ACM Annual Computer Science Conference (CSC'88), Atlanta GA, 23–25 February 1988, pp.239–248.

ON GENERATION OF STATE SPACE FOR TIMED PETRI NETS

W.M. Zuberek

Department of Computer Science
Memorial University of Newfoundland
St. John's, Canada A1C–5S7

A b s t r a c t

It is shown that the behavior of timed Petri nets with deterministic firing times (M–timed nets) can be described within one uniform formalism. Moreover, for both classes of nets the state spaces are homogeneous semi-Markov chains, the stationary probabilities of states and many performance measures can thus be obtained by standard techniques developed for analysis of Markov processes. Because of sparsity of nets as well as corresponding systems of equilibrium equations, list structure representations are proposed, and a general procedure for generation of the state space is outlined to show the required processing of list structures.

1. INTRODUCTION

The major application of Petri nets [1,3,17] is modeling of systems of events in which it is possible for some events to occur concurrently but there are constraints on the precedence, concurrence or frequency of these occurrences [3,13,18]. Multiprocessor and distributed systems, communication networks and data flow architectures are just a few examples of such systems. It appears the traditional methods developed for analysis of sequential systems are inadequate for analysis of systems exhibiting concurrency and synchronization of independent, asynchronous activities. A basic Petri net, however, is not complete enough for the study of systems performance since no assumption is made on the duration of systems activities. Several concepts of timed Petri nets have been proposed by assigning firing times to the transitions and/or places of Petri nets [2,7,10,11,14,15,19].

This paper is a continuation of the approach originated by Ramchandani [14]; the firing times are thus associated with transitions of a net, and tokens are removed from the transitions' input places at the beginning of firings. It is shown that the behavior of two very different classes of timed Petri nets, D–timed and M–timed nets, can be described within one, uniform formalism with only a few class-dependent details. In D–timed Petri nets [7,14,15,19] the firing times are deterministic (or constant), i.e., there is a positive (rational) number assigned to each transition of a net which determines the "duration" of its firings. In M–timed Petri nets (or stochastic Petri nets) [2,11,21] the firing times are exponentially distributed random variables, and the corresponding firing rates are assigned to transitions of a net.

The paper also shows that the set of reachable states (or the state space) of timed Petri nets can be derived automatically from net specifications, and it outlines an algorithm that performs such a derivation. There are several possible approaches to generation of the state space. The classical approach uses matrix computations which directly implement the required steps of derivations. This approach is particularly attractive for stochastic nets since their analysis is based on the set of reachable markings which can be generated in a rather simple way. Razouk and Hirschberg [16] used bit vectors (for analysis of safe nets) to obtain very efficient implementation of their Reachability Graph Builder. In some cases [5,6,12,20] the net is evaluated through simulation rather than generation of the state space, then, however, the transient behavior may distort the stationary results, and there is no guarantee that the events with small or very small probabilities will occur even in very long runs. The approach presented in this paper is based on the observation that the nets as well as their state graphs are usually very sparse; in nets with tens or hundreds of places and transitions, at each moment of time there are only a few "active" places and transitions, i.e., only a few elements that "need attention"; the remaining part of the net remains inactive and its description does not change. List structures are proposed to take advantage of this sparsity. It may seem that operations on such structures are much more complex than similar operations performed on vectors. This is certainly true, it should be observed, however, that these more complex operations are performed on selected elements only, and that the sparsity of processed elements usually more than compensates for the increased complexity of operations.

This paper is organized in 6 main sections. Section 2 recalls the basic concepts for marked Petri nets. Formal descriptions of D–timed nets and M–timed nets are given in sections 3 and 4, respectively. Section 5 presents the list structure representation of nets in the TPNEV package for evaluation of timed Petri nets, and section 6 contains an outline of list processing routines that are used for generation of set of reachable states. Some simple examples of D–timed and M–timed nets are given in section 7.

Note: This version of the paper is derived from the original text by using a different text processing system. A few minor corrections have been made during the reformatting.

2. MARKED PETRI NETS

A (generalized) Petri net $\mathbf{N}$ is a 6-tuple $\mathbf{N} = (P, T, A, w, B, C)$ where:

P is a finite, nonempty set of places,

T is a finite, nonempty set of transitions,

A is a set of directed arcs which connect places with transitions and transitions with places, $A \subseteq P \times T \cup T \times P$,

w is a weight function which assigns a positive integer "weight" to each arc of the net, $w : A \to \{1, 2, ...\}$,

B is a (possibly empty) set of inhibitor arcs, $B \subset P \times T$, and A and B are disjoint sets,

C is a (possibly empty) set of interrupt arcs, $C \subseteq B$.

medskip It should be noted that the distinction between interrupt and inhibitor arcs becomes important for times nets, however, it is convenient to introduce these two types of arcs from the very beginning. For ordinary nets (i.e., nets without time), interrupt arcs are just inhibitor arcs.

As usual, *Inp(p), Out(p), Inp(t), Out(t), Inh(t)* and *Int(t)* denote the sets of input and output transitions of a place p, the sets of input and output places of a transition t, and the sets of inhibiting and interrupting places of t, respectively. The notation is extended in the usual way to sets of places and transitions.

A place p is shared iff it is an input place for more than one transition. A shared place p is guarded iff for each two different transitions sharing p there exists another place $p \subset k$ which is in the input set of one and in the inhibitor set of the other of these two transitions. A place p is free-choice iff the input sets of transitions sharing p are identical and the weights of corresponding arcs are equal. A net is free-choice iff each shared place is either guarded or free-choice.

The relation of "sharing a free-choice place" is in fact an equivalence relation in the set of transitions T, hence it determines a partition of T into a set of free-choice equivalence classes denoted by $Free(T) = \{T_1, T_2, ..., T_k\}$.

A marked generalized Petri net $\mathbf{M}$ is a pair $\mathbf{M} = (\mathbf{N}, m_0)$ where:

$\mathbf{N}$ is a generalized Petri net, $\mathbf{N} = (P, T, A, w, B, C)$,

m_0 is the initial marking function, $m_0 : P \to \{0, 1, ...\}$.

Let any function $m : P \to \{0, 1, ...\}$ be called a marking of a net $\mathbf{N} = (P, T, A, w, B, C)$.

A transition t is enabled by a marking m iff every t's input place $p \in Inp(t)$ contains at least $w(p, t)$ tokens, i.e., $m(p) \geq w(p, t)$, and every t's inhibiting place $p \in Inh(t)$ contains zero tokens. The set of all transitions enabled by m is denoted by $En(m)$.

Every transition enabled by a marking m can fire. When an enabled transition t fires, tokens are removed from t's input places in numbers corresponding to the weights of input arcs, and similarly, the weights of t's output arcs determine the numbers of tokens added to output places. A firing of an enabled transitions is thus a transformation of marking functions. A marking m_j is directly reachable (or t_k–reachable) from a marking m_i iff there exists a transition $t_k \in En(m_i)$, such that

$$\forall(p \in P) m_j(p) = \begin{cases} m_i(p) - w(p, t_k), \text{ if } p \in Inp(t_k) - Out(t_k), \\ m_i(p) + w(p, t_k), \text{ if } p \in Out(t_k) - Inp(t_k), \\ m_i(p) - w(p, t_k) + w(t_k, p), \\ \qquad \text{if } p \in Inp(t_k) \cap Out(t_k), \\ m_i(p), \text{ otherwise.} \end{cases}$$

Since the firings in free–choice equivalence classes are selected in a random way, it is convenient to describe all possibilities of different firings as a function of the marking m.

A selection function of a marking m in a net $\mathbf{N}$ is any function $g : T \to \{0, 1, ...\}$ such that:

(1) there exists a sequence of transitions $u = (t_{i_1}, t_{i_2}, ..., t_{i_k})$ in which $t_{i_j} \in En(m_{i_{j-1}})$ for $j = 1, ...k$ and for $m_{i_0} = m$, where

$$\forall(p \in P) m_{i_j}(p) = m_{i_{j-1}}(p) \; - \begin{cases} w(p, t_{i_j}), \text{ if } p \in Inp(t_{i_j}), \\ 0, \text{ otherwise,} \end{cases}$$

(2) the set of transitions enabled by m_{i_k}, $En(m_{i_k})$, is empty,

(3) for each $t \in T$ the number of occurrences of t in the sequence u is equal to $g(t)$.

The set of all selection functions of a marking m is denoted by $Sel(m)$.

3. D–TIMED PETRI NETS

In timed Petri nets, each transition t takes a positive time to fire. When a transition t is enabled, a firing can be initiated by removing tokens from t's input places. The tokens remain in the transition t for the "firing time", and then the firing terminates by adding tokens to each of t's output places. Each of the firings is initiated in the same instant of time in which it is enabled; timed nets correspond thus to "maximally concurrent evolutions" in nets [8]. If a transition is enabled while it fires, a new, independent firing can be initiated.

It should be observed that the mechanism of firing in timed Petri nets is quite different from that in stochastic nets [2,11]; consequently these two classes of Petri nets have rather different properties.

In timed Petri nets with interrupt arcs, a firing of a transition can be discontinued. If, during a firing period of a transition t, at least one of t's interrupting places becomes nonempty (i.e., it receives a token as a result of a termination of another firing), the firing of t ceases and the tokens removed from t's input places at the beginning of firing, are returned to their original places.

A net $\mathbf{N} = (P, T, A, w, B, C)$ is simple if input sets of transitions with nonempty interrupting sets do not contain interrupting places

$$\forall(t \in T) \; Int(t) = \Phi \vee Int(Inp(t)) = \Phi$$

where Φ denotes the empty set. Simple nets do not allow "propagation" of interrupts when an interrupted transition, through its input places, interrupts another transition. Only simple timed Petri nets are considered in this paper since in most practical cases this is sufficient; non-simple nets can be described in a very similar way, with a minor modification of state–transition formulas.

In D–timed Petri nets [22] the firing times of transitions are deterministic (or constant), i.e., there is a nonnegative number assigned to each transition of a net which determines the duration of transition's firings. Since the memoryless property does not apply to such nets, the state description must explicitly include the "history" of firings.

A D–timed Petri net $\mathbf{T}$ is a triple $\mathbf{T} = (\mathbf{M}, c, f)$ where:

$\mathbf{M}$ is a free–choice marked Petri net, $\mathbf{M} = (\mathbf{N}, m_0)$, $\mathbf{N} = (P, T, A, w, B, C)$,

c is a choice function which assigns a "free–choice" probability to each transition of a net in such a way that

$$\forall(T_i \in Free(T))\ \sum_{t \in T_i} c(t) = 1,$$

f is a firing time function which assigns a nonnegative real number $f(t)$ to each transition t of the net, $f : T \to \mathbf{R}^{\oplus}$ and $\mathbf{R}^{\oplus}$ denotes the set of nonnegative real numbers.

Since in timed nets tokens are distributed in places as well as in (firing) transitions, the "state" of a timed net is defined as a triple of functions, one describes the distribution of tokens in places, the second distribution of tokens in (firing) transitions, and the third one provides a "history" of transition's firings.

A state s of a D–timed Petri net $\mathbf{T}$ is a triple $s = (m, n, r)$ where:

m is a marking function, $m : P \to \{0, 1, ...\}$,

n is a firing–rank function, $n : T \to \{0, 1, ...\}$,

r is a remaining–firing–time function which assigns the remaining firing time to each independent firing (if any) of a transition, i.e., if the firing rank of a transition t is equal to k, $n(t) = k$, the remaining–firing–time function $r(t)$ is a vector of k nonnegative nondecreasing real numbers denoted by $r(t)[1]$, $r(t)[2]$, ..., $r(t)[k]$; r is a partial function and it is undefined for all those transitions t for which $n(t) = 0$.

An initial state s_i of a net $\mathbf{T}$ is a triple $s_i = (m_i, n_i, r_i)$ where n_i is a selection function from the set $Sel(m_0)$, $n_i \in Sel(m_0)$, the remaining–firing–time function is equal to the firing times $f(t)$ for all those transitions t for which $n_i > 0$

$$\forall(t \in T)\ r_i(t)[k] \ = \ \begin{cases} f(t), \text{ if } n_i(t) > 0 \wedge 1 \le k \le n_i(t), \\ \text{undefined, otherwise;} \end{cases}$$

and the marking m_i is defined as

$$\forall(p \in P)\ m_i(p) \ = \ m_0(p) \sum_{t \in Out(p)} w(p,t) * n_i(t).$$

A free-choice net $\mathbf{T}$ may have several different initial states.

A state $s_j = (m_j, n_j, r_j)$ is directly reachable (or g_k–reachable) from the state $s_i = (m_i, n_i, r_i)$, iff:

(1) $g_k \in Sel(m_{ioj})$,

(2) $\forall(p \in P)\ m_j(p) \ = \ m_{ioj}(p) - \sum_{t \in Out(p)} g_k(t) * w(p,t)$,

(3) $\forall(t \in T)\ n_j(t) \ = \ n_i(t) - e_i(t) - d_{ij}(t) + g_k(t)$,

(4) $\forall(t \in T)\ r_j(t)[l] \ = \ \begin{cases} r_i(t)[l + e_i(t) + d_{ij}(t)] - h_i, \text{ if} \\ \quad 1 \le l \le n_i(t) - e_i(t) - d_{ij}(t), \\ f(t), \text{ if } g_k(t) > 0 \wedge n_i(t) - \\ \quad e_i(t) - d_{ij}(t) < l \le n_j(t), \end{cases}$

where:

(5) $\forall(p \in P)\ m_{ioj}(p) \ = \ m_{io}(p) + \sum_{t \in Out(p)} d_{ij}(t) * w(p,t)$,

(6) $\forall(p \in P)\ m_{io}(p) \ = \ m_i(p) + \sum_{t \in Inp(p)} e_i(t) * w(p,t)$,

(7) $\forall(t \in T)\ d_{ij}(t) \ = \ \min(n_i(t) - e_i(t), \sum_{p \in Int(t)} m_{io}(p))$,

(8) $\forall(t \in T)\ e_i(t) \ = \ \begin{cases} l, \text{ if } n_i(t) \le l \wedge r_i(t)[1] = ... \\ ... = r_i(t)[l] = h_i \wedge (n_i(t) = l \vee \\ \quad r_i(t)[l+1] > h_i), \\ 0, \text{ otherwise;} \end{cases}$

(9) $h_i \ = \ \min_{t \in T \wedge n_i(t) > 0}(r_i(t)[1])$.

A state s_j is (generally) reachable from a state s_i if there is a sequence of directly reachable states from the state s_i to the state s_j. A set $S(\mathbf{T})$ of reachable states is defined as the set of all states of a net $\mathbf{T}$ which are reachable from the initial states of the net $\mathbf{T}$.

A state graph $\mathbf{G}$ of a D–timed Petri net $\mathbf{T}$ is a labeled directed graph $\mathbf{G}(\mathbf{T}) = (V, D, h, q)$ where:

V is a set of vertices which is equal to the set of reachable states of the net $\mathbf{T}$, $V = S(\mathbf{T})$,

D is a set of directed arcs, $D \subset V \times V$, such that (s_i, s_j) is in D iff s_j is directly reachable from s_i,

h is a node labeling function which assigns the holding time of a state s to each vertex $s \in V$, $h : V \to \mathbf{R}^{\oplus}$, in such a way that if $s = (m, n, r)$ then

$$h(s) \ = \ \min_{t \in T \wedge n(t) > 0}(r(t)[1]),$$

q is an arc labeling function which assigns the probability of transition from s_i to s_j to each arc (s_i, s_j) in the set D, $q : D \to [0, 1]$, in such a way that if s_j is g_k–reachable from s_i, then

$$q(s_i, s_j) \ = \ \prod_{T_z \in Free(T)} a(T_z, g_k) \prod_{t \in T_z} c(t)^{g_k(t)},$$

where the coefficient $a(T_z, g_k)$ describes the number of ways in which the selection function g_k can be realized in a free–choice class $T_z \in Free(T)$. It can be determined as follows. Let an n–argument function ψ be defined recursively:

(1) $\psi(0, 0, ..., 0) = 1$,

$$(2)\ \psi(k_1,k_2,...,k_n) = \sum_{1\le i\le n} \begin{cases} \psi(k_1,...,k_i-1,...,k_n), \\ \qquad\qquad \text{if } k_i>0, \\ 0,\ \text{if } k_i=0. \end{cases}$$

Then, for the class $T_z=\{t_{z_1},t_{z_2},...,t_{z_n}\}$:

$$a(T_z,g) \ = \ \psi(g(t_{z_1}),g(t_{z_2}),...,g(t_{z_n})),$$

and, for any marking m

$$\sum_{g\in Sel(m)} \prod_{T_z\in Free(T)} a(T_z,g) \prod_{t\in T_z} c(t)^{g(t)} \ = \ 1.$$

It should be observed that state graphs of free–choice Petri nets are discrete–time homogeneous semi–Markov processes, the stationary probabilities of the states can thus be obtained in the standard way [9], and then operational analysis [4] can be used to evaluate the performance of a modeled system.

4. M–TIMED PETRI NETS

In M–timed Petri nets [21] the firing times of transitions are exponentially distributed random variables, and their corresponding rates are assigned to transitions of a net. Because of the memoryless property of the exponential distribution, the "history" of firings is immaterial, and this simplifies the description of states and state transitions.

An M–timed Petri net $\mathbf{T}$ is a triple $\mathbf{T}=(\mathbf{M},c,f)$ where:

$\mathbf{M}$ is a free–choice marked Petri net, $\mathbf{M}=(\mathbf{N},m_0)$, $\mathbf{N}=(P,T,A,w,B,C)$,

c is a choice function which assigns a "free–choice" probability to each transition of a net in such a way that

$$\forall(T_i\in Free(T))\sum_{t\in T_i} c(t) \ = \ 1,$$

f is a firing rate function which assigns the rate of firing $f(t)$ to each transition t of the net, $f:T\to\mathbf{R}^+$, and $\mathbf{R}^+$ denotes the set of positive real numbers; the firing time of a transition t is a random variable $x(t)$ with the distribution function

$$\text{Prob}[x(t)>y] \ = \ e^{-f(t)*y},\ y>0.$$

A state s of an M–timed Petri net $\mathbf{T}$ is a pair $s=(m,n)$ where:

m is a marking function, $m:P\to\{0,1,...\}$,

n is a firing–rank function which indicates (for each transition of the net) the number of active firings, i.e., the number of firings which have been initiated but are not yet terminated, $n:T\to\{0,1,...\}$.

An initial state s_i of a free–choice net $\mathbf{T}$ is a pair $s_i=(m_i,n_i)$ where n_i is a selection function from the set $Sel(m_0)$, $n_i\in Sel(m_0)$, and the marking m_i is defined by

$$\forall(p\in P)\ m_i(p) \ = \ m_0(p)\sum_{t\in Out(p)} n_i(T)*w(p,t).$$

A free–choice net $\mathbf{T}$ may have several different initial states.

A state $s_j=(m_j,n_j)$ is directly reachable (or (t_k,g_l)–reachable) from the state $s_i=(m_i,n_i)$ iff:

(1) $n_i(t_k) \ > \ 0$,
(2) $g_l \ \in \ Sel(m_{ikj})$,
(3) $\forall(p\in P)\ m_j(p) \ = \ m_{ikj}(p) - \sum_{t\in Out(p)} g_l(t)*w(p,t)$,
(4) $\forall(t\in T)\ n_j(t) \ = \ n_i(t)-e_i(t)- \ d_{ij}(t)+g_l(t)$,

where

(5) $\forall(p\in P)\ m_{ikj}(p) \ = \ m_{ik}(p)+\sum_{t\in Out(p)} d_{ij}(t)*w(p,t)$,
(6) $\forall(p\in P)\ m_{ik}(p) \ = \ m_i(p)+\sum_{t\in Inp(p)} e_i(t)*w(t,p)$,
(7) $\forall(t\in T)\ d_{ij}(t) \ = \ \min(n_i(t)-e_i(t),\ \sum_{p\in Int(t)} m_{ik}(p))$,
(8) $\forall(t\in T)\ e_i(t) \ = \ \begin{cases} 1, & \text{if } t=t_k, \\ 0, & \text{otherwise.} \end{cases}$

Similarly as for D–timed nets, a state s_j is (generally) reachable from a state s_i if there is a sequence of directly reachable states from the state s_i to the state s_j. Also, a set $S(\mathbf{T})$ of reachable states is defined as the set of all those states which are reachable from the initial states of the net $\mathbf{T}$.

A state graph $\mathbf{G}$ of an M–timed Petri net $\mathbf{T}$ is a labeled directed graph $\mathbf{G}(\mathbf{T})=(V,D,h,q)$ where:

V is a set of vertices which is equal to the set of reachable states of the net $\mathbf{T}$, $V=S(\mathbf{T})$,

D is a set of directed arcs, $D\subseteq V\times V$, such that (s_i,s_j) is in D iff s_j is directly reachable from s_i in $\mathbf{T}$,

h is a node labeling function which assigns the average holding time of a state s to each vertex $s\in V$, $h: V\to\mathbf{R}^{\oplus}$, in such a way that if $s=(m,n)$ then

$$h(s) \ = \ 1/\sum_{t\in T} f(t)*n(t),$$

q is an arc labeling function which assigns the probability of transitions from s_i to s_j to each arc (s_i,s_j) in the set D, $q:D\to[0,1]$, in such a way that if s_j is t_k,g_l–reachable from s_i, then

$$q(s_i,s_j) = h(s_i)*f(t_k)*n_i(t_k)\prod_{T_z\in Free(T)} a(T_z,g_l)\prod_{t\in T_z} c(t)^{g_l(t)}.$$

Since for M–timed nets the holding times are exponentially distributed, state graphs of free-choice M–timed Petri nets are continuous-time homogeneous Markov chains.

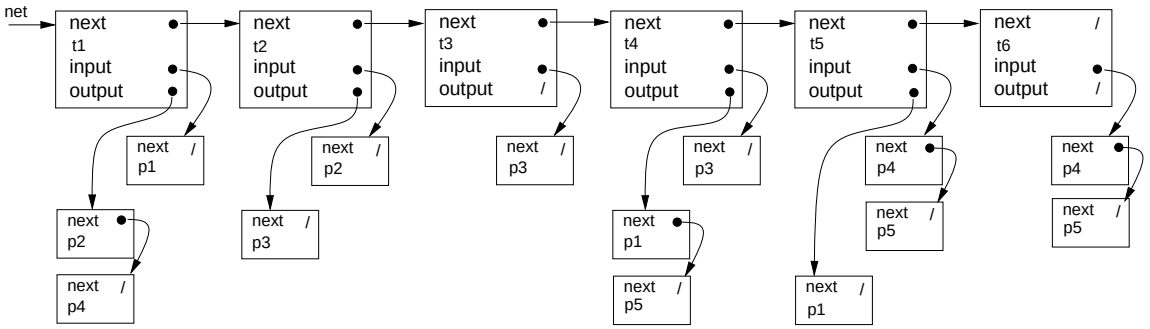

Fig.2. List representation of $\mathbf{T}_1$.

5. NET REPRESENTATION

Existing programs that perform analysis and evaluation of timed (and stochastic) Petri nets use quite different external specifications of nets [12,15,20]. Net description assumed in the TPNEV program [23] is 'transition oriented', i.e., Petri nets are specified as lists of transitions, and each transition contains all parameters associated with it. The syntax of net specifications in BNF notation is as follows:

```
<net> ::= Mnet ( <transitions> )
        |  Dnet ( <transitions> )
<transitions> ::= <transition>
        |  <transition> ; <transitions>
<transition> ::= <t-header> = <input-output-list>
<input-output-list> ::= <input-list>
        |  <input-list> / <output-list>
<input-list> ::= <arc> | <arc> , <input-list>
<output-list> ::= <arc> | <arc> , <output-list>
<arc> ::= <place> | <place> : <weight> | <place> -
<place> ::= <integer> | <name>
<weight> ::= <integer>
<t-header> ::= <t-ident> <t-time> <t-prob>
<t-ident> ::= # <integer> | # <name>
<t-time> ::= * <rational> | <empty>
<t-prob> ::= , <rational> | , <integer> / <integer>
        |  <empty>
<rational> ::= <integer> | <integer> . <integer>
```

Similarly, the initial marking function is specified as a list of marked places:

```
<imarking> ::= mark ( <marking list> )
<marking list> ::= <marked place>
        |  <marked place> , <marking list>
<marked place> ::= <place> | <place> : <count>
<count> ::= <integer>
```

Example: TPNEV description of the D–timed net shown in Fig.1 (as usual, places are represented by circles, transitions by bars, inhibitor and interrupt arcs by small circles and dots instead of arrowheads, respectively, the initial marking function is represented by dots inside circles, and the firing as well as the choice functions are given as additional descriptions of transitions) is as follows:

```
Dnet (#1*0=1/2,4;
      #2*10=2/3;
      #3*0,0.1=3;
      #4*5,0.9=3/1,5;
      #5*20=4,5-/1;
      #6*0=4,5 )
mark ( 1 )
```

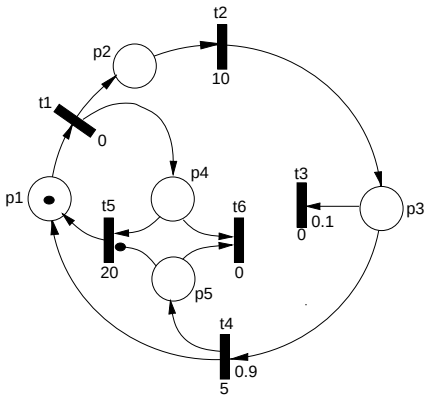

Fig.1. D–timed Petri net $\mathbf{T}_1$.

The internal list structure created by TPNEV during processing of this description is sketched in Fig.2, and Fig.3 shows internal representation of a single transition with a few more details. The weights of arcs are used to indicate generalized (or multiple) arcs. Inhibitor arcs are indicated by nonpositive weights, with interrupt arcs represented by negative weights, and inhibitor non-interrupting arcs by zeros.

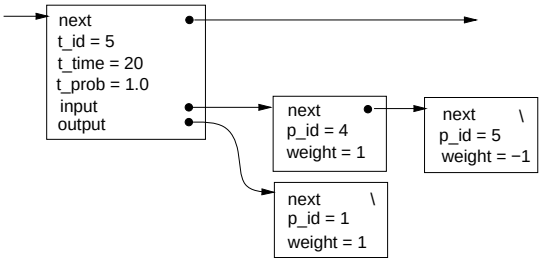

Fig.3. Internal representation of t_5.

6. STATE SPACE GENERATION

In the following descriptions, a Pascal–like programming notation is used, however, control structures are slightly modified by explicit delimiters:

if <condition> **then** <statement list> **fi**
if <condition> **then** <statement list>
else <statement list> **fi**
while <condition> **do** <statement list> **od**

Moreover, type consistency rules are assumed to be quite flexible, and in many cases generic procedures are given in order to simplify the description. Also, structured objects enclosed by braces "{" and "}" are used more freely than in (existing) Pascal-like languages.

States of a timed net are pairs of functions "mlist,flist" where "mlist" represents the marking function and "flist" describes the firing transitions; for M–timed nets, "flist" represents the firing–rank function, while for D–timed nets it also describes the remaining–firing–time (or shortly, "rft") function.

The graph of reachable states is generated by the following procedure "stategraph(net,imark,sgraph)" where "net" points to a list of transitions, "imark" points to a list of marked places representing the initial marking function, and "graph" returns the list of reachable states:

```
procedure stategraph (net,imark,sgraph);
begin selectset(net,imark,nil,sgraph);
  nlist := sgraph;
  while nlist ≠ nil do
    endfiring(nlist↑.mlist,nlist↑.flist,elist);
    while elist ≠ nil do
      interrupt(elist↑.mlist,elist↑.flist,mlist,flist);
      selectset(net,mlist,flist,slist);
      while slist ≠ nil do
        search(sgraph,slist↑.mlist,slist↑.flist,node);
        createarc(nlist↑.link,node);
        slist := slist↑.next od;
      elist := elist↑.next od;
    nlist := nlist↑.next od
end;
```

in which "endfiring" creates a list "elist" of intermediate states that correspond to terminations of the "next" firings, "interrupt" performs all interruptions of firing transitions and creates an updated state "mlist,flist", the invocation of "selectset" determines the set of states "slist" which correspond to different selection functions applied to the state "mlist,flist" in the net "net", "search(list,elem,point)" checks if the element "elem" is in the list "list", and appends it to the list if it is a new element while "point" returns a pointer to (appended or existing) element, and "createarc(node1,node2)" creates a labeled arc from "node1" to "node2" (each node "node" contains a list of arcs "node↑.link").

The procedure "selectset" determines the set of selection functions and creates a set "slist" of (intermediate) states which are obtained by applications of selection functions to the state "mlist,flist" in the net "net":

```
procedure selectset (net,mlist,flist,slist);
begin slist := nil;
  nlist := nil;
  append(mlist,flist,nlist);
  while nlist ≠ nil do
    enabled(net,nlist↑.mlist,tlist);
    if tlist=nil then
      append(nlist↑.mlist,nlist↑.flist,slist)
    else while tlist ≠ nil do
      initiate(tlist↑.trans,nlist↑.mlist,nlist↑.flist,ml,fl);
      search(nlist,ml,fl,point);
      tlist := tlist↑.next od fi;
    nlist := nlist↑.next od;
  release(nlist)
end;
```

where a generic procedure "append(elem,list)" appends the element "elem" to a list "list".

The procedure "enabled" creates a list "tlist" of transitions enabled by the marking function "mlist" in the net "net":

```
procedure enabled (net,mlist,tlist);
begin tlist := nil;
  t := net;
  while t ≠ nil do
    plist := t↑.input;
    b := true;
    while b and (plist ≠ nil) do
      val := plist↑.weight;
      if (val > 0 and marked(mlist,plist↑.id,val)) or
        (val ≤ 0 and not marked(mlist,plist↑ .id,0))
      then plist := plist↑.next else b := false fi od;
    if b then include(t↑.trans↑.id,1,tlist) fi;
    t :=t↑.next od
end;
```

where the boolean procedure "marked" is true if the marking "mlist" assigns at least "val" tokens to the place "plist↑.id" is false otherwise, and the generic procedure "include(elem,val,list)" checks if the element "elem" exists in the list "list" and either increases its count by "val" or appends it to this list with count set to "val".

The procedure "initiate" performs initiating of a new firing of the transition "trans" converting "m1list" and "f1list" into "m2list" and "f2list":

```
procedure initiate (trans,m1list,f1list,m2list,f2list);
begin copy(m1list,m2list);
  copy(f1list,f2list);
  plist := trans↑.input;
  while plist ≠ nil do
    if plist↑.weight > 0 then
      reduce(plist↑.id,plist↑.weight,m2list) fi;
    plist := plist↑.next od;
  include(trans↑.id,1,f2list)
end;
```

The procedure "reduce(elem,val,list)" deletes the element "elem" from a list "list" if the corresponding count is equal to "val", otherwise it subtracts "val" from the count associated with "elem" in "list".

The procedure "endfiring" creates a list of (intermediate) states which correspond to the termination of the "next" firings in the state "mlist,flist"; this procedure depends upon the type of timed nets; for D–timed nets it always returns a one-element list which contains a state obtained by termination of all those firings for which the remaining firing time is equal to the smallests remaining firing time, while for M–timed nets it returns a list of states that correspond to all firing transitions (since for exponentially distributed firing times, each of the firing transition can terminate its firing as the "next" one with a corresponding probability):

```
procedure endfiring (mlist,flist,slist);
begin slist := nil;
  if netclass = 'M' then
    while flist ≠ nil do
      copy(mlist,ml);
      copy(flist,fl);
      terminate(flist↑.trans,ml,fl);
      append(ml,fl,slist);
      flist := flist↑.next od
  else if netclass='D' then
    findmintime(flist,x);
    copy(mlist,ml);
    copy(flist,fl);
    while flist≠ nil do
      if flist↑.rft=x then
        terminate(flist↑.trans,ml,fl) fi;
      flist := flist↑.next od;
    f := fl;
    while f ≠ nil do
      f↑.rft := f↑.rft-x;
      f := f↑.next od;
  append(ml,fl,slist) fi fi
end;
```

where "findmintime" returns in "x" the smallest remaining firing time "rft" from the list "list", and the procedure "terminate" terminates a single firing of the transition "trans" modifying the marking function "mlist" and the list of firing transitions "flist" accordingly:

```
procedure terminate (trans,mlist,flist);
begin reduce(trans↑.id,1,flist);
  plist := trans↑.output;
  while plist ≠ nil do
    include(plist↑.id,plist↑.weight,mlist);
    plist := plist↑.next od
end;
```

Finally, the procedure "interrupt" performs all interrupts (of firing transitions) and updates the marking and firing functions "m1list" and "f1list" into "m2list" and "f2list", respectively; the processing is performed in two steps, first step determines the numbers of interrupted firings (as a list "ilist"), and the second step updates "m2list" and "f2list" accordingly:

```
procedure interrupt (m1list,f1list,m2list,f2list);
begin ilist := nil;
  f := flist;
  while f ≠ nil do
    plist := f↑.trans↑.input;
    num := 0;
    while plist ≠ nil do
      if plist↑.weight < 0 then
        add(num,mlist,plist↑.id) fi;
      plist := plist↑.next od;
    num := min(num,f↑.count);
    append(num,f↑a.trans,ilist);
    f := f↑.next od;
  copy(m1list,m2list);
  copy(f1list,f2list);
  while ilist≠ nil do
    reduce(ilist↑.trans↑.id,ilist↑.num,f2list);
    plist := ilist↑.trans↑.input;
    while plist≠ nil do
      if plist↑.weight > 0 then
      include(plist↑.id,ilist↑.num∗plist↑.weight,m2list) fi;
      plist := plist↑.next od;
    ilist := ilist↑.next od;
  release(ilist)
end;
```

where the procedure "add(num,mlist,place)" increases "num" by the count associated with the place "place" in the list "mlist"; for each transition of a net, "num" determines the number of firings to be interrupted.

7. EXAMPLES

Since state graphs of timed Petri nets are (discrete or continuous time) homogeneous semi–Markov processes, the stationary probabilities $x(s)$ of states $s \in S(\mathbf{T})$ can be obtained by solving the following system of simultaneous equations:

$$\begin{cases} \sum\limits_{1\le j\le K} h(s_j) * q(s_j, s_i) * x(s_j) \;=\; h(s_i) * x(s_i); \\ \qquad\qquad\qquad\qquad\qquad\qquad i = 1, ..., K-1 \\ \sum\limits_{1\le i\le N} x(s_i) \;=\; 1 \end{cases}$$

where K is the number of states in the (finite) set $S(\mathbf{T})$.

Stationary probabilities of states are used in derivations of many performance measures [4,9], for example utilization factors, throughput rates, response and waiting times, etc.

The D–timed net shown in Fig.1 is a model of a very simple protocol in which messages sent from the sender (place p_1) to the receiver (place p_3) are confirmed by acknowledgements sent back to sender (in the loop p_1, t_1, p_2, t_2, p_3 and t_4). There is nonzero probability that the system can lose a message or an acknowledgement; the place p_3 is a free–choice place and the transition t_3 models a message/acknowledgement "sink"; the probability associated with t_3, $c(t_3)$, represents thus the probability of incorrect transfer, i.e., the probability of losing a message or an acknowledgement in the system. A "timeout" is used to recover from incorrect transfers. It works in the following way. An event of "sending a message" is modeled by t_1. When it fires, single tokens are deposited in p_2 (a "message") and p_4 (a "timeout"). A token in p_4 immediately starts a firing of the "timeout" transition t_5 (since p_5 is "empty"). The firing time of t_5 is large enough to allow the transfer of a message and an acknowledgement. If there is no loss of tokens (i.e., if t_4 is selected for firing according to its probability), the transition t_4 will finish its firing before t_5, and then a token in p_6 interrupts t_5 and "cancels" the timeout by firing t_6 (t_6 is another token "sink"). If, however, a message or an acknowledgement gets lost (i.e., if t_3 is selected for firing instead of t_4), the timeout t_5 terminates its firing without interruption, and "regenerates" the lost token in p_1 which is retransmitted to the receiver.

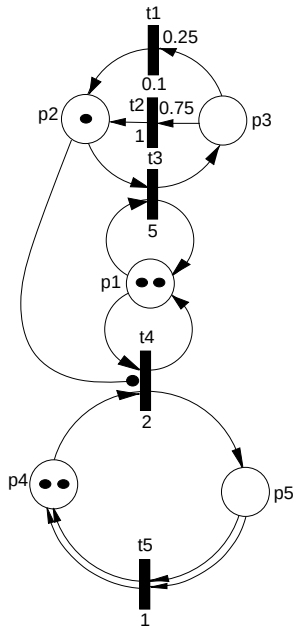

Fig.4. M–timed Petri net $\mathbf{T}_2$.

Tab.2. The set of reachable states for $\mathbf{T}_2$.

s_i	$x(s_i)$	m_i 1 2 3 4 5	n_i 1 2 3 4 5	$h(s_i)$	t_k	g_l 1 2 3 4 5	s_j	$q(s_i,s_j)$
1	0.0108	0 0 0 1 0	0 0 1 1 0	0.143	3	0 1 0 1 0	2	0.536
						1 0 0 1 0	3	0.179
					4	0 0 0 1 0	4	0.286
2	0.0326	0 0 0 0 0	0 1 0 2 0	0.200	2	0 0 1 1 0	1	0.200
					4	0 0 0 0 0	5	0.800
3	0.1041	0 0 0 0 0	1 0 0 2 0	0.244	1	0 0 1 1 0	1	0.024
					4	0 0 0 0 0	6	0.976
4	0.0149	0 0 0 0 1	0 0 1 1 0	0.143	3	0 1 0 0 0	5	0.536
						1 0 0 0 0	6	0.179
					4	0 0 0 0 1	7	0.286
5	0.0621	1 0 0 0 1	0 1 0 1 0	0.333	2	0 0 1 1 0	4	0.333
					4	0 0 0 0 1	8	0.667
6	0.2072	1 0 0 0 1	1 0 0 1 0	0.476	1	0 0 1 1 0	4	0.048
					4	0 0 0 0 1	9	0.952
7	0.0323	1 0 0 0 0	0 0 1 0 1	0.167	3	0 1 0 0 0	8	0.625
						1 0 0 0 0	9	0.208
					5	0 0 0 1 0	1	0.167
8	0.1227	2 0 0 0 0	0 1 0 0 1	0.500	2	0 0 1 0 0	7	0.500
					5	0 0 0 2 0	2	0.500
9	0.4134	2 0 0 0 0	1 0 0 0 1	0.909	1	0 0 1 0 0	7	0.091
					5	0 0 0 2 0	3	0.909

The derivation of the state space for $\mathbf{T}_1$ is shown in Tab.1, which also shows the stationary probabilities of states $x(s)$, $s \in S(\mathbf{T}_1)$. Since some of the firing times are equal to zero (e.g., $f(t_1) = 0$ which means that t_1 fires instantaneously), the holding times of states in which such transitions fire are equal to zero, and consequently, the stationary probabilities of such states are also equal to zero. The stationary probabilities can be used for evaluation of the performance of the system modeled by $\mathbf{T}_1$. For example, the throughput rate (i.e., the average number of correct message transfers in a time unit) can be obtained from the *utilization* of t_4 (t_4 models transfer of an acknowledgement, which is required for each correct transfer); this utilization is equal to $x(s_3) = 0.290$ (since $n_i(t_4) > 0$ only in state s_3 (see Tab.1). The throughput rate is equal to $0.290/f(t_4) = 0.059$ messages per time unit (or 58 messages per 1000 time units).

Tab.1. The set of reachable states for $\mathbf{T}_1$.

s_i	$x(s_i)$	m_i 1 2 3 4 5	n_i 1 2 3 4 5 6	$h(s_i)$	s_j	$q(s_i,s_j)$
1	0.000	0 0 0 0 0	1 0 0 0 0 0	0.0	2	1.00
2	0.645	0 0 0 0 0	0 1 0 0 1 0	10.0	3	0.90
					4	0.10
3	0.290	0 0 0 0 0	0 0 0 1 1 0	5.0	5	1.00
4	0.000	0 0 0 0 0	0 0 1 0 1 0	0.0	6	1.00
5	0.000	0 0 0 0 0	1 0 0 0 0 1	0.0	2	1.00
6	0.065	0 0 0 0 0	0 0 0 0 1 0	10.0	1	1.00

The M–timed Petri net shown in Fig.4 is a model of an interactive system with two classes of users (and jobs) and preemptive scheduling policy. The system consists of a (multiprocessor) central server with a queue of waiting jobs, n_1 terminals in class–1 and n_2 class–2 terminals. The class–1 terminals submit one job at a time, while class–2 terminals submitt two jobs at each arrival instant, and processing of both jobs must be completed before another "thinking" (or terminal) phase. Thinking times for class–2 terminals are exponentially distributed with the average of 1 time unit, while thinking times of class–1 terminals are hyperexponentially distributed, and the average is equal to 10 time units with probability 0.25, and 1 time unit with probability 0.75. All service times are exponentially distributed. Moreover, the class–1 jobs have higher priority to use the processors, so if a class–1 job arrives when there in no processor available, processing of one of class–2 jobs is interrupted and the processor is preempted to start processing of the new class–1 job. The central server is modeled by p_1, t_3 and t_4; the transitions t_3 and t_4 correspond to the central server processing class–1 and class–2 jobs, respectively, with service rates (or the firing rates) equal to 5 and 2, respectively. The place p_1 with it initial number of tokens represents the number of (available) processors, in this case 2. The places p_2 and p_4 model queues of waiting jobs (for class–1 and class–2, respectively). The interrupt arc (p_2, t_4) interrupts t_4 whenever a class–1 job arrives to its waiting queue (i.e., p_2). The transitions t_1, t_2 and t_5 models the class–1 and class–2 terminal times. The initial number of tokens in places p_2 and p_3 represents the number of terminals in class–1, n_1, and the initial number of tokens assigned to p_4 and p_5 determines the number of class–2 terminals, n_2.

For $n_1 = 1$ and $n_2 = 2$, the derivation of the set $S(\mathbf{T}_2)$ is given in Tab.2. As before, many performance measures can be derived from stationary probabilities of the states. For example, since the server is idle only in the states s_8 and s_9 ($m_8(p_1) = m_9(p_1) = 2$), the stationary probability that the system is idle is equal to the sum $x(s_8) + x(s_9) = 0.536$ (see Tab.2). Moreover, the average utilization of each processor is equal to 0.5*(0.302+2*0.162)=0.313 (only one processor is "busy" in states s_5, s_6 and s_7, and both processors in states s_i, $i = 1, 2, 3, 4$). The average turnaround time of class–2 jobs can be determined from the utilization of class–2 terminals which is equal to 0.568 ($x(s_7) + x(s_8) + x(s_9)$), and since the average terminal time is 1 time unit, the throughput

rate is 0.568/1=0.568 job pairs per time unit, and then the average turnaround time is simply 1/0.568=1.761 time units per pair of jobs.

A similar approach is used for models in which the number of states is infinite (state space of open queueing network models is usually infinite), but the "structure" of the state space exhibits a regularity that can be used for "folding" this infinite space into a finite reduced representation. For many unbounded timed nets such a regularity (and, in fact, a condition for timed unboundedness of a net) can be detected during the generation of the state space (actually this is performed in the procedure "search").

The unbounded M–timed net $\mathbf{T}_3$ shown in Fig.5 is a simple open network model in which a "source" with exponentially distributed interarrival times is represented by p_1 and t_1 (the arrival rate $f(t_1)$ is equal to 1 arrival per time unit), and the remaining part of the net models a (single channel) server composed of two consecutive stages. The first stage (t_2 and t_3) provides service with a hyperexponential distribution; the service rate is equal to 2 with probability 0.25 (t_2) and 5 with probability 0.75 ($_3$); the place p_2 is a free-choice place, and the subset t_2, t_3 is a free–choice equivalence class. Service times of the second stage (t_4) are exponentially distributed with the rate equal to 4. The total service time is thus hypoexponentially distributed with corresponding parameters.

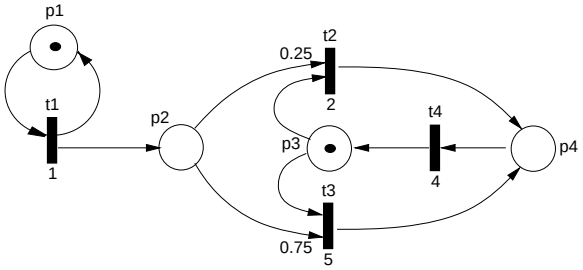

Fig.5. Unbounded M–timed Petri net $\mathbf{T}_3$.

The derivation of the (reduced) state space for $\mathbf{T}_3$ is shown in Tab.3, in which the second "layer" of the folded graph is indicated by star symbols (the states "*7(4)", "9(6)" and "10(8)") with the corresponding "basic" states of the graph given in parentheses; similarly, the states "*11", "*12" and "*13" form the third "layer", etc. The regular structure of this infinite graph is sketched in Fig.6 which clearly shows consecutive "layers" of folding.

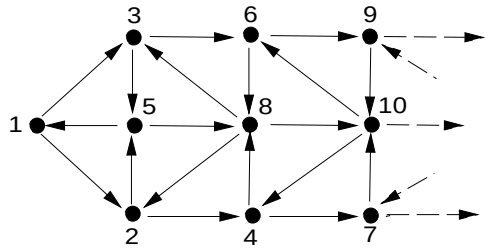

Fig.6. State graph for $\mathbf{T}_3$.

The infinite set of states $S(\mathbf{T}_3)$ can be subdivided into four disjoint classes of states, S_0 that contains all "irregular" states of $\mathbf{T}_3$ (i.e., s_1, s_2, s_3 and s_5), S_1 that represents the "bottom" layer of folding (s_4, s_6 and s_8), S_2 that represents the "second" layer of folding (s_7, s_9 and s_{10}), and all remaining states. Since the stationary probabilities of

Tab.3. The set of reachable states for $\mathbf{T}_3$.

s_i	m_i 1 2 3 4	n_i 1 2 3 4	$h(s_i)$	t_k	s_j	$q(s_i,s_j)$
1	0 0 1 0	1 0 0 0	1.000	1	2	0.750
					3	0.250
2	0 0 0 0	1 0 1 0	0.167	1	4	0.167
				3	5	0.833
3	0 0 0 0	1 1 0 0	0.333	1	6	0.333
				2	5	0.667
4	0 1 0 0	1 0 1 0	0.167	1	*7	0.167
				3	8	0.833
5	0 0 0 0	1 0 0 1	0.200	1	8	0.200
				4	1	0.800
6	0 1 0 0	1 1 0 0	0.333	1	*9	0.333
				2	8	0.667
8	0 1 0 0	1 0 0 1	0.200	1	*10	0.200
				4	2	0.600
					3	0.200
*7(4)	0 2 0 0	1 0 1 0	0.167	1	*11	0.167
				3	*10	0.833
*9(6)	0 2 0 0	1 1 0 0	0.333	1	*12	0.333
				2	*10	0.667
*10(8)	0 2 0 0	1 0 0 1	0.200	1	*13	0.200
				4	4	0.600
					6	0.200

folded states in consecutive layers (i.e., $x(s_4)$, $x(s_7)$, $x(s_{11})$, ..., $x(s_6)$, $x(s_9)$, $x(s_{12})$, ..., and $x(s_8)$, $x(s_{10})$, $x(s_{13})$, ...) are geometrically distributed, the infinite sum of probabilities can be replaced be the sums of corresponding geometrical series with the quotient of these series (denoted by ρ) as a new unknown. This leads to a system of simultaneous nonlinear (and for single unbounded place nets, quadratic) equilibrium equations:

$$\begin{cases} \displaystyle\sum_{s_j\in S_0\cup S_1} h(s_j)*q(s_j,s_i)*x(s_j) = h(s_i)*x(s_i);\ s_i\in S_0 \\ \displaystyle\sum_{s_j\in S_0\cup S_1} h(s_j)*q(s_j,s_i)*x(s_j) + \\ \rho\displaystyle\sum_{s_j\in S_2} h(s_j)*q(s_j,s_i)*x(s_jk) = h(s_i)*x(s_i);\ s_i\in S_1 \\ (1-\rho)\displaystyle\sum_{s_i\in S_0} x(s_i) \ +\ \sum_{s_j\in S_1} x(s_j) \ =\ 1-\rho \end{cases}$$

For $\mathbf{T}_3$ this system contains 8 equations (4 for S_0, 3 for S_1 and the 'normalizing' equation), and the solution is $\rho = 0.483$, $x(s_1) = 0.472$, $x(s_2) = 0.093$, $x(s_3) = 0.062$, $x(s_4) = 0.032$, $x(s_5) = 0.118$, $x(s_6) = 0.032$ and $x(s_7) = 0.068$. The remaining probabilities can be obtained from recursive formulas, e.g., $x(s_7) = \rho * x(s_4)$, etc. Since the server is idle only in the state s_1 (see $m_i(p_3) = 1$ in Tab.3), the utilization of the server is equal to $1 - x(s_1) = 0.528$, and so on.

8. CONCLUDING REMARKS

It has been shown that two different classes of timed Petri nets, D–timed nets with deterministic firing times, and M–timed Petri nets with exponentially distributed random firing times, can be described by a uniform formalism and represented in a very similar way. This similarity can

be used for derivation of other classes of timed nets, for example nets with exponentially distributed as well as deterministic firing times. Recently Ajmone Marsan et al. derived some results for stochastic nets in which there is only one firing transition with a deterministic firing time; this strong restriction in not really surprising since the stochastic approach is based on the set reachable markings, and does not provide "memory" which is necessary for nets with non-Markovian firing times. Timed Petri nets should be much more flexible for such generalizations; the difficulty is expected rather in efficient evaluations of transition probabilities and holding times than in new representations of the states.

The stationary probabilities of the states are obtained by solving a system of simultaneous linear equations. Therefore it may seem that the proposed approach is feasible only for analysis of rather small systems. It should be noted, however, that the state graphs generated by timed Petri nets have a regular structure implied by (usually) small number of firing transitions. This regularity can be used for "folding" isomorphic subgraphs into reduced representations that can be solved efficiently. The same regularity can be used for finite representations of infinite state space which correspond to unbounded timed nets.

Timed Petri nets discussed in this paper are restricted in a number of ways (free–choice simple nets), some of these restrictions, however, can be removed easily by appropriate modifications of the formalism. In fact, nets with more general conflicts can be handled in a very similar way if the probabilities of conflicting transitions are known and included in the state description (e.g., state–dependent probabilities, as proposed in [7] or random switches from [2]). Also, nonsimple nets, i.e., nets with several levels of interrupts, can be taken into account by a rather simple modification of state transition definitions. Moreover, some additional flexibility is offered by enhanced nets [21] in which the set of transitions is partitioned into two classes of transitions, timed and immediate transitions, with firing times assigned to timed transitions only (immediate transitions fire instantaneously). This provides not only a possibility to model arbitrarily complex conditions, but also reduces many intermediate states which are insignificant for performance analysis; e.g., all states with zero holding times in Tab.1.

Acknowledgement

The Natural Sciences and Engineering Research Council of Canada partially supported this research through Operating Grant A8222.

References

1. T. Agerwala, "Putting Petri nets to work"; IEEE Computer, vol.12, no.12, pp.85–94, 1979.
2. M. Ajmone Marsan, G. Conte, G. Balbo, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems"; ACM Trans. on Computer Systems, vol.2, no.2, pp.93–122, 1984.
3. W. Brauer, W. Reisig, G. Rozenberg (eds.), "Advances in Petri Nets 1986"; Lecture Notes in Computer Science 254 and 255, Springer Verlag 1987.
4. J.P. Buzen, "Fundamental operational laws of computer system performance"; Acta Informatica, vol.7, no.2, pp.167–182, 1976.
5. J.B. Dugan, K.S. Trivedi, R.M. Geist, V.F. Nicola, "Extended stochastic Petri nets - applications and analysis"; Performance'84, E. Gelenbe (ed.), pp.507–519, Elsevier 1984.
6. J.B. Dugan, A. Bobbio, G. Ciardo, K. Trivedi, "The design of a unified package for the solution of stochastic Petri net models"; Proc. Int. Workshop on Timed Petri Nets, Torino, Italy, pp.6–13, 1985.
7. M.A. Holliday, M.K. Vernon, "A generalized timed Petri net model for performance evaluation"; Proc. Int. Workshop on Timed Petri Nets, Torino, Italy, pp.181-190, 1985.
8. R. Janicki, P.E. Lauer, M. Koutny, R. Devillers, "Concurrent and maximally concurrent evolution of nonsequential systems"; Theoretical Computer Science, vol.43, pp.213–238, 1986.
9. L. Kleinrock, "Queueing systems"; J. Wiley & Sons 1975, 1976.
10. P.M. Merlin, D.J. Farber, "Recoverability of communication protocols - implications of a theoretical study"; IEEE Trans. on Communications, vol.24, no.9, pp.1036–1049, 1976.
11. M.K. Molloy, "Performance analysis using stochastic Petri nets"; IEEE Trans. on Computers, vol.31, no.9, pp.913-917, 1982.
12. M.T. Ozsu, "Modeling and analysis of distributed database concurrency control algorithms using an extended Petri net formalism"; IEEE Trans. Software Engineering, vol.11, no.10, pp.1225–1240, 1985.
13. J.L. Peterson, "Petri net theory and the modeling of systems"; Prentice–Hall 1981.
14. C. Ramchandani, "Analysis of asynchronous concurrent systems by timed Petri nets"; Project MAC Technical Report MAC–TR–120, Massachusetts Institute of Technology, Cambridge MA, 1974.
15. R.R. Razouk, "The derivation of performance expressions for communication protocols from timed Petri nets"; Computer Communication Review, vol.14, no.2, pp.210-217, 1984.
16. R.R. Razouk, D.S. Hirschberg, "Tools for efficient analysis of concurrent software systems"; Proc. SOFTFAIR II - Second Conf. on Software Development Tools, Techniques and Alternatives, San Francisco CA, pp.192–198, 1985.
17. W. Reisig, "Petri nets - an introduction"; Springer Verlag 1985.
18. G. Rozenberg (ed.), "Advances in Petri Nets 1987" (Lecture Notes in Computer Science 266); Springer Verlag 1987.
19. J. Sifakis, "Use of Petri nets for performance evaluation"; in: "Measuring, modeling and evaluating computer systems", pp.75–93, North-Holland 1977.
20. A.A. Torn, "Simulation nets, a simulation, modeling and validation tool"; Simulation Journal, vol.45, no.2, pp.71–75, 1985.
21. W.M. Zuberek, "M–timed Petri nets, priorities, preemptions, and performance evaluation of systems"; in: "Advances in Petri Nets 1985" (Lecture Notes in Computer Science 222), G. Rozenberg (ed.), pp.478–498, Springer Verlag 1986.
22. W.M. Zuberek, "Inhibitor D–timed Petri nets and performance analysis of communication protocols"; INFOR Journal, vol.24, no.3, pp.231–249, 1986.
23. W.M. Zuberek, "TPNEV, an interactive program for evaluation of timed Petri nets"; Department of Computer Science, MUN, St. John's, Canada A1C 5S7.