Statistica Neerlandica (2000) Vol. 54, nr. 2, pp. 237–255

Analysis of discrete-time stochastic Petri nets

W. M. P. van der Aalst[1,2], K. M. van Hee[1,3] and H. A. Reijers[1,3]

[1] *Eindhoven University of Technology, Department of Mathematics and Computing Science, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands*

[2] *Eindhoven University of Technology, Department of Technology Management, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands*

[3] *Deloitte & Touche Bakkenist, Management and ICT Consultants, P.O. Box 23103, NL-1100 DP, Amsterdam, The Netherlands*

The Petri net formalism is widely applied in both theoretical and practical settings. For the sake of performance analysis, the original Petri net model has been extended with the notion of time. This paper addresses the different issues involved with this extension. Also, it provides a state-of-the-art overview of different analysis techniques for timed Petri nets. A new analysis technique is presented, which combines the freedom of choosing arbitrary time distributions within a Petri net model on the one hand and efficient computation means on the other.

Key Words and Phrases: Petri nets, performance analysis.

1 Introduction

We are happy to contribute to the special issue in honor of Jaap Wessels. Directly or indirectly each of us was a scholar of Jaap. Kees van Hee was one of the first Ph.D. students of Jaap (1978). Wil van der Aalst was a Ph.D. student of both Jaap and Kees (1992). Hajo Reijers is currently a Ph.D. student of both Kees and Wil. The three of us are working in Computer Science on the theory and application of Petri nets, while Jaap's research is in stochastic processes. There is a strong relationship between both fields: each Petri net determines the incidence matrix of a discrete, possibly infinite, Markov chain. If random variables are associated to non-deterministic choices in a Petri net, we obtain a Markov process. Computer scientists use Petri nets to model complex systems and to verify if the modeled systems satisfy some correctness criterions (invariant properties and reachability of states). Petri nets are a powerful modeling technique because they provide a way to decompose the states of a system. (Note that a finite Petri net may correspond to an infinite Markov chain.) There is a growing interest in Computer Science to study performance of systems. Stochastic Petri nets extend the traditional Petri net with timing and probability features. As a result, a stochastic Petri net describes a

stochastic process. Clearly we did not deviate that much from Jaap's research interests.

The classical Petri net is a directed bipartite graph. The two types of nodes are called *places* and *transitions*. In a Petri net, places and transitions are connected via *arcs*. Places are graphically represented by circles, transitions by boxes or bars. Places can store *tokens*, represented by black dots. A distribution of tokens on the places of a net is called a *marking*, and corresponds to the "state" of the Petri net. A transition of a net is *enabled* at a marking if all its input places (the places from which some edge leads to it) contain at least one token. An enabled transition can fire: it removes one token from each of the input places, and adds one token to each of its output places. This is called the *firing rule*.

In Figure 1, a Petri net example is depicted. As a matter of coincidence, the Petri net has just as many transitions as places. The places have labels p_1, p_2 ... p_{11}; the transitions t_1, t_2 ... t_{11}. In the particular notation that is used for this example, two types of transitions are used. Transitions that are drawn as bars are timeless when fired; the transitions that are depicted as boxes consume time. As will be discussed in the next section, this is one of the many ways to add time behavior to Petri nets. Note that both types of transitions respect the firing rule. In the Petri net depicted in Figure 1, places p_8 and p_{10} are marked. The transitions that are enabled are t_7 and t_{10}. In particular, transition t_{11} is *not* enabled. If transition t_{10} would fire on basis of the token in p_8, then t_{11} becomes enabled as a token would be added to p_9. As a final remark, transitions t_7 and t_{10} are said to be *in conflict*, as they compete for the same token to fire.

This paper focuses on *timed Petri nets*. Because there are many ways to incorporate time in Petri nets, we present in Section 2 an up-to-date overview of the different timing mechanisms. In Section 3, we discuss the state-of-the-art analysis techniques for the various timed Petri net models. Based on this discussion, we motivate the analysis technique presented in this paper in Section 4. The technique differs from existing techniques in the sense that it combines the use of arbitrary stochastic timing with efficient means of computation and the rich expressiveness of the Petri net model.

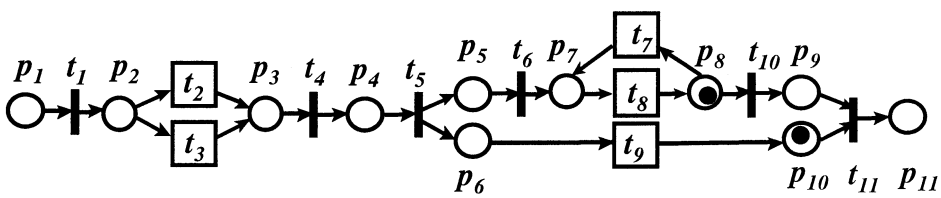

Fig. 1. A Petri net.

2 Time in Petri nets

Petri nets were originally proposed as a causal model without any notion of time or probability. In fact, the concept of time was intentionally avoided in the original work by Carl Adam Petri (1962). The addition of time restricts the behavior of the net, i.e., the dynamic behavior of the Petri net is only partially reflected by the network structure. However, for many practical applications, the addition of time is a necessity. Without an explicit notion of time it is not possible to analyze the performance (e.g., throughput and utilization) of the modeled system. Since the early seventies there has been a discussion within the Petri net community on the addition of time. More theory-oriented researchers oppose or simply ignore timing issues. More application-oriented researchers advocate and investigate different timing mechanisms and analysis techniques. We typically use a mixed approach, i.e., we start with a Petri net extended with time. In fact, we often use high-level Petri nets extended with color (data), time, and hierarchy as described by Van der Aalst (1993), Chiola *et al*. (1990), Ghezzi *et al*. (1991), Van Hee (1994), Jensen (1996), Lin and Marinescu (1987), Morasca *et al*. (1991), and Zenie (1985). For analysis purposes we abstract from some of these extensions. The reason for such abstractions is the need to make analysis tractable: many properties are undecidable for Petri nets extended with data and/or time. In the next paragraphs, we will discuss the different aspects of incorporating time in Petri nets.

2.1 *Location of delay*

There are many ways to introduce time into Petri net models. As explained in the introduction, a Petri net consists of places and transitions connected via arcs. Therefore, time can be associated with *places*, *transitions*, or *arcs*. In most timed Petri net models, transitions determine time delays. In only a few models, time delays are determined by places and/or arcs. It seems that it is more natural to associate time to transitions: Transitions represent activities and activities take time. However, authors such as Sifakis (1977, 1980) argue that it is more convenient to associate time to places since this leaves the original firing rule intact, i.e., enabling and firing are instantaneously. For high-level Petri nets with colored tokens (i.e., tokens carry a data value), it is most natural to attach timestamps to tokens, as done by Van der Aalst (1993), Van Hee (1994), and Jensen (1996). The timestamp indicates the time a token becomes available for consumption. In the models presented by Van der Aalst (1993), Van Hee (1994), and Jensen (1996) transitions set the timestamps of produced tokens, i.e., time delays are determined by transitions.

2.2 *Type of delay*

Independent of the choice where to put the delay (i.e., transitions, places, or arcs), several types of delays can be distinguished. In this paper, we distinguish between *deterministic*, *non-deterministic*, and *stochastic* delays. Many of the older timed

Petri net models by authors such as Van Hee *et al*. (1989), Ramchandani (1984), Sifakis (1977), Wong *et al*. (1985), and Zuberek (1980) use deterministic delays, i.e., the delay assigned by a transition, place, or arc is fixed. Deterministic delays allow for simple analysis methods but have limited applicability. In real applications, delays correspond to the duration of activities which are typically variable. Therefore, fixed delays are often less appropriate. There are two ways to describe the intrinsic variability. One way is to assume constraints on delays (e.g., it takes less than 15 minutes to type a letter), i.e., non-deterministic delays. Another way is to assume a probability distribution for each delay, i.e., stochastic delays. Most of the models handling non-deterministic delays use time intervals to specify the duration of the delay. Merlin (1974, 1976) introduced such a model in the early 1970's. Other models using interval timing have been proposed by Van der Aalst (1993, 1994), Van der Aalst and Odijk (1995), Berthomieu and Diaz (1991), and Berthomieu and Menasche (1993). However, most of the timed Petri net models use stochastic delays. In these models each delay is described by a probability distribution. To make analysis tractable typically only a restricted set of probability distributions is allowed. In the SPN model by Florin and Natkin (1982), only exponential delays (i.e., delays described by a negative exponential probability density function) are allowed. The widely used GSPN model by Marsan *et al*. (1984) allows for both immediate transitions (i.e., transitions with no delay) and timed transitions (i.e., transitions with exponential delays). Models allowing for arbitrary probability distributions typically defy exact analytical analysis. Another way to classify the types of delay used in a timed Petri net model is to distinguish between *discrete* and *continuous* delays. Most discrete models use the natural numbers as the time domain. Continuous models typically use the set of non-negative real numbers as the time domain. Nearly all timed Petri nets allow for continuous delays. The analysis method described in the second part of the paper assumes discrete delays. This is not a real limitation since a fine-grained discrete time domain can be used to approximate a continuous time domain.

2.3 *Preselection versus race semantics*

Adding time to Petri nets requires a redefinition of the enabling and firing rules. In a classical Petri net a transition is enabled if each of the input places contains enough tokens (typically 1), only enabled transitions can fire, and firing is instantaneously (i.e., the moment tokens are consumed from the input places, tokens are added to the output places). Transitions are in conflict if they share input places. Note that firing a transition in conflict with other transitions may disable some or all of these transitions. The choice between several enabled transitions in conflict with each other is resolved in a non-deterministic manner. When adding time to a Petri net the enabling and firing rules need to be modified to specify how conflicts are resolved (i.e., the relation between enabling and firing) and whether firing is instantaneous or not (the semantics of the firing rule). Clearly, these two issues are related. Assume that transitions determine the delays. If firing is

instantaneous (i.e., does not take any time), then it is necessary to associate time to the enabling of a transition. If time is associated to the enabling, there is no need to explicitly define how conflicts are resolved, i.e., enabled transitions "race" against each other and the one that is scheduled to fire first will fire. This firing/enabling semantics is called the *race semantics*. It is also possible to specify the way conflicts are resolved explicitly. This firing/enabling semantics is called the *preselection semantics*. For example, priorities or probabilities are used to resolve conflicts. In the preselection semantics there is no race between enabled transitions: The moment transitions become enabled one of the enabled transitions is selected. Race semantics are typically combined with instantaneous firing, i.e., time is in the enabling of transitions. Therefore, we also use the term *enabling delays* to refer to these semantics. Preselection semantics are typically combined with *holding times*, i.e., tokens reside for some time inside a place or transition. Note that for race semantics the resolution of conflicts and the delay are handled by the same mechanism. For preselection semantics the mechanism to resolve conflicts is separated from the actual delay. Most of the stochastic Petri nets by authors such as Balbo and Silva (1998), Florin and Natkin (1982), Ajmone Marsan (1990), and Ajmone Marsan *et al*. (1984, 1985, 1986, 1995) use race semantics. As established by Van der Aalst (1992), race semantics allow for a more direct translation to Markov chains and timed Petri nets using race semantics are more expressive than timed Petri nets using preselection semantics. For example, race semantics allow for a compact representation of time-outs. Preselection semantics are more intuitive and easier to use. Therefore, most of the high-level Petri nets such as the ones described by Van der Aalst (1993), Van Hee (1994) and Jensen (1996) support preselection semantics. Other authors such as Razouk and Phelps (1984) propose a mixture of race and preselection semantics.

For preselection semantics the delays (i.e., holding times) can be associated to the firing of a transition (e.g. described by Berthomieu and Diaz 1991) or the minimal time a token spends in a place (e.g. described by Sifakis 1980).

For race semantics the delays are associated to the enabling time. Note that an enabled transition can be disabled by another transition in case of a conflict. Such a transition loses the race and will not fire. If the transition becomes enabled again, a new race starts. In this new race there are several possibilities for the new enabling time of this transition. Authors such as Balbo and Silva (1984), Ajmone Marsan *et al*. (1985, 1995) typically distinguish three so-called memory policies: *age memory*, *enabling memory*, and *reset memory*. For age memory, the remaining enabling time is frozen the moment the transition becomes disabled and is resumed the moment the transition becomes enabled again. For enabling memory, a new enabling time is sampled every time a transition becomes enabled, i.e., previously interrupted transitions have to start from scratch. For reset memory, a new enabling time is sampled every time a transition fires, i.e., also transitions not in conflict with the transition that fired are interrupted and have to resample a new enabling time. It is interesting to note that for stochastic Petri nets with just exponential delays the three memory

policies coincide. The memoryless property of the negative exponential probability density function makes the residual enabling time statistically equivalent to the originally sampled enabling time.

2.4 *Capacity, priority, and queuing policy*

For timed Petri nets, the capacity of places and transitions is relevant. Places can have a limited capacity to restrict the number of tokens residing in a place at the same moment in time. Transitions can have a capacity to limit the number of concurrent enablings/firings of the same transition. Consider a transition with one input place containing three tokens. Is this transition enabled three times under race semantics? Can the transition fire concurrent with itself under preselection semantics? To answer these questions, we identify three types of capacity related semantics: *single server semantics*, *multiple server semantics*, *infinite server semantics*. For single server semantics the capacity of a place/transition is 1, for multiple server semantics the capacity of a place/transition is some integer k, and for infinite server semantics there are no capacity restrictions. Most timed Petri net models assume infinite server semantics.

Several timed net models allow for a priority mechanism, i.e., if multiple transitions compete for the same token, the transition with the highest priority fires. Note that the priority mechanism can be used for preselection purposes. In the widely used GSPN model by Marsan *et al*. (1984) immediate transitions (i.e., transitions with no delay) have priority over timed transitions.

Some Petri net models allow for the specification of queuing policies. However, since tokens in the same place (of the same color) are indistinguishable, it often does not make any sense to choose a queuing discipline. In general priorities (i.e., not transition priorities but token priorities), random selection (RS), and processor sharing (PS) are easy to handle in a stochastic Petri net, as established by Balbo and Silva (1984). State-dependent queuing disciplines such as first-come-first-served, last-come-first-served, longest-job-first, and earliest-due-date-first are more difficult to represent and analyze.

In this section we presented an overview of the various ways time can be incorporated in Petri nets. In the second part of this paper, we will use a timed Petri net model with time in transitions using preselection semantics, i.e., firing a transition takes time and conflicts are resolved independent of the firing time using probabilities. The type of delay is stochastic. However, unlike most stochastic Petri nets, we use a discrete time domain. On this discrete time domain, we allow arbitrary, but finite, probability distributions. The model used in this paper does not support priorities. However, transition priorities could easily be added. Since the Petri nets considered in this paper are safe (i.e., the maximum number of tokens in a place is one), capacity and queuing policy are not relevant.

3 Analysis of timed nets

As indicated in the previous section, there are many ways to introduce time in Petri nets. All timed Petri net models are executable, i.e., it is possible to construct a trace of the modeled system by playing the token game. Therefore, simulation can be used to analyze the model. If all non-determinism is replaced by stochastic measures (i.e., delays and conflict resolution), then simulation can be used to obtain confidence intervals for performance measures such as utilization and throughput. Because simulation does not require difficult mathematical techniques, it is easy to understand for people with a non-technical background. Simulation is also a very flexible analysis technique, since it does not set additional constraints. However, sometimes simulation is expensive in terms of the computer time necessary to obtain reliable results. Another drawback is the fact that (in general) it is not possible to use simulation to *prove* that the modeled system has the desired set of properties. In the remainder, we discuss alternative analysis techniques to overcome the limitations of simulation approaches. Since analysis techniques are typically restricted by the type of delay, we first consider timed Petri nets with deterministic timing, then timed Petri nets with non-deterministic timing, and finally timed Petri nets with stochastic timing.

3.1 *Deterministic timing*

There are several methods to calculate upper and lower bounds for the *cycle time* of a timed Petri net with deterministic delays. For example, check authors such as Murata (1992), Ramamoorthy and Ho (1980), Ramchandani (1984), and Sifakis (1939). The cycle time is a criterion for the performance of the system. For a specific class of deterministic timed Petri nets, the so-called Timed Event Graphs, the exact cycle time can be computed quite efficiently, see Ramamoorthy and Ho (1980) and Chretienne (1983). Other researchers such as Zuberek (1980) analyze deterministic timed Petri nets by building the reachability graph. Although this requires a lot of computing effort, such a graph can be used to answer a variety of questions. A serious drawback of these methods is the fact that in many real systems the activity durations are not fixed, i.e., they vary because of disturbances and other interferences. Assuming deterministic delays often results in inaccurate results.

3.2 *Non-deterministic timing*

Most timed Petri nets models using non-deterministic delays, such as described by Van der Aalst (1993, 1994), Van der Aalst and Odijk (1995), Berthomieu and Diaz (1991), Berthomieu and Menasche (1993), Merlin (1974), and Merlin and Faber (1976), use intervals to describe lower bounds and upper bounds for the duration of activities. The method presented by Berthomieu, Diaz and Menasche (1983, 1991) uses Merlin's (1974) timed Petri net model. The method generates a reachability graph where nodes represent state classes instead of states. Sets of linear equations

are solved to calculate these state classes. The method is able to reduce the number of states by using a relative time scale. Another method using interval timed colored Petri nets was presented by Van der Aalst (1993). This method uses an absolute time scale and allows for colored tokens. The method also generates a reachability graph where nodes represent state classes. The number of states is reduced by exploiting "timed" specialization and generalization properties. Van der Aalst and Odijk (1995) describe an application of this method and Van der Aalst (1992) gives two additional analysis methods based on interval timing.

3.3 *Stochastic timing*

The majority of stochastic Petri nets models uses a continuous time domain. In these models, each delay is described by a probability density function. For arbitrary probability density functions, only simulation or approximation are feasible analysis techniques. Therefore, many stochastic Petri nets models impose restrictions on the type of delay distribution. In the SPN (Stochastic Petri Net) model as described by Florin and Natkin (1982) and Molloy (1981) only exponential delays are allowed. Due to the memoryless property of the exponential distribution and the race semantics, Florin and Natkin (1982) and Molloy (1981) show that SPN's are isomorphic to continuous time Markov chains. The number of states of the Markov chain corresponds to the number of reachable markings of the SPN. The GSPN (Generalized Stochastic Petri Net) model extends the SPN model with immediate transitions. Immediate transitions fire without any enabling time and have priority over timed transitions (i.e., transitions with exponential enabling times under the race semantics). A marking is vanishing if an immediate transition is enabled. A marking is tangible if only timed transitions are enabled. The GSPN model distinguishes between these two types of markings: only tangible markings consume time, i.e., the average sojourn time of vanishing states is zero and the average sojourn time of tangible states is positive. The dynamics of a GSPN corresponds to a semi-Markov process: The embedded Markov chain which ignores the sojourn time in each state is a discrete time Markov chain. By using the embedded Markov chain, it is fairly straightforward to calculate various performance measures. Note that only the tangible states consume time. Therefore, the vanishing markings are not relevant for most performance measures. Therefore, as shown by Balbo and Silva (1984) and Ajmone Marsan *et al*. (1985, 1995), it is possible to reduce the number of states by eliminating the vanishing markings in the embedded Markov chain.

The GSPN model has been extended in various directions. First of a the GSPN model has been extended with marking dependent transition probabilities and enabling delays. It is easy to see that such an extension can be handled by using an embedded Markov chain as long as immediate and timed transitions do not interfere. Second, the GSPN model has been extended to allow for other types of delay distributions (i.e., non-exponential). Basically there are two ways to incorporate non-exponential delays. First of all, it is possible to introduce transitions with arbitrary

delay distributions as long as none of these transitions can be enabled concurrently. The work of Ajmone Marsan and Chiola (1987) is an example of the DSPN model which allows for timed transitions which have either fixed (i.e., deterministic) or exponential enabling times. The DSPN model can be analyzed as a semi-Markov process as long as only one deterministic transition is enabled at the same time and the enabling memory policy is assumed. Several variations and refinements of the DSPN have been proposed in literature (Balbo and Silva 1998 give pointers). Another approach to incorporate non-exponential delays is to allow for delay distributions which can be represented by a continuous time Markov chain. Examples of such delays are the Erlang, the hyperexponential, and the phase-type distribution. The possibility to incorporate such delays was already mentioned by Florin and Natkin (1982) and Molloy (1981). The relation between the various memory policies and phase-type distributed transitions is discussed by Balbo and Silva (1984) and Ajmone Marsan *et al*. (1995). Using non-exponential delays which are expanded to multiple phases in the corresponding Markov chain typically results in Markov chains which are difficult to analyze. (In worst case, the size of the Markov chain grows exponentially in the number of phases.)

Nearly all stochastic Petri net models described in literature use a continuous time domain and are analyzed using Markov-chain analysis. For most applications, the GSPN model and its extensions are a convenient and practical tool. However, these models have two potential drawbacks. First of all, the size of the Markov chain typically grows exponentially in the size of the corresponding Petri net. Second, the use of non-exponential delays is restricted: Only by restricting the topology of the Petri net or by expanding the Markov chain to encode the phases of a phase-type distribution it is possible to allow for non-exponential delays. Net-driven decomposition techniques can be used to partially alleviate the state-explosion problem, as shown by Balbo and Silva (1984). Unfortunately, it is almost impossible to handle arbitrary delay distributions. Therefore, we propose an alternative analysis technique. The technique uses a discrete time domain instead of a continuous time domain. An arbitrary probability distribution can be used for specifying the firing delays. (Recall that the stochastic Petri net used in this paper uses preselection and holding durations for transition firings.) The constraints on the topology are quite acceptable: Typical routing constructs such as sequencing, parallelism, choice, and iteration are supported. Since the method allows for arbitrary distributions, the algorithm is potentially inefficient, i.e., distributions over the discrete time domain are represented explicitly. However, the Fast Fourier Transform is used to improve the efficiency of several analysis routines. Moreover, the method allows for efficient construction rules and decomposition techniques.

4 Discrete time stochastic nets

The analysis method presented in this section is applicable to a subclass of the so-called *discrete time nets*. As a matter of fact, this subclass coincides with the

process-algebraic expressions that can be constituted with the merge ($\|$), choice ($+$), sequential ($\cdot$), and star (*) operators in the process algebra ACP, as described by Bergstra and Klop (1984). From a practical viewpoint, its is possible to model the majority of typical business processes as found in organizations as banks, insurance companies, and governmental agencies with this particular subclass.

The nets under consideration in this section are composed out of smaller blocks. In the next paragraph, we will start to describe the composition method (synthesis). Next, the notion of throughput is introduced, so that the computation of the throughput time can be explained. An example is added to illustrate the approach. Finally, possible extensions of the approach are discussed in this section.

4.1 *Synthesis*

The models on which we will apply the analysis technique are based on the classical Petri net.

DEFINITION (PETRI NET) A *Petri net* is a triplet (P, T, F):

- P is a finite set of places,
- T is a finite set of transitions ($P \cap T = \emptyset$),
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation).

A place is called an *input place* of a transition t iff there exists a directed arc from p to t. Place p is called an *output place* of transition t iff there exists a directed arc from t to p. We use $\bullet t$ to denote the set of input places for transition t. The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g. $p\bullet$ is the set of transitions sharing p as an input place.

Using the notion of a classical Petri net, we can introduce the discrete time stochastic nets. Our approach is applicable to a sub-class of these nets (cf. Van der Aalst 1998).

DEFINITION (DISCRETE TIME STOCHASTIC NETS, DTS-NET). A Petri net PN = (P, T, F) is a *discrete time stochastic net*, or *dts-net* for short, iff:

1. PN has two special places i and o; place i is a source place: $\bullet i = \emptyset$; place o is a sink place: $o\bullet = \emptyset$;
2. if transition t^* would be added to the set of transitions T and the arcs (o, t^*) and (t^*, i) would be added to the flow relation F of PN, for every two places (transitions) x and y, there is a directed path leading from x to y.

The first requirement in the definition of a dts-net reflects the beginning and termination of the process. That event is to be handled in such a way that a desired end situation is reached. The second requirement in the definition ensures that there are no dangling transitions or places. In other words, transitions or places are not permitted to be part of a Petri net model if they do not contribute to the dynamic

behavior of the model. A dts-net specifies the dynamic behavior of a *single case in isolation*. Note that the net in Figure 1 is a dts-net.

The synthesis method to construct a dts-net is comparable with the top-down synthesis technique as used by Valette (1979) and Van der Aalst (2000) in which a transition is substituted by a block. We will use four blocks that can be used to substitute transitions. These are: sequence, choice, parallelism, and iteration. The blocks that express these constructs are depicted in Figure 2.

Arc labels occur in the (b) iteration and (d) choice blocks. They represent the values of a Bernoulli-distributed random variable that is associated with these blocks. An independent draw from its distribution function determines the route of the flow of tokens through the net. In other words, conflicts in the blocks are resolved using preselection semantics. Each new application of a block is accompanied by the introduction of a new, independent random variable. Furthermore, two types of transitions are distinguished. *Immediate* or *timeless* transitions are depicted in the blocks as black bars; the rectangular blocks are transitions to which transitions delays (i.e., holding times) are associated for the firing of a transition.

As the starting point of each Petri net model construction we will take a simple start net. This net is depicted in Figure 3.

The net consists of one transition, a source place, a sink place, relations between them, and an initial marking of the source place. It is easy to see that this Petri net is a dts-net. We will refer to this specific Petri net as SN, for start net. Net synthesis, then, is straightforward. The timed transitions in the start-net can be substituted by one of the blocks, after which in a recursive fashion each of the resulting timed transitions may again be replaced by a block. For a formal description of this substitution, the reader is referred to Valette (1979). Note that the nets that can be constructed in the described fashion coincide with the process algebraic expressions. A net synthesis of the dts-net as introduced in Figure 1, is depicted in Figure 4. As the start net is a dts-net, it is easy to verify that the resulting net is a dts-net too.

The delay or holding time of each transition firing is called the *service time*. As

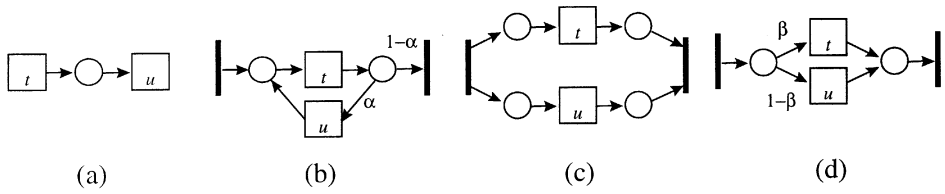

Fig. 2. Sequence (a), iteration (b), parallelism (c), and choice (d).

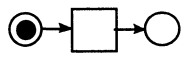

Fig. 3. The start net SN.

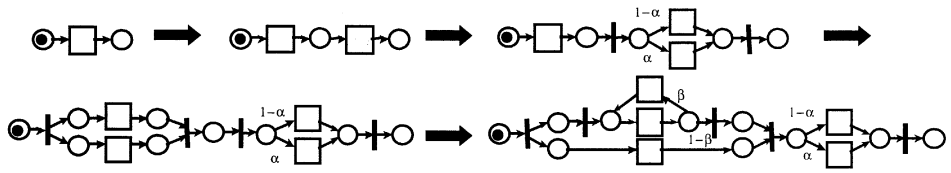

Fig. 4. Synthesis of a dts-net from the start net.

stated before, in our approach a delay characteristic is associated with each timed transition. All service times for one specific timed transition are independently sampled on basis of this probability distribution. We will call this distribution the *service distribution*. Its matching probability density is the *service density.*

DEFINITION (SERVICE TIME, SERVICE DENSITY, SERVICE DISTRIBUTION, UPPER BOUND). The time which is taken by a service of transition $t \in T$ within a dts-net W = (P, T, F) is called the service time. The service time is a discrete random variable t. Its matching probability density $f_t : N \rightarrow R$ is called the service density (N is the set of natural numbers and R is the set of non-negative reals):

$$f_t(k) = \mathbb{P}(t = k), \qquad \text{for } k \in \mathrm{N}.$$

Its matching probability distribution $F_t : N \rightarrow R$, is called the service distribution:

$$F_t(k) = \mathbb{P}(t \leqslant k), \qquad \text{for } k \in \mathrm{N}.$$

The service time t is bounded: there is an upper bound $u_t \in \mathrm{N}$ which is the smallest value such that for all $j \in \mathrm{N}$ and $j \geqslant u_t$ holds that $f_t(j = 0$.

Our assumption of the service time to be discrete is no real constraint: for practical purposes it is always possible to find an appropriate representation. As we will see, we do need the boundedness of the service time to perform some of the computations to come.

For each dts-net the *throughput time* is now defined as the time that elapses between the arrival of a token at the source place and the corresponding arrival of a token in the sink place. Similar to the service time notions, it is possible to distinguish the throughput distribution and throughput density of a dts-net. Assuming the service densities to be known of each of the transitions within a block, we will show how the throughput density of an entire block can be computed. Each of the blocks requires a specific algorithmic approach. For the computations to come, we assume a source and a sink place to be added to each of the blocks, so that our notion of throughput time is applicable to these blocks.

SEQUENCE. Consider the sequence block B in Figure 2 with two transitions t and u. We want to compute throughput density f_B, given f_t and f_u.

$$\text{Let } y \in \mathrm{N},\ f_B(y) = \sum_{i=0}^{y} \mathbb{P}(t = i \wedge u = y - i)$$

$$= \sum_{i=0}^{y} \mathbb{P}(t = i)\mathbb{P}(u = y - i) = f_t \otimes f_u(y).$$

To constrain the computation effort drastically, the convolution $f_t \otimes f_u$ can be computed with the Fast Fourier Transform and its inverse. Therefore, we can compute a vector representation of $f_t \otimes f_u$ in $\theta(n \log n)$ time, with n the smallest power of two that is at least twice as large as the maximum of the upper bounds of tasks s and t.

ITERATION. Now consider the iteration block B as depicted in Figure 2. We want to compute throughput density f_B, given f_s and f_t.

$$\text{Let } y \in \mathrm{N} \text{ then } f_B(y) = \sum_{n=0}^{\infty}(1-\alpha)\alpha^n \mathbb{P}\left(\sum_{j=1}^{n+1} \underline{t}_j + \sum_{j=1}^{n} \underline{u}_j = y\right)$$

$$= \sum_{n=0}^{\infty}(1-\alpha)\alpha^n \mathbb{P}\left(\bigotimes_{j=1}^{n+1} f_t \otimes \bigotimes_{j=1}^{n} f_u(y)\right),$$

with notation $\bigotimes_{j=1}^{n} a_j = a_1 \otimes a_2 \ldots \otimes a_n$.

Now consider $\vec{B}$, the vector representation of f_B. Its Discrete Fourier Transform is:

$$DFT_l(\vec{B}) = DFT_l\left(\sum_{n=0}^{\infty}(1-\alpha)\alpha^n\left(\bigotimes_{j=1}^{n+1} \vec{t} \otimes \bigotimes_{j=1}^{n} \vec{u}\right)\right)$$

$$= \sum_{n=0}^{\infty}(1-\alpha)\alpha^n DFT_l^{n+1}(\vec{t})DFT_l^n(\vec{u}) = \frac{(1-\alpha)DFT_l(\vec{t})}{1-\alpha DFT_l(\vec{t})DFT_l(\vec{u})}$$

with yet unknown l.

We do not know yet an approximated, proper size l of the vectors we have to "feed" the *DFT*. We cannot expect f_B to have an exact upper bound. After all, t and u can be executed infinitely often if α is non-zero. We will show how an approximated, relevant length of $\vec{B}$ can be determined before actually computing $\vec{B}$. We would be happy to find a value ν such that for some very small ε holds that $\mathbb{P}(\underline{B} \geqslant \nu) \leqslant \varepsilon$. Using Chebyshev's inequality – for any random variable $\underline{x}$ for which $E\underline{x}^2$ exists holds that $\mathbb{P}(|\underline{x} - E\underline{x}| \geqslant c) \leqslant \operatorname{var}\underline{x}/c^2$ – and the mean and variance of the throughput density f_B, it can be determined which probability part of the density falls before or after a hypothetical border. As the service time for any transition t is denoted by $\underline{t}$ we will denote its mean by $E\underline{t}$ and its variance by $\operatorname{var}\underline{t}$. The mean and variance of $\underline{B}$ are:

$$E\underline{B} = \frac{E\underline{t} + \alpha E\underline{u}}{1-\alpha}, \qquad \text{var}\,\underline{B} = \frac{\text{var}\,\underline{t} + \alpha\,\text{var}\,\underline{u}}{1-\alpha} + \alpha\left(\frac{E\underline{u} + E\underline{t}}{1-\alpha}\right)^2,$$

so that we can derive using Chebyshev and our desired value ε that:

$$\nu \geqslant E\underline{B} + \sqrt{\frac{\text{var}\,\underline{B}}{\varepsilon}}.$$

Concluding, given f_t and f_u, we can compute a vector representation of f_B for the iteration block by using the *DFT*:

$$DFT_\nu(\vec{B}) = \frac{(1-\alpha)DFT_\nu(\vec{t})}{1-\alpha DFT_\nu(\vec{t})DFT_\nu(\vec{u})},$$

with ν the smallest power of two such that $\nu \geqslant E\underline{B} + \sqrt{\text{var}\,\underline{B}/\varepsilon}$.

With the *DFT* we can compute a vector representation of f_B in $(\nu \log \nu)$ time, with ν as specified. To appreciate its efficiency we have to establish the computing time of calculating f_B in a straightforward manner. The complexity of this calculation depends on the maximal number of successive times that transitions t and u can be executed. We know that if both $f_t(0)$ and $f_u(0)$ are equal to zero, at most ν executions of these transitions are of interest. Any more executions of transitions t and u would result in throughput times that we do not take into consideration. As a result, a straightforward approach requires the convolution of ν times the function f_t and f_u. This is an operation requiring $\theta(n^\nu)$ time, with n the maximum of upper bounds of transitions t and u. A comparison with the $\theta(\nu \log \nu)$ time required by our newly found computation method illustrates the efficiency of the latter.

PARALLELISM. The block we will consider next is the parallel block. Consider the parallel block B in Figure 2 with transitions t and u. Due to the structure of B, transitions t and u can be executed in parallel. We want to compute throughput density f_B, given f_t and f_u.

$$\text{Let } y \in \mathrm{N},\ f_B(y) = \mathbb{P}(\underline{t}\max\underline{u} = y) = f_t(y)\sum_{i=0}^{y} f_u(i) + f_u(y)\sum_{j=0}^{y-1} f_t(j)$$

$$= \begin{cases} f_t(y)F_u(y) + f_u(y)F_t(y-1), & y > 0 \\ f_t(y)f_u(y), & y = 0 \end{cases}$$

The computation of the distribution function F_t can be performed in u_t steps, just as the distribution function F_u can be done in u_u steps. Therefore, the total computation of f_B can be done in $\theta(t)$ time, with t equal to the maximum of u_t and u_u.

CHOICE. The final block we will consider in this section is the choice block. Initiating choice block B in Figure 2 results in either the execution of transition t or

transition u, with respective probabilities α and $1 - \alpha$. We would like to compute throughput density f_B, given f_t and f_u.

$$\text{Let } y \in \mathrm{N},\ f_B(y) = \alpha f_t(y) + (1 - \alpha) f_u(y)$$

From this expression follows that we can compute f_B in $\theta(n)$ time, with n equal to the maximum of u_s and u_t.

4.2 *Computation*

Now that for each block it is clear how the overall service time characteristics can be computed, the computation of the throughput time of a complete synthesized dts-net is rather straightforward. If the reverse direction is taken of the synthesis route, the service time characteristic for each specialized transition can be computed, which – in its turn – can be used to compute the service characteristic of a transition that has been specialized earlier during the synthesis. Finally, the service time characteristic of the transition in the start net can be computed. As a result, the throughput time characteristic of the complete dts-net is known.

To illustrate the presented algorithms we have computed the throughput density of the synthesized dts-net depicted in Figure 5. Each of the five timed transitions in the dts-net has been modeled to behave in accordance with a distinct, quirky service density. These service densities, together with the dts-net under consideration, are depicted in Figure 5. The service densities in this example are bounded by 64 time units.

The probabilities α and β of the model have been set on 0.3 and 0.15 respectively. For the computation of the throughput density of the iteration block an inaccuracy (ε) of 1 percent has been allowed. The resulting throughput density for the entire dts-net is depicted in Figure 6.

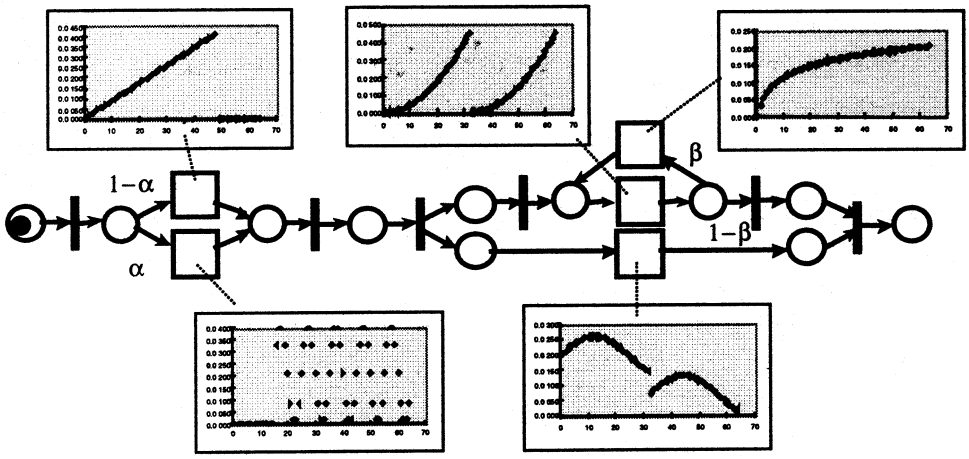

Fig. 5. Dts-net with service time distributions.

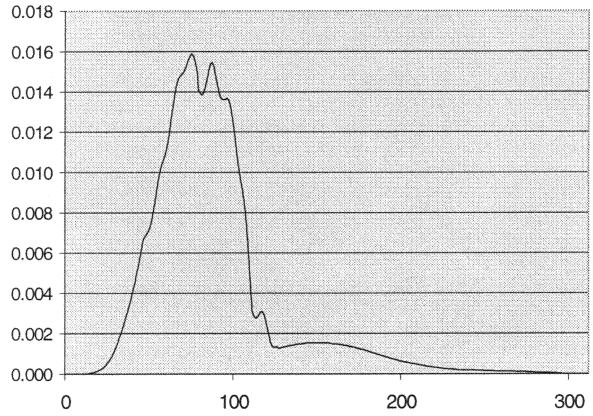

Fig. 6. Resulting throughput time density.

4.3 *Extensions of the approach*

The approach that is presented in this section can be extended in at least two directions. The first and most straightforward direction is to add more blocks to the current set of synthesis blocks. The subclass of dts-nets that can be synthesized thus becomes larger, and the application domain grows. At the time of writing, the authors are working on several complex blocks to be added.

We will sketch another interesting direction of extension informally. Intuitively, it seems possible to compute the throughput time of each free-choice, acyclic and sound dts-net. A dts-net is free-choice iff, for every two places p and q either $(p\bullet \cap q\bullet) = \varnothing$ or $p\bullet = q\bullet$. A dts-net is acyclic if there is no directed path within the net leading from either a place or a transition to itself. Lastly, a dts-net is sound if it is ensured that a token in the source place will always lead to a token in the sink place, while the rest of the net is empty. On basis of Petri net theory it can be proven that, starting with one token in the source place, each place will contain at most one token during execution. Moreover, the outcome of each conflict can be modeled as a random variable that is independent of the execution order of the net. As a result, it is possible to define an arrival function for each single place in the net. That function expresses the time that elapses between the marking of the source net and the arrival of a token in the particular place. Of course, the arrival function of the sink place is equal to the throughput distribution of the net. Note that this informally sketched method is stochastically feasible on basis of formal reasoning about *the structure and behavior of the net*.

If we mix both approaches – the one presented in this paper and the one described above – we have a powerful toolbox to analyze hybrid nets. Suppose we have a hierarchical dts-net, such that each transition in the net (recursively) can be a larger Petri net in itself. Considered on the highest level, a dts-net may be free-choice, acyclic and sound, but it cannot be constructed by our block synthesis. On a

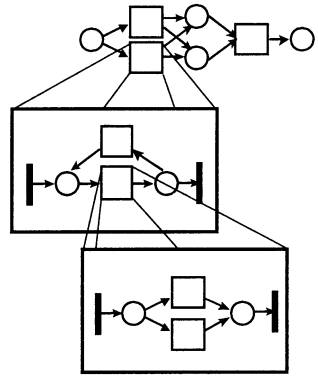

Fig. 7. A hybrid approach.

second level, the Petri nets that determine the behavior of the transitions may be synthesized out of blocks including cyclic constructions such as the iteration block. Yet a following, deeper level in the nets may be free-choice, sound and acyclic again or constructed from blocks, etc. On each level of the net we can apply the feasible approach, until the throughput behavior of the entire net is determined. This hybrid approach is illustrated in Figure 7.

The method works, as the transitions on each level of consideration are "black boxes" for the analysis approach that is applied. As a result of the mixed approach, the analyzable class of dts-nets has grown significantly. In a forthcoming paper, we will describe this approach in detail.

Acknowledgement

The authors are grateful to Fred Steutel, Eindhoven University of Technology, for his hint to use the Fast Fourier Transform.

References

VAN DER AALST W. M. P. (1992), *Timed coloured Petri nets and their application to logistics*, PhD thesis, Eindhoven University of Technology, Eindhoven.

VAN DER AALST W. M. P. (1993), Interval timed coloured petri nets and their analysis, in: Ajmone Marsan (ed.), *Application and theory of Petri nets 1993*, volume 691 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 453–472.

VAN DER AALST W. M. P. (1994), Using interval timed coloured Petri nets to calculate performance bounds, in: G. Haring and G. Kotsis (eds.), *Proceedings of the 7th International Conference of Modelling Techniques and Tools for Computer Performance Evaluation*, volume 794 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 425–444.

VAN DER AALST W. M. P. (1998), The application of Petri nets to workflow management, *The Journal of Circuits, Systems and Computers* **8**, 21–66.

VAN DER AALST W. M. P. (2000), Workflow verification: finding control-flow errors using Petri-net-

based techniques, in: *Business Process Management: Models, Techniques, and Empirical Studies*, Springer-Verlag, Berlin.

VAN DER AALST W. M. P. and ODIJK M. A. (1995), Analysis of railway stations by means of interval timed coloured Petri nets, *Real-Time Systems* **9**, 241–263.

BALBO G. and SILVA M. (eds.) (1998) *Performance models for discrete event systems with synchronisations: formalisms and analysis techniques*, Zaragoza, Kronos.

BERGSTRA J. A. and KLOP J. W. (1984), Process algebra for synchronous communication, *Information and Control* **60**, 109–137.

BERTHOMIEU B. and DIAZ M. (1991) Modelling and verification of time dependent systems using Time Petri Nets, *IEEE Transactions on Software Engineering* **17**, 259–273.

BERTHOMIEU B. and M. MENASCHE (1983), An enumerative approach for analyzing time Petri nets, in: R. E. A. Mason (ed.), *Information Processing: proceedings of the IFIP congress 1983*, volume 9 of *IFIP congress series*, Elsevier Science Publishers, Amsterdam, 41–46.

CARLIER J. and P. CHRETIENNE (1988), Timed Petri net schedules, in: G. Rozenberg (ed.), *Advances in Petri nets 1988*, volume 340 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 62–64.

CHIOLA G., C. DUTHEILLET, G. FRANCESCHINIS and S. HADDAD (1990), On well-formed coloured nets and their symbolic reachability graph, in: *Proceedings of the 11th International Conference on Applications and Theory of Petri Nets*, Paris, 307–411.

CHRETIENNE P. (1983), *Les réseaux de petri temporisés*, PhD thesis, Univ. Paris VI, Paris, 1983.

DUTHEILLET C. and S. HADDAD (1989), Regular stochastic Petri nets, in: *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets*, Bonn.

ELLIS C. A. and G. J. NUTT (1993), Modelling and enactment of workflow systems, in: Ajmone Marsan (ed.) *Application and theory of Petri nets 1993*, volume 691 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1–16.

FLORIN G. and S. NATKIN (1982), Evaluation based upon stochastic Petri nets of the maximum throughput of a full duplex protocol, in: C. Girault and W. Reisig (eds.), *Application and theory of Petri nets: selected papers from the first and the second European workshop*, volume 52 of *Informatik Fachberichte*, Berlin, Springer-Verlag, Berlin, 280–288.

GHEZZI C., D. MANDRIOLI, S. MORASCA and M. PEZZE (1991), A unified high-level Petri net formalism for time-critical systems, *IEEE Transactions on Software Engineering*, **17**, 160–172.

VAN HEE K. M. (1994), *Information system engineering: a formal approach*, Cambridge University Press.

VAN HEE K. M., L. J. SOMERS and M. VOORHOEVE (1989), Executable specifications for distributed information systems, in: E. D. Falkenberg and P. Lindgreen, *Proceedings of the IFIP TC 8/WG 8.1 Working Conference on Information System Concepts: An in-depth analysis*, Namur, Belgium, Elsevier Science Publishers, Amsterdam, 139–156.

HOLLIDAY M. A. and M. K. VERNON (1987), A Generalised timed Petri net model for performance analysis, *IEEE Transactions on Software Engineering* **13**, 1279–1310.

JABLONSKI S. and C. BUSSLER (1996), *Workflow management: modeling concepts, architecture, and implementation*, International Thomson Computer Press, London, UK.

JENSEN K. (1996), *Coloured Petri nets. Basic concepts, analysis methods and practical use*, EATCS monographs on Theoretical Computer Science, Springer-Verlag, Berlin.

LAWRENCE P., (ed.) (1997), *Workflow handbook 1997, workflow management coalition*, John Wiley and Sons, New York.

LIN C. and D. C. MARINESCU (1987), On stochastic high-Level Petri nets, in: *Proceedings of the International Workshop on Petri Nets and Performance Models*, IEEE Computer Society Press, Madison, 34–43.

AJMONE MARSAN M. (1990), Stochastic Petri nets: an elementary introduction, in: G. Rozenberg, (ed.), *Advances in Petri nets 1989*, volume 424 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1-29.

AJMONE MARSAN M., G. BALBO, A. BOBBIO, G. CHIOLA, G. CONTE and A. CUMANI. (1985), On

Petri nets with stochastic timing, in: *Proceedings of the International Workshop on Timed Petri Nets*, Torino, IEEE Computer Society Press, 80–87.

AJMONE MARSAN M., G. BALBO and G. CONTE (1984), A class of generalised stochastic Petri nets for the performance evaluation of multiprocessor systems, *ACM Transactions on Computer Systems* **2**, 93–122.

AJMONE MARSAN M., G. BALBO and G. CONTE (1986), *Performance models of multiprocessor systems*, The MIT Press, Cambridge.

AJMONE MARSAN M., G. BALBO and G. CONTE *et al* (1993), *Modelling with generalized stochastic Petri nets*, Wiley series in parallel computing. Wiley, New York.

AJMONE MARSAN M. and G. CHIOLA (1987), On Petri nets with deterministic and exponentially distributed firing times, in: G. Rozenberg (ed.), *Advances in Petri nets 1987*, volume 266 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 132–145.

MERLIN P. (1974), *A Study of the recoverability of computer systems*, PhD thesis, University of California, Irvine, California, 1974.

MERLIN P. and D. J. FABER (1976), Recoverability of communication protocols, *IEEE Transactions on Communications* **24**, 1036–1043.

MOLLOY M. K. (1981), *On the integration of delay and throughput measures in distributed processing models*, PhD thesis, University of California, Los Angeles.

MORASCA S., M. PEZZÈ and M. TRUBIAN (1991), Timed high-level nets, *The Journal of Real-Time Systems* **3**, 165–189.

MURATA T. (1989), Petri nets: properties, analysis and applications, *Proceedings of the IEEE*, **77**, 541–580.

PETRI C. A. (1962), *Kommunikation mit Automaten*, PhD thesis, Institut für instrumentelle Mathematik, Bonn.

RAMAMOORTHY C. V. and G. S. HO (1980), Performance evaluation of asynchronous concurrent systems using Petri nets, *IEEE Transactions on Software Engineering* **6**, 440–449.

RAMCHANDANI C. (1973), *Performance evaluation of asynchronous concurrent systems by timed Petri nets*. PhD thesis, Massachusetts Institute of Technology, Cambridge.

RAZOUK R. R. and PHELPS C. V. (1984), Performance analysis using timed Petri nets, in: *Proceedings of the 1984 International Conference on Parallel Processing*, IEEE Computer Society Press, Ohio, 126–128.

REISIG W. and G. ROZENBERG (eds.) (1998), *Lectures on Petri nets I: basic models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin.

REISIG W. and G. ROZENBERG, (eds.) (1998), *Lectures on Petri nets II: applications*, volume 1492 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

SIFAKIS J. (1977), Use of Petri nets for performance evaluation, in: Beilner and E. Gelenbe, *Proceedings of the Third International Symposium IFIP W.G. 7.3., Measuring, modelling and evaluating computer systems (Bonn-Bad Godesberg, 1977)*, Elsevier Science Publishers, Amsterdam.

SIFAKIS J. (1980), Performance evaluation of systems using nets, in: W. Brauer (ed.) *Net theory and applications: Proceedings of the advanced course on general net theory, processes and systems (Hamburg, 1979)*, volume 84 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 307–319.

VALETTE R. (1979), Analysis of Petri nets by stepwise refinements. *Journal of Computer and System Sciences* **18**, 35–46.

WONG C. Y., T. S. DILLON and K. E. FORWARD (1983), timed places Petri nets with stochastic representation of place time, in: *Proceedings of the International Workshop on Timed Petri Nets*, Torino, IEEE Computer Society Press (96–103).

ZENIE A. (1985), Coloured stochastic Petri nets, in: *Proceedings of the International Workshop on Timed Petri Nets*, Torino, IEEE Computer Society Press (262–271).

ZUBEREK W. M. (1980), Timed Petri nets and preliminary performance evaluation, in: *Proceedings of the 7th annual Symposium on Computer Architecture*, volume 8(3) of *Quarterly Publication of ACM Special Interest Group on Computer Architecture*, 62–82.