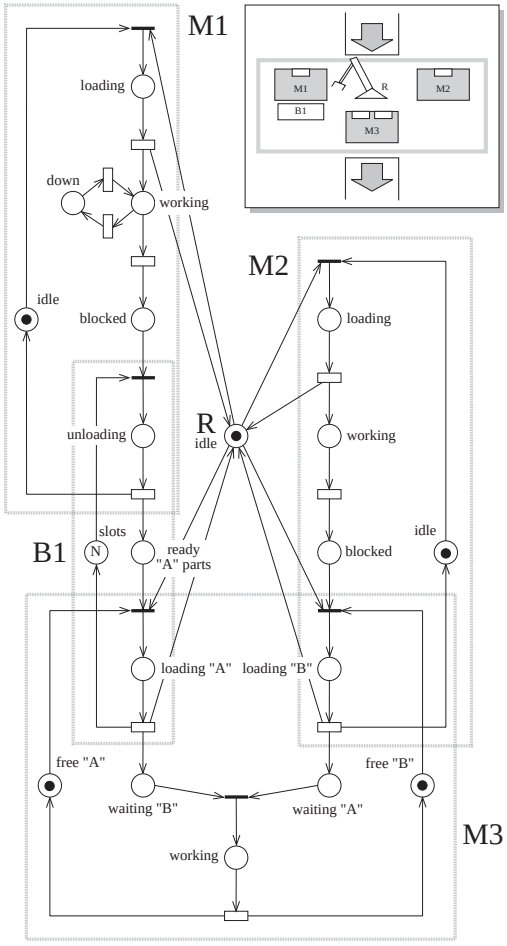

Fig. 6. An autonomous place/transition system that formally describes the logic behavior of a manufacturing cell.

models a manufacturing cell — see the layout in the top-right corner — that is composed by three machines (*M1*, *M2*, and *M3*). The work plan is as follows: Raw

parts arrive through a conveyor; A raw part is processed by *M1* to obtain a part of type "A", or by *M2* to obtain a part of type "B"; In *M3* two parts, one of each type, are assembled to obtain a final product, that leaves the cell; We assume saturation (i.e., the cell is never starved or blocked); Parts are handled by a robot (R). We assume that only *M1* may fail (operation dependent failures). To reduce the effect of *M1* failures, it deposits the "A" parts in a temporary buffer (*B1*, capacity N), without using R for this movement.

The net in Figure 6 models both the plant and the work plan, from a coordination viewpoint. In the initial state, all machines and the robot are idle, and the buffer is empty. The only enabled transitions are those that represent the start of the loading operation of either *M1* or *M2*, but only one of them can occur (i.e., there is a *conflict* situation). The autonomous model leaves undetermined which will occur, it only states that these are the possibilities. Assume *M1* is to be loaded, what is represented by the occurrence of the corresponding transition. Then the marking changes: one token is removed from each input place of the transition (*R idle* and *M1 idle*) and one token is put in the output place (*M1 loading*). Notice that tokens were required from two input places, meaning that the loading operation requires that both the machine and the robot are ready: it is a *synchronization* of both. Now the only enabled transition is the one representing the end of the loading operation, but the autonomous model leaves undetermined when will this happen, it only states that it can only happen whenever loading is in course (what allows representing *sequencing*). At the firing, the token is removed from *M1 loading* and tokens are put in *M1 working* and *Ridle*. In this new marking, both output transitions of *M1 working* are enabled in conflict (it may either complete the work or fail), and also the start of the loading of *M2* is enabled. This latter transition and a transition from *M1* can occur simultaneously, or in whichever order (their enabling is independent), what allows to faithfully model *concurrency*. Notice the correspondence of subnets and subsystems (*M1*, *M2*, *M3*, *B1*, and R), and the natural representation of their mutual interactions. (It goes without saying that operation places could be refined to show the detailed sequence of operations in each machine, etc.)

This autonomous model can be used for documentation/understanding purposes, and also to formally analyze the possible behaviors. Classical PN analysis techniques allow to efficiently decide that this system model is bounded (i.e., finite state space) live (i.e., no action can become unattainable), and reversible (i.e., from any state the system can evolve to its initial state).

We have depicted as bars those transitions that represent control events, while transitions depicted as boxes represent the end of an operation, or the occurrence of a failure. At the present stage of autonomous systems, these drawing conventions, and also the various labels, are literature: the dynamics of the model is not affected by these details, which are intended to make clearer the "physical" meaning of the model. In fact, obviously, the same net system could have been modeling a different system (say a parallel algorithm with some critical sections), since place/transition nets (like differential equations) are an abstract, and rather general, formalism for representing DEDS (continuous variable dynamic systems, respectively).

V. Interpreted Extensions of PN

In the sense of Section II, autonomous PN are semi-interpreted graphs: places have the meaning of state variables, the marking represents their value, and transitions are state transformers. We can associate a precise meaning with places and transitions (e.g., this place represents a store, this transition represents the arrival of a part, etc.; recall our previous example from Figure 6) in the form of a labeling (with statements) indicating to the human observer the intent of the model.

But in many situations the association of a meaning with the net objects has strong implications on the intended dynamic behavior of the model: 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 PN 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* it would 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 with 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, in practice, 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 with *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. The argumentation presented in Section II is now repeated, but using autonomous nets as the basic formalis (instead of directed graphs). The interpreted extensions lead to different PN based formalisms sharing some basic principles. In this sense, it can be said that PN are *multifacetted*, since inter-related variations and elaborations of some central concepts and objects allow the capture of diverse views, best suited for diverse objectives, of a range of systems. This is why we speak of a *Petri net paradigm*. Some of the formalisms developed from PN 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 [23] made them a standard within queueing networks, or LOTOS (Language of Temporal Ordering Specification) [24] is a standardized language, based on process algebra, oriented to open distributed systems).

In the following subsections we will illustrate two particular kinds of interpretations. Our selection is motivated by the relevance for automation applications. In Subsection V-A, constraints on the timing and conflict resolutions are provided, leading to timed/stochastic PN. These formalisms are used in performance evaluation and optimization, or in scheduling. In Subsection V-B, the evolution is constrained by external inputs, which is interesting in control. In this second case, 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 PN

One among the very many possible ways to incorporate time in a PN system is by associating it with 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 meantime 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 PN* [25]. The interval may be just a point, and then timing is deterministic, as in *timed PN* [26].
- In a probabilistic fashion, giving the pdf, as in *stochastic PN* [27], [28], [29], [30].
- In a possibilistic fashion, by way of fuzzy sets. In [31] *fuzzy PN* 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 PN.
- 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, [32] explores different sound possibilities, and it shows that the net structure should be carefully taken into account. Moreover, the stochastic interpretation of nets can consider, like in QN, a diversity of disciplines for queuing and service (e.g., FIFO), or routing (e.g., JSQ).

Regarding the matter of standardization, there is still a wide diversity of timed/stochastic PN formalisms. Nevertheless the use of *generalized stochastic PN* (GSPN) [29] 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 PN [26] where all transitions are controllable are used.

Coming back to the manufacturing example, if the purpose of the model is to evaluate the performance of the manufacturing cell, or to investigate different scheduling policies, then timing information (e.g., duration of operations, mean time between failures, etc.) can be incorporated to the model, for instance specifying the delay in the firing of transitions. Diverse timing specifications are possible (e.g., stochastic, deterministic, time intervals, etc.), each one best suited for a particular purpose or degree of detail required. In Figure 7 the delays are specifyed by their mean times.

In a preliminary design stage, where the issue is machine selection and dimensioning of the system, a stochastic timing specification, such as that of *generalizad stochastic PN* [29], is best suited. In the example we assume that the distribution of time delays corresponding to operations and movements is *phase-type*, namely *Erlang-3*, while failures and repairs follow *exponential* distributions. All other transitions are *immediate*, they

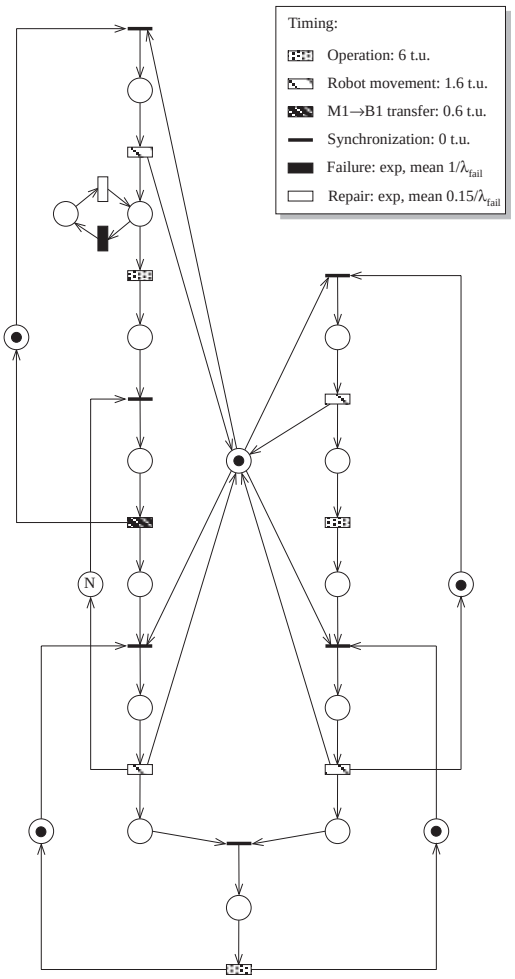

Fig. 7. A timed place/transition system that allows performace evaluation and optimization of a manufacturing cell.

fire as soon as they are enabled (so they are *prioritary* wrt. timed transitions). Conflicts between timed transitions are solved by *race* policy, while conflicts between immediate ones are solved in a probabilistic fashion).

By reversibility (a property that holds on the autonomous model), the reachability graph is strongly connected, and this allows to deduce ergodicity of the stochastic process with the interpretation given in the example, and the irreducibility of the underlying Markov chain. Markovian performance analysis can be used to assist in the dimensioning of *B1*, or to analyze its impact. With given failure and repair rates for *M1*, throughput is plotted versus buffer size in Figure 8.a. Economic considerations (in terms of throughput, required investment, and work in progress) would allow to optimize the buffer size. The plots in Figure 8.b

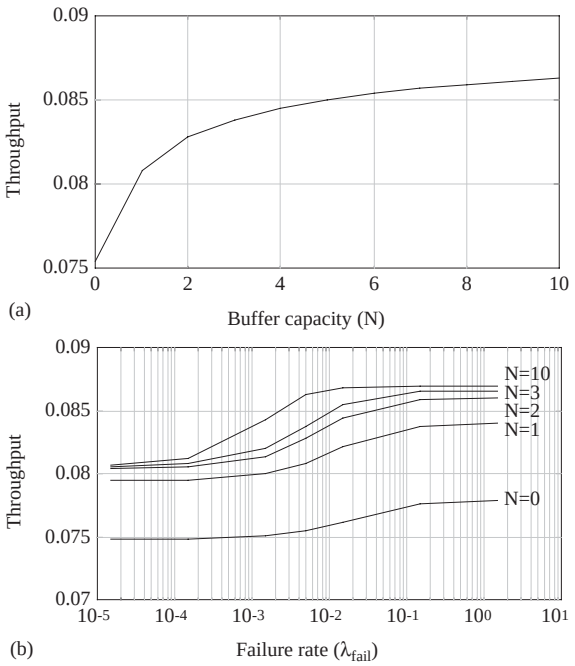

Fig. 8. Performance evaluation of the cell in Figure 6 with respect to buffer capacity and failure rate.

show how the effect of the buffer varies depending on the nature of the failures. Keeping the failure/repair ratio constant:

- Unfrequent failures with long repair times (left side of the plot) make the throughput insensible wrt. the buffer size, because the repair time exceeds largely the time to empty the buffer.
- On the other extreme, in the case of very frequent slight failures, a relatively small buffer is able to filter out the high frequency perturbations represented by the failures.
- When the order of magnitude of repair times are similar to the time required to empty the buffer, its size is most critical in order to increase the throughput.

Notice that for the case $N = 0$ the model in Figure 6 is changed, removing *B1* (*M1* becomes essentially identical to *M2*, except for the presence of failures), resulting in a more tight coupling of the machines that leads to a significantly lower throughput.

Assume that, after the optimization of the design that involved performance evaluation, the capacity of the buffer is fixed to two. Although the plant parameters are fixed, the actual performance of the system may vary depending on how it is controlled. The scheduler is in charge of controlling the evolution by enabling/disabling the transitions that initiate robot load

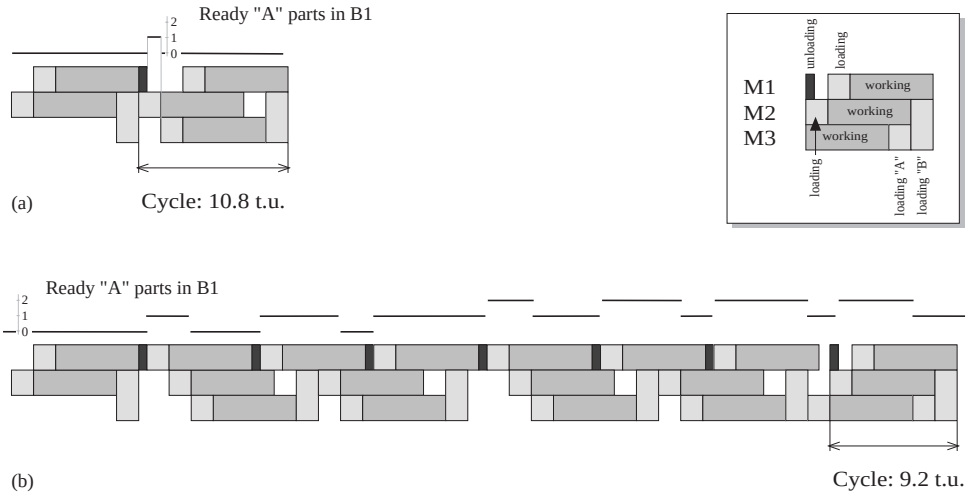

Fig. 9. Effect of different scheduling policies in the manufacturing cell of Figure 6.

operations (i.e., these are the *controllable* transitions).

Figure 9 shows the Gantt charts of two possible scheduling policies assuming deterministic timing and disregarding failures. In Figure 9.a operations are scheduled as soon as possible, solving eventual conflicts in the allocation of the robot by fixed priorities (*M2* is prioritary over *M1*). A periodic regime is quickly reached, in which:

- The cycle time is 10.8 (i.e., throughput 0.0926 *without failures*).
- The buffer contains at most one part, so parts are not accumulated to be used in the event of a failure.

The Gantt chart in Figure 9.b shows the evolution when the scheduler prevents interrupting *M1* until it gets blocked, and interrupting *M2* and *M3* from then on. This policy fills up the buffer to be prepared for eventual failures and achieves a cycle time of 9.2 (i.e., throughput 0.1087) in normal operation, thus the buffer allows to increase productivity in more than 11%.

B. Marking Diagrams

In order to use PN in automation it is necessary 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*. We call a PN model with an association of inputs and outputs similar to that of SD a *marking diagram*, as a natural — although not generally used — name, provided that in PN the state is called marking. Models of this kind can be found in [33], [13], [34], [35], [36], [37].

Inputs (either in the form of external events or logic conditions) are associated with 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 with 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.

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 [33], [34]. *Grafcet*, an International Standard since 1987, is another tool for the specification of logic controllers which is essentially a subclass of interpreted PN [38], [39].

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) [40], [41], [42].

Coming back to the manufacturing example, if the model is meant as a specification for a logic controller, the firing of transitions must be related to the corresponding external events or inputs, and the outputs that must be emitted have to be specified. The inputs, that condition the evolution of the controller, may come

from plant sensors (e.g., when R finishes loading $M2$ it emmits a signal `loaded_M2`) or from other levels in the control hierarchy (e.g., when the scheduler decides — in view of the state of the system and the production requirements — that $M1$ should be loaded, it sends `sched_M1`). The outputs may command the actuators (e.g., `START_M3` initiates the assembly sequence in $M3$) or send information to other levels in the control hierarchy (e.g., `REPAIR!` raises an alarm to call the attention of maintenance staff, or an interrupt that activates automatic recovery; `B1_CONT(m)` updates the number of ready "A" parts in the production database, etc.). The PN model in Figure 10 captures this information.

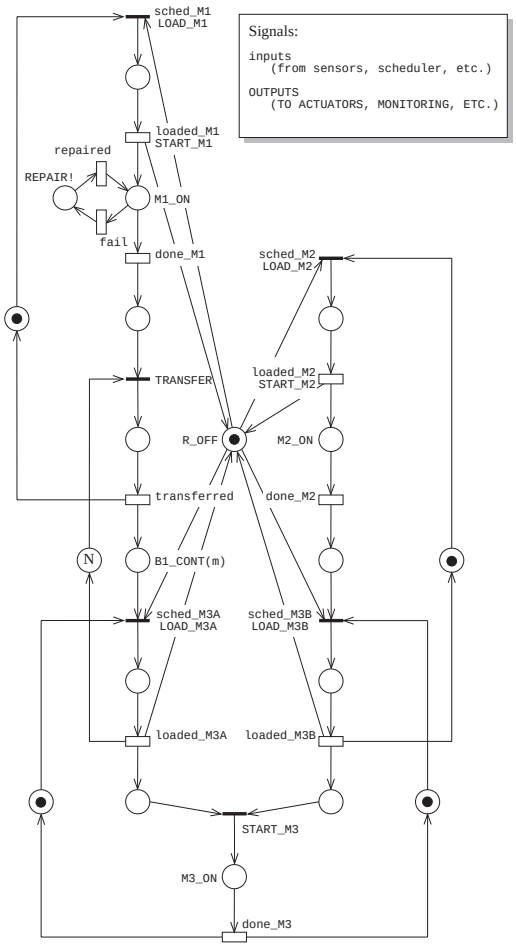

Fig. 10. A marking diagram that specifies the behavior of the logic controller of a manufacturing cell.

Following appropriate conventions in the specification (e.g., those imposed in the definition of Grafcet [38]), a model similar to this one could be used directly as a logic controller program.

Implementation of Marking Diagrams: Once a suitable PN model for a controller has been obtained it has to be *implemented*. Basically an implementation is a physical device which emulates the behavior expressed by the model. One advantage of using PN as a specification formalism is their independence wrt. the precise technology (pneumatic, electronic, etc.) and techniques (hardwired, microprogrammed, etc.) of the final implementation. Presently, in MS control, programmed implementations are the most usual, running on a wide range of computer systems (e.g., industrial PC's, programmable logic controllers, etc.).

The (programmed) implementation is affected by the selected PN *formalism* (low or high level, different interpretations of the firing rule), the *algorithmic approach* (interpreted, where the PN model is a data structure, or compiled, where a program is obtained from the given PN; centralized or parallel/distributed schemas), and the *computer architecture* (high or low level programming language; single or multi processor).

For the case of local controllers specified by low level PN with input and output signals (like that shown in Figure 10), a usual choice are interpreted implementations ("token players") [43], [13]. The basic schema is a cyclic program that reads the inputs, computes the evolution of the marking, and generates the outputs once and again. A major issue is the efficient computation of enabled transitions. An example of an efficient technique for this purpose are *representing places* (see, for instance, [44]). The idea is to appropriately select one input place per transition (its *representing place*). It is always possible (perhaps after some net transformations) to classify places as either representing or *synchronization places*, where each of the former is the representing place of all its output transitions. The marked representing places are kept in a list (we assume safeness for simplicity), that is updated at each transition firing. In each cycle, only the output transitions of marked representing places are tested for enabledness, eventually checking the marking of some synchronization places. A possible selection of representing places for the net in Figure 10 are all but R *idle*, *slots*, *ready* "A"*parts*, *waiting* "A", and *free* "B" (thus, these would be the synchronization places).

The inherent parallelism captured by a PN model is somehow dismissed in centralized implementations. Diverse parallel and distributed implementations have been proposed (see, for instance, [44]). The structure theory of PN allows to identify certain components in a given net that are useful for distributing

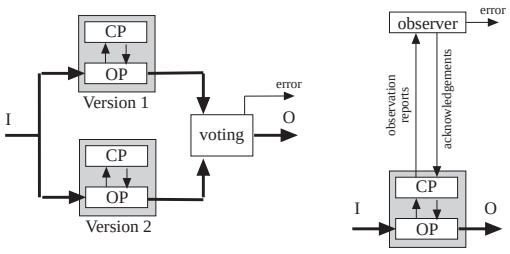

Fig. 11. Duplication versus observation.

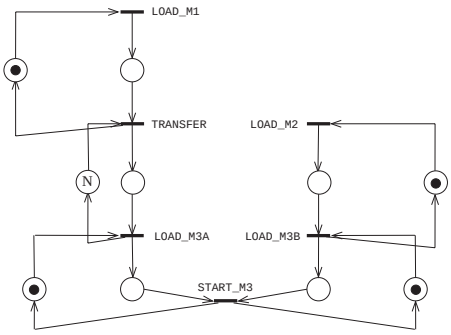

Fig. 12. A spy for the net in Figure 10.

or parallelizing the implementation. Particularly, live and safe state machine components lead to cyclic sequential processes that can be directly implemented, for instance, as Ada tasks. In such case, other places can be represented as global variables, semaphores, etc. Coming back to the example, we easily identify *M1* and *M2* as sequential tasks, *M3* can be decomposed into two synchronized sequential tasks, *slots* and *ready* "A" *parts* are semaphores, and R *idle* is a mutual exclusion semaphore.

An important issue when designing a control system is that of *safety*. Formal modeling and analysis tools are needed to engineer safe computer-controlled systems. For this task it is necessary to consider both the control system and its environment, for which PN are a suitable formalism [45]. When faults can happen the controller should be able to detect them and even react appropriately degrading system's performance as little as possible.

Let us briefly concentrate here on the detection and recovery of faults in the controller itself. Several techniques have been proposed to produce safe and/or *fault-tolerant* PN based controllers. We illustrate next one of these techniques which are supported by PN theory: the *spy/observer* schema.

In general, *N-version* programming techniques, that is, the controller is replicated and a voting mechanism is introduced [46] can be used. A less expensive schema is based on the idea of an *observer* [47] or *spy* [48], which accepts "normal" behaviors seen through some *observable*, or *check*, points. In Figure 11 duplication and observation schemas are compared. The observable points are transitions whose firing is reported to the spy/observer (transitions are classified as observable or non-observable, dually to the classification into controllable and uncontrollable). The spy/observer can be modeled as a PN equivalent to the original one wrt. observable transitions (non observable transitions are considered silent and can be reduced). In the final implementation, the code corresponding to the spy is merged with the code of the proper controller.

Considering as observable all the synchronization transitions in the net (i.e., those corresponding to the initiation of robot operations, initiation of a transfer from *M1* to *M2*, and initiation of an assembly in *M3*) the corresponding *spy* is shown in Figure 12. (Notice that this spy is obtained applying the same reduction rules that were applied for the analysis.)

VI. Concluding Remarks

Interpreted extensions *restrict* the behavior of the underlying autonomous model in order to capture the required features of the dynamic behavior. Obviously, since it affects the behavior, the interpretation must be carefully taken into account for analytical purposes. Nevertheless, the shared structure allows the re-use of *strucural 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 [49], [50]. For instance, the computability of the *visit ratios* (relative occurrence of transitions) in stochastic net models opened the way to discover the so called *rank theorems* [51], [52], [53], which characterize in polynomial time important logical properties. As another exam-

ple, symmetry detection at the logical level is a fundamental step towards efficient performance evaluation of stochastic colored PN [54]. Also, structure net theory provides essential concepts and techniques in order to make appropriate decompositions at net level, that can be used in *divide-and-conquer* performance evaluation techniques (either exact, approximate, or bounds) [55].

We tried to convey the idea that Petri nets constitute a formal paradigm, where specific formalisms for particular phases of the life-cycle can be defined. The interpreted extensions overviewed in this work do not represent a full set of possibilities. For instance, merging fuzzy sets concepts with Petri nets via extensions/interpretations (see [31], [56]) different fuzzy net formalisms arise. They have been applied either to model reasoning systems (transitions are fuzzy rules, and the net model represents an expert system) or to model DEDS (transitions represent state changes; in [57] fuzzy timing is attached to transitions, while in [58] a fuzzy marking is defined by attaching a fuzzy location to tokens).

References

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

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

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

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

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

[6] J. Carlier and P. Chretienne, *Problèmes d'Ordonnancement. Modélisation, Complexité et Algorithmes*, Masson, 1988.

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

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

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

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

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

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

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

[14] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[15] M. Silva and E. Teruel, "A systems theory perspective of discrete event dynamic systems: The Petri net paradigm," In Borne et al. [59], pp. 1–12.

[16] C. A. Petri, *Kommunication mit Automaten*, Ph.D. thesis, Institut für Instrumentelle Mathematik, Univ. Bonn, 1962.

[17] R. E. Cavallo, "Special issue on systems research movement: Characteristics, accomplishments and current developments," *General Systems Bulletin*, vol. 9, no. 3, 1979.

[18] C. A. Petri, "State-transition structures in physics and in computation," *International Journal of Theoretical Physics*, vol. 21, no. 12, pp. 979–992, 1982.

[19] C. A. Petri, "Interpretations of net theory," Tech. Rep. Interner Bericht 75-07, G.M.D., Bonn, Germany, 1975.

[20] M. K. Molloy, "Petri net modelling: The past, the present, and the future," in *Procs. of the 3^{th} Int. Workshop on Petri Nets and Performance Models (PNPM89)*. 1989, pp. 2–9, IEEE Computer Society Press.

[21] A. Ehrenfeucht and G. Rozenberg, "Partial 2-structures: Part I: Basic notions and the representation problem; part II: State spaces of concurrent systems," *Acta Informatica*, vol. 27, 1990.

[22] E. Badouel and P. Darondeau, "A survey on net synthesis," In Borne et al. [59], pp. 309–316.

[23] 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*, vol. 22, no. 2, pp. 248–260, 1975.

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

[25] P. M. Merlin, *A study of the Recoverability of Computer Systems*, Ph.D. thesis, Univ. California, Irvine, CA, USA, 1974.

[26] C. Ramchandani, "Analysis of asynchronous concurrent systems by Petri nets," Tech. Rep. Project MAC, TR-120, M.I.T., Cambridge, MA, USA, 1974.

[27] M. K. Molloy, "Performance analysis using stochastic Petri nets," *IEEE Trans. on Computers*, vol. 31, no. 9, pp. 913–917, 1982.

[28] 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*, vol. 2, no. 2, pp. 93–122, 1984.

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

[30] F. Bause and P. S. Kritzinger, *Stochastic Petri Nets: An Introduction to the Theory*, Vieweg, 1996.

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

[32] 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*, vol. 15, no. 7, pp. 832–846, 1989.

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

[34] 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*, vol. 33, no. 1, pp. 1–8, 1986, Reprinted in [60].

[35] M. Silva, "Logic controllers," in *IFAC Symposium on Low Cost Automation (vol. II)*, Milano, Italy, Nov. 1989, pp. 157–166.

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

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

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

[39] R. David, "Grafcet: A powerful tool for specification of logic controllers," *IEEE Trans. on Control Systems Technology*, vol. 3, no. 3, pp. 253–268, 1995.

[40] 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,* 12^{th} *World Congress on Scientific Computation*, 1988, pp. 522–524.

[41] 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 *Artificial Intelligence in Scientific Computation*, R. Huber et al., Eds., pp. 307–313. Scientific Publishing Co., 1989.

[42] R. Valette and M. Courvoisier, "Petri nets and artificial intelligence," in *Modern Tools for Manufacturing Systems*, R. Zurawski and T. Dillon, Eds., pp. 385–405. Elsevier, 1993.

[43] R. Valette, M. Courvoisier, J. M. Bigou, and J. Albukerque, "A Petri nets based programmable logic controller," in *IFIP* 1^{st} *Int. Conf. on Computer Applications in Production and Engineering*, Amsterdam, Holland, Apr. 1983.

[44] J. M. Colom, M. Silva, and J. L. Villarroel, "On software implementation of Petri nets and colored Petri nets using high-level concurrent languages," in *Proc.* 7^{th} *European Workshop on Application and Theory of Petri Nets*, Oxford, England, July 1986, pp. 207–241.

[45] N. G. Leveson and J. L. Stolzy, "Safety analysis using Petri nets," *IEEE Trans. on Software Engineering*, vol. 13, no. 3, pp. 386–397, 1987.

[46] A. Avizenis and J. P. Kelly, "Fault tolerance by design diversity: Concepts and experiments," *Computer*, vol. 17, no. 8, pp. 67–80, 1984.

[47] J. M. Ayache, P. Azema, and M. Diaz, "Observer, a concept for on line detection for control errors in concurrent systems," in *Proc.* 9^{th} *IEEE Int. Sypm. Fault-Tolerant Computing*, Madison, WI, USA, June 1992, pp. 79–86.

[48] S. Velilla and M. Silva, "The spy: A mechanism for safe implementation of highly concurrent systems," in *Real Time Programming 1988,* 15^{th} *IFAC/IFIP Workshop*, Valencia, Spain, May 1988, pp. 95–102, Pergamon.

[49] M. Silva, "Interleaving functional and performance structural analysis of net models," In Ajmone Marsan [61], pp. 17–23.

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

[51] J. Campos, G. Chiola, and M. Silva, "Properties and performance bounds for closed free choice synchronized monoclass queueing networks," *IEEE Trans. on Automatic Control*, vol. 36, no. 12, pp. 1368–1382, 1991.

[52] E. Teruel and M. Silva, "Structure theory of equal conflict systems," *Theoretical Computer Science*, vol. 153, no. 1-2, pp. 271–300, 1996.

[53] L. Recalde, E. Teruel, and M. Silva, "On linear algebraic techniques for liveness analysis of P/T systems," *Journal of Circuits, Systems, and Computers*, vol. 8, 1998, To appear.

[54] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, "Stochastic well-formed coloured nets for symmetric modelling applications," *IEEE Trans. on Computers*, vol. 42, no. 11, 1993.

[55] M. Silva and J. Campos, "Performance evaluation of deds with conflicts and synchronizations: Net-driven decomposition techniques," in *Procs. of the Int. Workshop On Discrete Event Systems (WODES '98)*. 1998, IEE Computer and Control Division.

[56] J. Cardoso and H. Camargo, Eds., *Fuzziness in Petri Nets*, Physica (Springer), To appear.

[57] T. Murata, "Temporal uncertainty and fuzzy-timing high-level Petri nets," in *Application and Theory of Petri Nets 1996*, J. Billington and W. Reisig, Eds., vol. 1091 of *Lecture Notes in Computer Science*, pp. 11–28. Springer, 1996.

[58] J. Cardoso, R. Valette, and D. Dubois, "Petri nets with uncertain markings," in *Advances in Petri Nets 1990*, G. Rozenberg, Ed., vol. 483 of *Lecture Notes in Computer Science*, pp. 64–78. Springer, 1991.

[59] P. Borne, J. C. Gentina, E. Craye, and S. El Khattabi, Eds., *Symposium on Discrete Events and Manufacturing Systems. CESA '96 IMACS Multiconference*, Lille, France, July 1996.

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

[61] M. Ajmone Marsan, Ed., *Application and Theory of Petri Nets 1993*, vol. 691 of *Lecture Notes in Computer Science*, Springer, 1993.