IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC 1998); San Diego, California, 11-14 October 1998, pp-278-283.
Copyright © 1998 IEEE (DOI 10.1109/ICSMC.1998.725422).

Timed Petri Nets and Performance Evaluation of Systems

W.M. Zuberek
Department of Computer Science
Memorial University of Newfoundland
St. John's, Canada A1B 3X5

Abstract

Several simple applications of timed Petri nets to modeling and performance analysis of concurrent systems are presented as an illustration of a uniform approach to analysis of a wide class of discrete–event systems. Such a unified approach is used in a graduate course on performance evaluation of systems at Memorial University.

1. INTRODUCTION

A graduate course on modeling and performance evaluation of systems, offered by the Department of Computer Science at Memorial University (CS–6726), covers a number of "standard" topics, including a brief introduction to discrete–event simulation and basics of queueing theory. It also contains a module (approximately two weeks) on modeling and evaluation of systems using timed Petri nets. Since this module is usually at the end of the course, it relies quite heavily of many concepts introduced earlier, like Markov and semi–Markov chains, stationary probabilities of states, solutions techniques, etc. The focus of this module is on modeling with Petri nets and then analysis and evaluation of the models.

Elemets of Petri net theory which are usually covered in this module include:

- basic concepts of ordinary Petri nets [1, 10, 12], bounded and unbounded nets, liveness and deadlocks, reachability graph;
- place and transition invariants [12], properties of nets based on invariants;
- inhibitor arcs and priorities;
- introduction to colored nets [9], token attributes and transition occurrences, unfolding into place/transition nets;
- timed [14, 8] and stochastic [3] nets, states and state transitions, deterministic and stochastic firing times; performance analysis.

Examples of timed Petri net models used for illustrations cover a wide range of systems:

- simple models of interactive systems; exhaustive reachability analysis, repeated for different values of model parameters, produces well–known characteristics of response time or throughput as a function of the initial marking;
- models of simple communication protocols with failures and timeouts used for recovery; the performance characteristics are obtained by reachability analysis and also by net reductions;
- models of multiprocessor systems; model symmetries are used for a straightforward conversion to colored nets; because of the size of the model, results are obtained by discrete–event simulation of the net model.

Evaluation of net models can be quite involved as it requires maintaining and updating detailed state descriptions of the model. It is, therefore, not surprising that many different tools have been developed for analysis of a variety of net types [7]. A collection of software tools developed for analysis of timed Petri net models, TPN–tools [15, 16], has a modular structure with common internal representation of models and a common 'language' for the specification of modeling nets. TPN–tools support reachability analysis (with a simple interpreted language for extracting performance information from the generated state space), analysis of place and transition invariants, and also net simulation (with another simple interpreted language for post–processing of simulation results). TPN–tools accept a broad class of timed place/transition nets with arc weights, and also with inhibitor and interrupt arcs, as well as timed colored nets [15].

The remaining part of this paper discusses in greater detail several typical examples of timed net models. Section 2 provides a very brief overview of basic concepts of Petri nets and timed nets. Section 3 discusses a simple model of an interactive system; it also identifies some of many variations of this model including job classes and scheduling with and without preemption, unreliable and multiple processors. A model of a simple communication protocol is presented in Section 4, while Section 5 discusses a model of a more complex, distributed memory multithreaded architecture.

2. BASIC CONCEPTS

Petri nets have been proposed as a simple and convenient formalism for modeling systems that exhibit parallel and concurrent activities [1, 11, 12, 10]. In Petri nets, these activities are represented by the so called tokens which can move within a (static) graph–like structure of the net. More formally, a marked (place/transition) Petri net $\mathcal{M}$ contains two basic components, $\mathcal{M} = (\mathcal{N}, m_0)$, the structure $\mathcal{N}$ which is a bipartite directed graph, $\mathcal{N} = (P, T, A)$, where P and T are two disjoint classes of nodes, and A is a set of directed arcs, $A \subseteq T \times P \cup P \times T$, and an initial marking function m_0, which assigns nonnegative numbers of tokens to places of the net, $m_0 : P \rightarrow \{0, 1, ...\}$. Place/transition Petri net models are also known as 'condition/event' systems as places usually represent conditions (in the most general sense), while transitions – events. If a nonzero number of tokens is assigned to a place, the place is 'marked', which means that the condition represented by it is satisfied. If all places connected by directed arcs to a transition are marked, the transition is 'enabled' and can fire (i.e., the event represented by this transition can occur). Firing a transition removes (simultaneously) a single token from each of the input places of the transition and adds a single token to each of the transition's output places. This creates a new marking in a net, a new set of enabled transitions, and so on. The set of all possible marking functions which can be derived in such a way is called the set of reachable markings (or the forward marking class) of a net. This set can be finite or infinite.

An important extension of the basic net model is addition of inhibitor arcs [2, 13]. Inhibitor arcs (which connect places with transitions) provide a 'test if zero' condition which is nonexistent in the basic Petri net. Nets with inhibitor arcs are usually called inhibitor nets. In inhibitor nets, a transition is enabled only if all places connected to it by directed arcs are marked and all places connected by inhibitor arcs are empty (i.e., not marked). Formally, the set B of inhibitor arcs is an additional element of the net structure, $\mathcal{N} = (P, T, A, B)$, $B \subset P \times T$, and usually the same place cannot be connected by a directed and inhibitor arc with the same transition, $A \cap B = \emptyset$.

In order to study performance aspects of Petri net models, the duration of activities must also be taken into account and included into model specifications. Several types of Petri nets 'with time' have been proposed by assigning 'firing times' to the transitions or places of a net. In timed nets [14], firing times are associates with transitions, and transition firings are 'real–time' events, i.e., tokens are removed from input places at the beginning of the firing period, and they are deposited to the output places at the end of this period (sometimes this is also called a 'three–phase' firing mechanism as opposed to 'one–phase' instantaneous firings of transitions). All firings of enabled transitions are initiated in the same instants of time in which the transitions become enabled (although some enabled transitions cannot initiate their firings). If, during the firing period of a transition, the transition becomes enabled again, a new, independent firing can be initiated, which will 'overlap' with the other firing(s). There is no limit on the number of simultaneous firings of the same transition (sometimes this is called 'infinite firing semantics'). Similarly, if a transition is enabled 'several times' (i.e., it remains enabled after initiating a firing), it may start several independent firings in the same time instant.

In timed nets, the initiated firings continue until their termination. A special type of inhibitor arcs, called interrupt arcs, provides a convenient extension which can discontinue transition firings. If, during a firing period of a transition, any place connected to this transition by an interrupt arc becomes marked, the firing discontinues, and tokens removed from the transition's input places at the beginning of firing are returned to these places. Interrupt arcs do not increase the "modeling power" of net models, but they provide a very convenient addition.

In timed nets, the firing times of some transitions may be equal to zero, which means that the firings are instantaneous; all such transitions are called immediate (while the other are called timed). Since the immediate transitions have no tangible effects on the (timed) behavior of the model, it is convenient to 'split' the set of transitions into two parts, the set of immediate and the set of timed transitions, and to fire first the (enabled) immediate transitions, and then (still in the same time instant), when no more immediate transitions are enabled, to start the firings of (enabled) timed transitions. It should be noted that such a convention effectively introduces the priority of immediate transitions over the timed ones, so the conflicts of immediate and timed transitions should be avoided.

The firing times of transitions can be either deterministic or stochastic (i.e., described by some probability distribution function); in the first case, the corresponding timed nets are referred to as D–timed nets, in the second, for the (negative) exponential distribution of firing times, the nets are called M–timed nets (Markovian nets). In both cases, the concepts of state and state transitions have been formally defined and used in the derivation of different performance characteristics of the model [14].

3. M–TIMED NET MODELS

A very simple model of an interactive computer system is shown in Fig.1, in which place p_1 represents the (idle) processor, transition t_1 models a processor executing a job, place p_2 is the queue of jobs waiting for execution, transition t_2 represents the 'thinking time' of users, and place p_3 is simply a termination of job execution (which immediately initiates thinking phase because there is no other condition of t_2's firing). The initial marking function indicates one processor (p_1) and three jobs waiting for execution in p_2 (the number of active users in this model is equal to the sum of initial marking of p_2 and p_3).

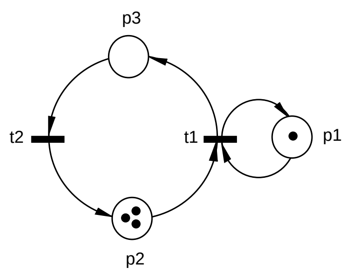

Fig.1. A simple model of an interactive system.

If the queue of waiting jobs (p_2) is nonempty and the processor is available (i.e., p_1 is marked), t_1 is enabled, and a job from p_2 can start its execution by firing t_1 and removing single tokens from p_1 and p_2. The job execution time is an exponentially distributed random variable; its average value is associated with t_1 as the firing time. When the firing of t_1 ends, a token is deposited into p_3 (the completed job) and into p_1 (the processor becomes available for the execution of another job); if, at this instant of time, p_2 contains a token, another firing of t_1 is initiated (again by removing tokens from p_1 and p_2). The token deposited in p_3 initiates firing of t_2; the job just completed enters its 'thinking phase', which is another exponentially distributed random variable with its average value associated with t_2. When the firing of t_2 is finished, a token is deposited into p_2 as a new job waiting for execution.

It should be observed that, in the net shown in Fig.1, t_2 can have any number of simultaneous firings (actually this number is limited by the initial marking of p_2 and p_3), while t_1 can be firing at most once at any instant of time (because of one initial token in p_1).

If the initial marking function assigns more than one token to p_1, the model changes to an interactive system with several parallel processors, in which several jobs can be executed at the same time.

The exponentially distributed firing times of transitions can be combined into hypo- and hyper-exponential distributions (and thus approximate other distributions). Fig.2(a) shows a model of a two–stage hypo–exponential server, and Fig.2(b) a two–stage hyper–exponential server in which the two transitions form a free–choice structure, with 'choice' probabilities describing random selections [14].

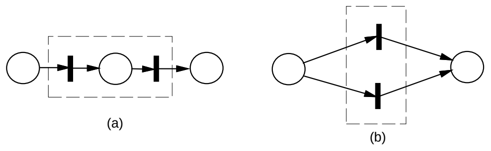

Fig.2. A model of a hypo-exponential (a) and hyper–exponential (b) server.

A modification of the model from Fig.1, in which the processing time is hyper-exponentially distributed, and the thinking time is hypo-exponentially distributed, is shown in Fig.3. Many other variations of this model can easily be derived.

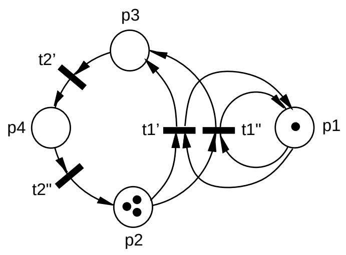

Fig.3. Another model of an interactive system.

A different type of modification of the basic interactive model is shown in Fig.4; in this case, there are two classes of jobs (and users), say A and B; class A is represented by t_1, p_3, t_2 and p_2, class B by t_3, p_5, t_4 and p_4. The processor is shared by both classes; either t_1 can fire or t_3, but not both. Jobs of class A have priority in accessing the processor; the inhibitor arc from p_2 to t_3 disables t_3 if there are any waiting jobs of class A (p_2), so class A jobs are selected for processing (t_1) before class B jobs (non-preemptive priority scheduling).

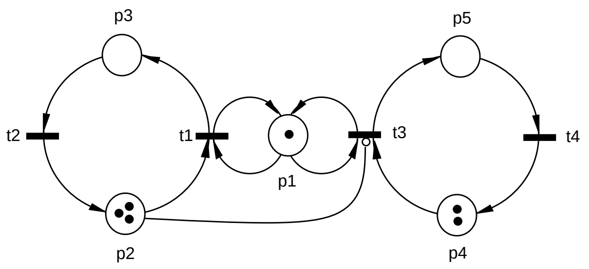

Fig.4. A system with two classes of jobs.

It should be noted that if the inhibitor arc in Fig.4 is replaced by an interrupt arc (interrupt arcs have a black dot instead of the arrowhead), the model will represent preemptive scheduling of class A jobs, in which executing jobs of class B will be interrupted (and preempted of the processor) if a job of class A is ready for execution.

Yet another modification of the basic model of an interactive system is shown in Fig.5; in this case the processor is assumed to be unreliable, so it goes through 'operative' – 'inoperative' cycle, with both 'operative' and 'inoperative' periods of time that are exponentially distributed (but – most likely – with different average values). The 'operative' – 'inoperative' cycle is represented by the subnet t_4, p_4, t_3 and p_5, in which the firing time of t_4 represents the 'operative' periods of time, and the firing time of t_3 – the 'inoperative' periods of time; it should be observed that whenever t_3 fires, the 'processor' token is removed form p_1, so no job can be executed during the firing periods of t_3. The interrupt arc from p_4 to t_1 is used for processor failures during execution of (user) jobs; if a token is deposited into p_4 during t_1's firing, the firing is interrupted by the arc (p_4, t_1), the job token is returned to p_2, the processor token returns to p_1, from where it is removed by firing t_3.

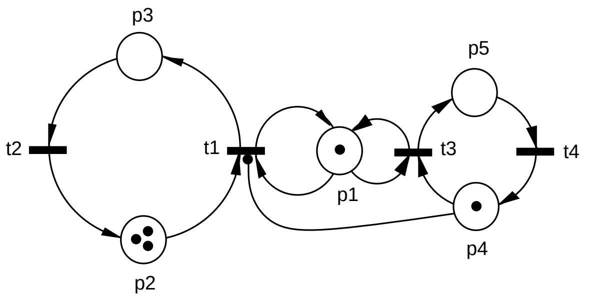

Fig.5. A system with unreliable processor.

It should be observed that the net shown in Fig.5 is structurally similar to the net shown in Fig.4 (with an interrupt arc instead of the inhibitor arc); the model of processor failures is thus similar to a higher priority class of jobs that (conceptually) preempt the processor (for a failure and its repair).

4. D–TIMED NET MODELS

A model of a simple communication protocol with a timeout mechanism, shown in Fig.6, is used as an illustration of timed nets with deterministic firing times [16].

The token in p_1 represents a message which a 'sender' (p_1) sends to a 'receiver' (p_3) and which is confirmed by an acknowledgement sent back to the sender. The message is sent by firing t_1, after which a single token is deposited in p_2 (the message) and in p_5 (the timeout). Firing time of t_2 represents the 'communication delay' of sending a message, and firing time of t_6, the timeout time. When the firing of t_2 is finished, a token is deposited in p_3, the receiver. p_3 is a free-choice place, so t_3 and t_4 are enabled simultaneously, but only one of them can fire; the random choice is characterized by 'choice probabilities' assigned to t_3 and t_4. t_3 represents (in a simplified way) the loss or distortion of the message or its acknowledgement; it t_3 is selected for firing (according to its free–choice probability), the token is removed from p_3 as well as from the model (t_3 is a 'token sink'). In such a case the timeout transition t_6 will finish its firing with no token in p_6; the termination of t_6's firing regenerates the lost token in p_1, so the message will be retransmitted. If the message is received correctly, t_4 is selected for firing rather than t_3, and after another communication delay (modeled by t_5), tokens are deposited in p_6 and p_1 (so another message can be sent to the receiver). The token in p_6 interrupts the firing of t_6, so the 'timeout token' is returned to p_5 and immediately removed by firing t_7.

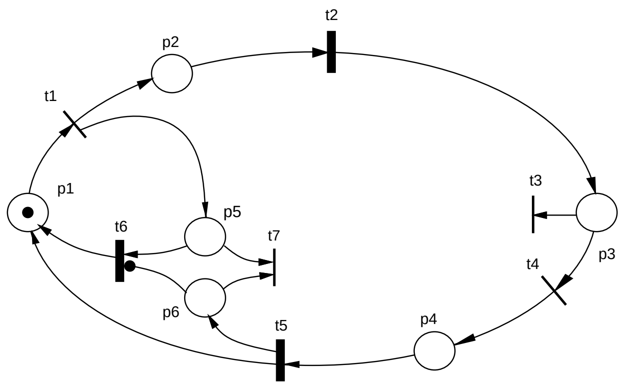

Fig.6. A model of a simple communication protocol.

All immediate transitions (i.e., transitions with zero firing time) are represented by (thin) bars, while timed transitions are represented by (black) rectangles. The firing times of timed transitions must be selected in such a way that the timeout time (t_6) is greater than the sum of the delays of sending a message (t_2) and an acknowledgement (t_5).

The transition t_4 may seem redundant in this model but in fact is required due to the restriction that all free–choice classes of transitions must be uniform, i.e., each free–choice class must contain either immediate or timed transitions but not both [14].

It should be noted that only a small modification of the net in Fig.6 is needed to represent a 'sliding window' protocol, i.e., a protocol with several messages in different stages of transmission/acknowledgement or recovery.

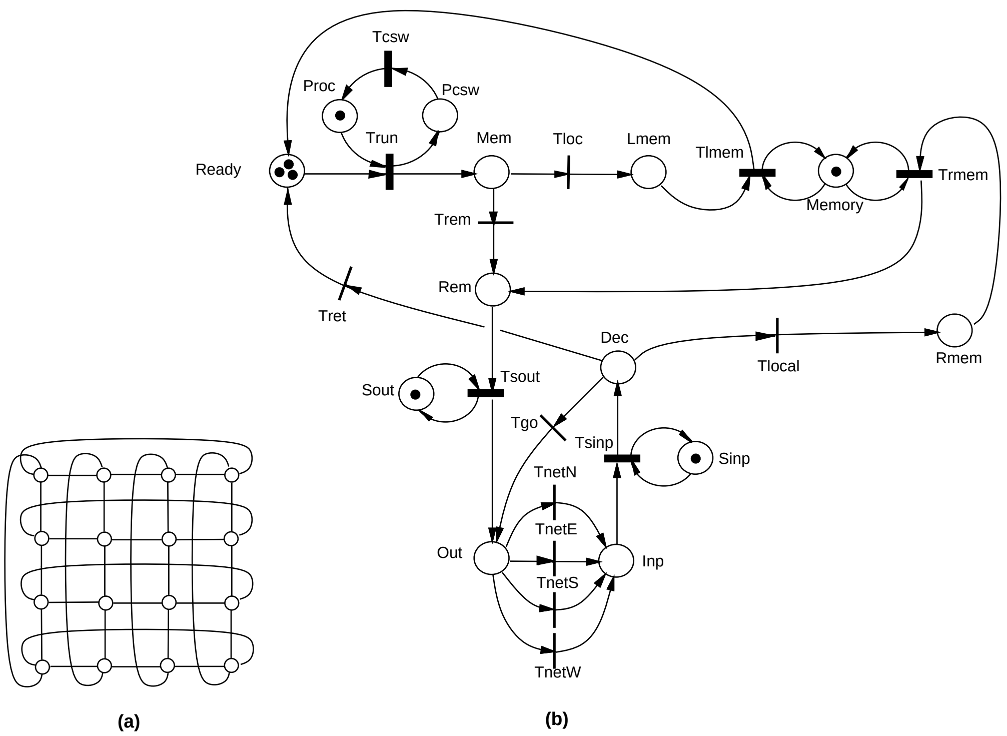

Fig.7. An outline of a 16–processor system (a) and a model of a distributed memory multithreaded system (b).

5. COLORED NET MODELS

In colored Petri nets [9], tokens have attributes called colors. Token colors can be modified by (firing) transitions and also transitions can have several different occurrences (or variants) of firing. The basic idea of colored nets is to 'fold' identical parts of an ordinary Petri net. The original set of places is partitioned into a set of disjoint classes, and each class is replaced by a single place with token colors indicating which of the original places the tokens belong to. Similarly, the original set of transitions is partitioned into a set of disjoint classes, and each class is replaced by a single transition with occurrences indicating which of the original transitions the firing corresponds to.

A timed net model of a distributed memory multithreaded architecture is used as an illustration of colored net models [17]. The distributed memory system is composed of a number processors connected by an interconnection network; Fig.7(a) outlines a 16–processor system with a two–dimensional torus–like interconnection network.

Each node in Fig.7(a) is a multithreaded processor. Fig.7(b) shows a model of a processor as well as its connection with the interconnection network (using two switches, $Tsinp$ and $Tsout$), and the network itself (transitions $TnetN$, $TnetE$, $TnetS$ and $TnetW$). The processor shown in Fig.7(b) performs context switching (i.e., switching from one thread to another) for each long–latency memory access (local or remote).

The execution of threads is modeled by transition $Trun$ with place $Proc$ representing the (available) processor (if marked) and *Ready* – the pool of threads waiting for execution. The initial marking of *Ready* represents the average number of threads. It is assumed that this number does not change in time.

The firing time of *Trun* is exponentially distributed (all other firing times are deterministic) and its average value represents the runlength of threads, i.e., the average number of instructions executed before context switching occurs. Context switching is represented by transition $Tcsw$ (with its firing time).

Mem is a free–choice place, with a random choice of either accessing local memory (*Tloc*) or remote memory ($Trem$); in the first case, the request is directed to *Lmem* where it waits for availability of *Memory*, and after accessing the memory, the thread returns to the pool of waiting threads, *Ready*. *Memory* is a shared place with two conflicting transitions, $Trmem$ (for remote accesses) and $Tlmem$ (for local accesses); the resolution of this conflict (if both accesses are waiting) is based on marking–dependent (relative) frequencies determined by the numbers of tokens in *Lmem* and *Rmem*, respectively.

Requests for remote accesses are directed to *Rem*, and then, after a sequential delay (the switch modeled by *Sout* and *Tsout*), forwarded to *Out*, where a random selection is made of one of the four (for this interconnection network) adjacent nodes (transitions *TnetN*, *TnetE*, etc. represent links NORTH, EAST, SOUTH and WEST in Fig.7(a)). Similarly, the node incoming traffic is collected from all neighboring nodes in *Inp*, and, after a sequential delay (*Sinp* and *Tsinp*), forwarded to *Dec*. *Dec* is a free–choice place with three transitions sharing it: *Tret*, which represents the satisfied requests reaching their 'home' nodes; *Tgo*, which represents requests as well as responses forwarded to another node (another 'hop'); and *Tlocal*, which represents remote requests accessing the memory at the destination node; these remote requests are queued in *Rmem* and served by *Trmem* when the memory module *Memory* becomes available.

Colors are used to 'fold' processors into a single model shown in Fig.7(b). Since transitions *TnetN*, ..., *TnetW* pass messages between processors of the system, they must transform the colors of tokens. A more detailed description of colors and their transformations is given in [17]; some performance results are also shown there.

6. CONCLUDING REMARKS

Timed Petri nets provide a high–level formalism (at the level of abstraction similar to queueing systems) for modeling a wide class of discrete–event systems. The models usually correspond very closely to the real, modeled systems, so the selection of parameters and interpretation of results is quite straightforward.

Since the evaluation of net models can be automated, it is not discussed in this paper. However, it should be indicated that the evaluation of complex models can be quite involved, and therefore systematic methods that can deal with complex models are needed. Hierarchical approach [4, 5, 6] may provide a satisfactory solution once it is better understood.

At Memorial, Petri nets are also used in several other (undergraduate and graduate) courses, for example, operating systems (for modeling systems of interacting processes) and computer architecture (for representing concurrent activities).

References

[1] Agerwala, T., "Putting Petri nets to work"; IEEE Computer Magazine, vol.12, no.12, pp.85-94, 1979.

[2] Agerwala, T., Flynn, M., "Comments on capabilities, limitations and 'correctness' of Petri nets"; Proc. of the First Annual Symp. on Computer Architecture, pp.81-86, 1973.

[3] Ajmone Marsan, M., Balbo, G., Conte, G., "Performance models of multiprocessor systems"; MIT Press 1986.

[4] Buchholz, P., Hierarchical high-level Petri nets for complex system analysis; in: "Application and Theory of Petri Nets 1994"(Lecture Notes in Computer Science 815), pp.119-138, Springer Verlag 1994.

[5] Buchholz, P., Hierarchical structuring of superposed GSPNs; Proc. 7-th Int. Workshop on Petri Net and Performance Models (PNPM'97), pp.81-90, IEEE Press 1997.

[6] Fehling, R., "A concept of hierarchical Petri nets with building blocks"; Proc. Int. Conf. on Applications and Theory of Petri Nets 1991, pp.370–389, 1991.

[7] Feldbrugge, F., "Petri net tool overview 1992"; in: "Advances in Petri Nets 1993" (Lecture Notes in Computer Science 674), Rozenberg, G., (ed.), pp.169-209, Springer Verlag 1993.

[8] Holliday, M.A., Vernon, M.K., "Exact performance estimates for multiprocessor memory and bus interference"; IEEE Trans. on Computers, vol.36, no.1, pp.76-85, 1987.

[9] Jensen, K., "Coloured Petri nets"; in: "Advanced Course on Petri Nets 1986" (Lecture Notes in Computer Science 254), Rozenberg, G. (ed.), pp.248-299, Springer Verlag 1987.

[10] Murata, T., "Petri nets: properties, analysis and applications"; Proceedings of IEEE, vol.77, no.4, pp.541–580, 1989.

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

[12] Reisig, W., "Petri nets - an introduction" (EATCS Monographs on Theoretical Computer Science 4); Springer Verlag 1985.

[13] Valk, R., "Test on zero in Petri nets"; in: "Applications and Theory of Petri Nets"(Informatik–Fachberichte 52), Girault, C., Reisig, W. (eds.), pp.193-197, Springer Verlag 1982.

[14] Zuberek, W.M., "Timed Petri nets – definitions, properties and applications"; Microelectronics and Reliability, vol.31, no.4, pp.627–644, 1991.

[15] Zuberek, W.M., "Modeling using timed Petri nets – model description and epresentation"; Technical Report #9601, Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada A1B 3X5, 1996.

[16] Zuberek, W.M., "Modeling using timed Petri nets – event–driven simulation"; Technical Report #9602, Department of Computer Science, Memorial Univ. of Newfoundland, St. John's, Canada A1B 3X5, 1996.

[17] Zuberek, W.M., Govindarajan, R., "Timed colored Petri net models of distributed memory multithreaded multiprocessors"; Proc. Workshop on Practical use of Colored Petri Nets and Design/CPN, Aarhus, Denmark, 1998.