1987 Int. Conference on Parallel Processing (ICPP'87); St.Charles, IL, 17–21 August 1987, pp.695–698.
Copyright © 1987 IEEE/Penn State University (DOI 10-1007/0-271-10060-0).

TIMED PETRI NETS IN MODELLING AND EVALUATION OF MULTIPROCESSOR SYSTEMS

W.M. Zuberek

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

Abstract

Timed Petri nets discussed in this paper are extended Petri nets with exponentially distributed firing times. Interrupt arcs are introduced as an extension of inhibitor arcs to allow modelling of preemptions. Since the behavior of extended free-choice timed Petri nets can be represented by probabilistic *Qs-tate* graphs, stationary probabilities of states can be obtained by standard techniques used for analysis of continuous-time homogeneous Markov chains. An immediate application of such a model is performance analysis of queueing systems with exponentially distributed service and interarrival times, and in particular distributed computer systems. Places of Petri nets model systems queues, transitions represent servers, inhibitor arcs are used to model priorities of simultaneous events, and interrupt arcs provide preemption of servers. Simple models of multiprocessor systems are used as an illustration of modelling and performance evaluation.

1. INTRODUCTION

There are three basic methods for evaluating the performance of a complex system: (1) measuring the system during its operation, (2) generating the results by a simulation program that *imitates* the behavior of the system, and (3) solving a mathematical model that captures the essential features of the system. The last of these methods is in many ways the most satisfying since it provides many insights into dependencies and relationships between system parameters and performance characteristics. On the other hand, it is often necessary to make simplifying assumptions to ensure numerical or analytical tractability of a mathematical description, and therefore a number of tools and techniques have been developed that simplify and automate mathematical modelling [2,11,12]. Stochastic Petri nets and timed Petri nets are two examples of such techniques. Application of timed Petri nets to the modelling and evaluation of multiprocessor systems is the subject of this paper.

Petri nets [1,4,11] are abstract, formal models of systems with interacting, concurrent or parallel components. Multiprocessor systems, distributed databases, communication and computer networks are just a few examples of such systems. Petri nets, however, do not take into account the duration of modelled activities and therefore they are not complete enough for the study and evaluation of systems performance. Several different concepts of *timed* Petri nets [7,12,13,14] and stochastic Petri nets [2,3,10] have been proposed by assigning (deterministic or stochastic) firing or enabling times to the transitions or places of Petri nets.

The approach used in this paper is a continuation of the approach originated by Ramchandani [12], however, the firing times assigned to transitions of a net are independent, exponentially distributed random variables, similarly as in stochastic nets [2,10]. The description of firing transitions is, however, quite different than in stochastic nets, and this results in a significantly different class of models. It appears, that in many cases timed Petri models are much simpler than stochastic models of the same system [16].

2. EXTENDED M-TIMED PETRI NETS

A marked (extended) Petri net is a 6-tuple, $\mathbf{M} = (P, T, A, B, C, m_0)$ where P is a finite, nonempty set of places, T is a finite, nonempty set of transitions, A is a nonempty set of directed arcs which connect places with transitions and transitions with places, $A \subseteq P \times T \cup T \times P$, such that for each transition t there exists at least one input and one output place of t, B is a (possibly empty) set of inhibitor arcs which connect places with transitions, $B \subset P \times T$, and A and B are disjoint sets, C is a (possibly empty) set of interrupt arcs, $C \subseteq B$, and m_0 is an initial marking function which assigns a nonnegative number of so called tokens to each place of the net, $m_0 : P \to \{0, 1, ...\}$.

A place p of a net is shared if it is an input place for more than one transition. A (marked) Petri net is free-choice (or extended free-choice [4]) iff the sets of input and inhibitor places of transitions sharing p are identical (which means that all transitions sharing p are either simultaneously enabled or disabled. A shared place p is guarded iff for each two transitions sharing p there exists another place which is an input place for one of these transitions, and an inhibitor place for the other one (which means that no two transitions sharing p can be enabled simultaneously. A net is free-choice iff all its shared places are either guarded or free-choice. In a free-choice net, the relation of sharing a free-choice place is an equivalence relation, hence it determines a partition of the set of transitions into equivalence classes denoted by $Free(T) = T_1, T_2, ..., T_k$.

In timed Petri nets [7,12,13,14,15] each transition t takes a *real* 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. If a transition is enabled while it fires, a new, independent firing can be initiated. If a net contains conflicts, and there are several different possibilities of firings for the same marking, the selection of firing transitions is assumed to be a random process.

In extended timed Petri nets, firing transitions may be *interrupted* by a special type of inhibitor arcs which are called interrupt arcs. If, during a firing period of a transition t, any one of places connected with t by interrupt arcs becomes nonempty (i.e., it receives at least one token), 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. It should be noticed that a firing of transition t can be interrupted only as a result of the termination of another firing; since interrupt arcs are inhibitor arcs, all places connected with t by interrupt arcs must be empty to enable t and to initiate its firing.

An (extended) M-timed Petri net is a triple, $\mathbf{T} = (\mathbf{M}, c, r)$, where $\mathbf{M}$ is a marked (extended) Petri net, $\mathbf{M} = (P, T, A, B, C, m_0)$, c is a choice function which assigns a *free-choice* probability to each transition of a net in such a way that

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

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

$$Prob(v(t)) = e^{x*r(t)}, x > 0.$$

In timed Petri nets tokens are distributed in places as well as in (firing) transitions. The description of *states* or *configurations* of such nets is thus composed of two functions, a marking and a firing function. Firing functions indicate all *active* firings of transitions.

A state s of an M-timed Petri net $\mathbf{T}$ is a pair $s = (m, f)$ where m is a marking function, $m : P \to \{0, 1, ...\}$, and f is a firing 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, $f : T \to \{0, 1, ...\}$.

For extended free-choice M-timed nets, a discrete-state continuous-time description has been introduced in [14,15] which represents the behavior of nets by equivalent continuous-time homogeneous Markov chains that can be generated directly from net specifications. Stationary probabilities of system states can thus be obtained by standard techniques and this provides many performance measures such as utilization of systems components, average queue lengths, average waiting times, etc. Consequently, the modelling can be performed at the Petri net level rather than at the *state* level of the system, and the net models are usually much simpler than the corresponding state spaces. Moreover, Petri net models are directly related to the modelled system while the state space is an *indirect* representation since it models the behavior of the system.

3. MODELLING AND EVALUATION

The widespread acceptance that Petri nets have gained in modelling of systems is due to very simple representation of concurrency and synchronization of systems activities. In the case of queueing system models, places represent systems queues, transitions servers, directed arcs model flow of activities, and inhibitor arcs are used to indicate priorities of simultaneous events. In extended Petri nets, servers may be preempted by interrupt arcs.

A very simple central server model of an interactive multiprocessor system is shown in Fig.1 (as usual, places are denoted by circles, transitions by bars, and the firing rates and free-choice probabilities are given as additional descriptions of transitions). The central server is modelled by the transition t_1 with p_3 representing a pool of available processors (or server's channels). The terminals are modelled by t_2, while p_1 represents the queue of jobs submitted for execution. The initial number of tokens in p_3 determines the number of processors (in this case 2), and the total initial number of tokens in p_1 and p_2 is equal to the number of (active) terminals (in this case 5). Suppose that all jobs submitted for execution at the central server are of the same type, i.e., they are statistically identical, and that their execution times are exponentially distributed with the average equal to 0.5 time units (i.e., the corresponding firing rate is equal to 2.0, $r(t_1) = 2.0$). Moreover, suppose that the terminal (or *thinking*) times are also exponentially distributed, but with the rate equal to 1.0 (i.e., $r(t_2) = 1.0$). The system is then a Markovian queueing system M/M/n//k (in Kendall classification) with $n = 2$ and $k = 5$.

There are six states of this system determined by the number of jobs in the service station (i.e., in the central server and its queue of waiting jobs) [6,8], and the derivation of these states is shown in Tab.1 which also contains stationary probabilities of the states. It can be observed that the central server is idle in the state s_6 only ($f_6(t_1) = 0$). The stationary

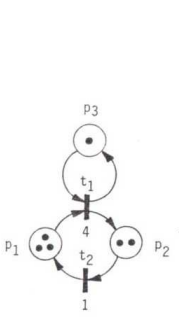

Fig.1. M-timed Petri net $\mathbf{T}_1$.

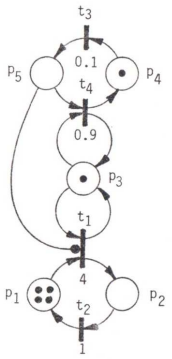

Fig.2. M-timed Petri net $\mathbf{T}_2$.

probability of this state, $x(s_6)$, is equal to 0.111 or 11.1server are busy in the states s_1, s_2, s_3 and s_5 (since $f_i(t_1) = 2$ for $i = 1, 2, 3, 5$), so the utilization of dual processor capabilities of this system is equal to 0.613 or 61.3The throughput of the system is equal to the total utilization of processors divided by the average service time [5,6] or (0.276+2*0.613)/0.5=3.004 jobs per time unit, and then the average turnaround time of a single job is equal to 5/3.004=1.664 time units.

It is known [9] that for such systems the performance deteriorates when the number of processors increases in such a way that the *processing capacity* of the system (or the total service rate, i.e., the sum of service rates of individual processors) is preserved. In other words, for a given processing capacity, the best solution is to use one, fast processor. In terms of Petri net models, since the processing capacity of $\mathbf{T}_1$ from Fig.1 is equal to 4.0, the 'em best model will contain one processor ($m_0(p_3) = 1$), with the service rate equal to 4.0 ($r(t_1) = 4.0$). It can be shown [15] that the corresponding throughput is equal to 0.801/0.25=3.204 jobs per time unit, and the average turnaround time is equal to 5/3.204=1.561 time units. Similarly, for the central server with four processors and service rates equal to 1.0, the throughput decreases to 2.481 jobs per time unit, and the average turnaround time increases to 2.015 time units.

		m_i	f_i		
s_i	$x(s_i)$	1 2 3	1 2	s_j	$u(s_i, s_j)$
1	0.207	1 0 0	2 2	2	4.00
				3	2.00
2	0.276	0 0 0	2 3	4	4.00
				1	3.00
3	0.104	2 0 0	2 1	1	4.00
				5	1.00
4	0.276	0 0 1	1 4	6	2.00
				2	4.00
5	0.026	3 0 0	2 0	3	4.00
6	0.111	0 0 2	0 5	4	5.00

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

Now consider a more interesting case when the processors are not completely reliable. Each processor goes through alternating periods of being operative and broken down, independently of other processors. If the operative and inoperative periods are distributed exponentially with means a and b, respectively, the average probability that a processor is operative is equal to $a/(a + b)$. This means that for the total number of processors equal to N, the effective number of available processors is reduced to $N*a/(a+b)$. It turns out that, in presence of this type of unreliability, the optimal number of processors is no longer 1; in general it is greater than 1, and the closer is the

system to the saturation, the greater is the optimal number of processors [9].

A Petri net model of a central server system with unreliable processors is shown in Fig.2 (in fact, $\mathbf{T}_2$ models central server with one processor, $m_0(p_3) = m_0(p_4) = 1$). The circuit p_4, t_3, p_5, t_4 models exponentially distributed operative and inoperative periods of time with rates $r(t_3) = 0.1$ and $r(t_4) = 0.9$, respectively. During inoperative periods, the tokens representing processors are removed from the central server (i.e., from p_3). If a processor becomes inoperative while it is processing one of jobs, the arc (p_5, t_1) interrupts the firing transition t_1 (and returns the tokens to p_1 - the *job* token, and p_3 - the *processor* token), after which t_4 initiates its firing removing the token from p_3 for an inoperative period.

The derivation of the state space $S(\mathbf{T}_2)$ is shown in Tab.2. The throughput of this system is equal to 2.54 jobs per time unit, and the average turnaround time is equal to 1.575 time units. For two processors ($m_0(p_3) = m_0(p_4) = 2$) and for the same processing capacity as before (i.e., $r(t_1) = 2.0$) the throughput increases to 2.798 jobs per time unit, and the average turnaround time decreases to 1.430 time units. For three processors, the throughput decreases to 2.191 and the average turnaround time increases to 1.825 time units.

		m_i	f_i		
s_i	$x(s_i)$	1 2 3 4 5	1 2 3 4	s_j	$u(s_i, s_j)$
1	0.036	3 0 0 0 0	1 0 1 0	2	4.00
				3	0.10
2	0.116	2 0 0 0 0	1 1 1 0	4	4.00
				1	1.00
				5	0.10
3	0.034	4 0 0 0 0	0 0 0 1	1	0.90
4	0.212	1 0 0 0 0	1 2 1 0	6	4.00
				2	2.00
				7	0.10
5	0.027	3 0 0 0 0	0 1 0 1	3	1.00
				2	0.90
6	0.271	0 0 0 0 0	1 3 1 0	8	4.00
				4	3.00
				9	0.10
7	0.020	2 0 0 0 0	0 2 0 1	5	2.00
				4	0.90
8	0.265	0 0 1 0 0	0 4 1 0	6	4.00
				10	0.10
9	0.012	1 0 0 0 0	0 3 0 1	7	3.00
				6	0.90
10	0.005	0 0 0 0 0	0 4 0 1	9	4.00
				8	0.90

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

Quite often jobs arriving to a multiprocessor system belong to several different types (or classes) with (perhaps) different arrival rates and average execution times. Moreover, some of the jobs may require better service than the other. A popular solution is to assign priorities to different classes of jobs and to select jobs (from the queue of waiting jobs) in order of their priorities. A very simple Petri net model of a system with two classes of jobs and with preemptive priority scheduling is shown in Fig.3 (it should be observed that the net from Fig.3 has the same *structure* as the net from Fig.2; the two nets differ in the initial marking functions and the firing rate functions).

The central server is represented by p_1, t_2 and t_3. The first class of jobs (or users) is modelled by the subnet t_1, p_2, t_2 and p_3, and the second class by the subnet p_4, t_3, p_5 and t_4. The interrupt arc (p_2, t_3) performs preemption of processors executing class-2 jobs when there is any waiting class-1 job. For $n_1 = 1$ user in class-1 and $n_2 = 3$ users in class-3, the derivation of the state space is shown in Tab.3.

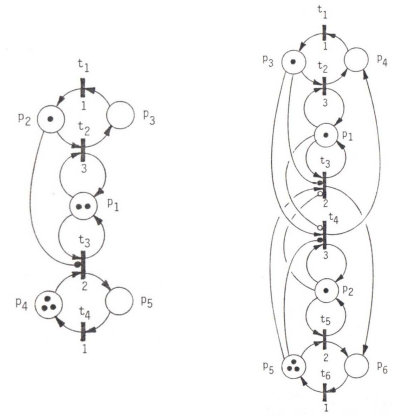

Fig.3. M-timed Petri net $\mathbf{T}_3$.

Fig.4. M-timed Petri net $\mathbf{T}_4$.

		m_i	f_i		
s_i	$x(s_i)$	1 2 3 4 5	1 2 3 4	s_j	$u(s_i, s_j)$
1	0.023	0 0 0 2 0	0 1 1 0	2	3.00
				3	2.00
2	0.048	0 0 0 1 0	1 0 2 0	1	1.00
				4	4.00
3	0.068	0 0 0 1 0	0 1 1 1	4	3.00
				5	2.00
				1	1.00
4	0.173	0 0 0 0 0	1 0 2 1	3	1.00
				6	4.00
				2	1.00
5	0.093	0 0 0 0 0	0 1 1 2	6	3.00
				7	2.00
				3	2.00
6	0.320	1 0 0 0 0	1 0 1 2	5	1.00
				8	2.00
				4	2.00
7	0.066	1 0 0 0 0	0 1 0 3	8	3.00
				5	3.00
8	0.209	2 0 0 0 0	1 0 0 3	7	1.00
				6	3.00

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

Using the same relationships as before, the performance is evaluated independently for each class of jobs. Since the class-1 jobs are executed in the states s_1, s_3, s_5 and s_7, the utilization of processors for class-1 jobs is equal to the sum of $x(s_i)$, $i = 1, 3, 5, 7$. The utilization of processors for class-2 must take into account the fact that in the states s_2 and s_4 both processors execute class-2 jobs. It should be noticed that due to preemption of class-2 jobs, the increasing number of class-1 jobs will effectively *block* class-2 jobs from execution, i.e., the limiting values of throughputs (when n_1 increases to infinity) are 6.0 and 0.0 for class-1 and class-2, respectively. Several performance measures for different values of n_1 and for $n_2 = 3$ are as follows:

	$n_1 = 1$	$n_1 = 3$	$n_1 = 5$
class-1 utilization of processors	0.250	0.744	1.194
average class-1 throughput rate	0.750	2.232	3.582
average class-1 turnaround time	1.333	1.344	1.395
average class-1 waiting time	0.000	0.011	0.062
class-2 utilization of processors	0.946	0.808	0.602
average class-2 throughput rate	1.892	1.616	1.204
average class-2 turnaround time	1.586	1.856	2.492
average class-2 waiting time	0.086	0.356	0.992

In order to avoid *blocking* of jobs and to *guarantee* certain levels of processing in all job classes, another strategy is used in which the set of N processors is subdivided into groups of K_i processors designated to class-i jobs, $N = K_1 + ... + K_m$, where m is the number of job classes. Class-i jobs are then given preemptive priority on designated K_i processors. The processing capacity for class-i jobs is K_i if there are at least K_j class-j jobs in the server, $j \neq i$, and it rises up to N when the number of other jobs decreases. Fig.4 shows a Petri net modelling such a system with 2 classes of jobs, and with one processor designated to each class, $K_1 = K_2 = 1$. Performance results for several values of n_1 and for $n_2 = 3$ are as follows:

	$n_1 = 1$	$n_1 = 3$	$n_1 = 5$
class-1 utilization of processors	0.250	0.681	0.978
average class-1 throughput rate	0.750	2.043	2.934
average class-1 turnaround time	1.333	1.468	1.704
average class-1 waiting time	0.000	0.135	0.361
class-2 utilization of processors	0.946	0.875	0.822
average class-2 throughput rate	1.892	1.750	1.644
average class-2 turnaround time	1.586	1.714	1.825
average class-2 waiting time	0.086	0.214	0.325

In this case, the limiting values of throughputs depend upon n_1 and n_2. When both, n_1 and n_2, tend to infinity, the throughputs approach 3.0 and 2.0 for class-1 and class-2, respectively. For a fixed value of n_2, the limiting value of the class-1 throughput is between 3.0 and 6.0 (it is equal to 6.0 for $n_2 = 0$). For $n_2 = 3$, the limiting value of the class-2 throughput is equal to the throughput of a single processor system serving only class-2 jobs, i.e., 1.579 jobs per time unit. Then the limiting class-1 throughput is equal to 3.632. It can be observed that the results obtained for $n_1 = 10$ are quite close to the limiting values. ¡p¿ There are many other results that can be derived in a similar way.

4. CONCLUDING REMARKS

It can be observed that extended M-timed Petri nets provide a uniform and rather straightforward method for typical evaluation studies of queueing systems, and in particular for performance analysis of multiprocessor systems. Petri net models are usually rather simple, and their parameters directly correspond to components or activities of the modelled systems (e.g., the numbers of processors or jobs, different classes of jobs, service rates, etc.). The state space can be automatically derived from model specifications, and many performance measures can be obtained from the state equilibrium probabilities using rules derived for operational analysis.

All models presented in this paper belong to the class of closed-network systems, however, timed Petri nets can also be applied to modelling and analysis of open-network systems [15]. Analysis of open-network models is much more difficult because the corresponding state space is infinite (the modelling nets are unbounded), and the states must be *reduced* or *folded* in order to perform effective evaluations.

The class of extended timed Petri nets discussed in this paper is restricted in several ways (simple free-choice nets), some of the restrictions, however, can be removed easily by appropriate modifications of the formalism. In fact, nets with more general conflicts (or *random switches* [2]) can be handled in a very similar way provided the probabilities of conflicting firings are known and included in the state description. Also, non-simple nets, i.e., net with several levels of interrupts, can be covered by a rather straightforward generalizations. Moreover, some further *flexibility* is offered by enhanced Petri nets [14] in which the set of transitions is subdivided into two classes, timed and immediate transitions, and the firing times are associated with timed transitions only (immediate transitions fire *instantaneously*). This allows not only modelling of arbitrarily complex selection and scheduling strategies, but it also reduces many *intermediate* states which are insignificant for performance analysis.

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 Magazine, 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] G. Balbo, S.C. Bruell, S. Ghanta, Modeling priority schemes; Performance Evaluation Review (Proc. of the 1985 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems), vol.13, no.2, pp.15-26, 1985.

[4] W. Brauer (ed.), **Net theory and applications**; Proc. of the Advanced Course on General Net Theory of Processes and Systems, Hamburg 1979 (Lecture Notes in Computer Science 84), Springer Verlag 1980.

[5] J.P. Buzen, Fundamental operational laws of computer system performance; Acta Informatica, vol.7, no.2, pp.167-182, 1976.

[6] D. Ferrari, **Computer systems performance evaluation**; Prentice-Hall 1978.

[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] L. Kleinrock, **Queueing systems**, vol.1: Theory, vol.2: Computer applications; J. Wiley & Sons 1975, 1976.

[9] I. Mitrani, Probabilistic modelling of distributed computing systems; in: **Distributed Computing Systems Programme**, D.A. Duce (ed.), pp.139-153, Peregrinus Ltd., London 1984.

[10] M.K. Molloy, Performance analysis using stochastic Petri nets; IEEE Trans. on Computers, vol.31, no.9, pp.913-917, 1982.

[11] J.L. Peterson, **Petri net theory and the modeling of systems**, Prentice-Hall 1981.

[12] 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.

[13] J. Sifakis, Use of Petri nets for performance evaluation; in: **Measuring, modelling and evaluating computer systems**, pp.75-93, North-Holland 1977.

[14] 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.

[15] W.M. Zuberek, On modelling and evaluation of multiprocessor systems using extended M-timed Petri nets; Technical Report #8605, Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada A1C 5S7, 1986.

[16] W.M. Zuberek, On modelling and performance evaluation using stochastic and M-timed Petri nets; Technical Report #8601, Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada A1C 5S7, 1986.