Technical Report #2000–1

PETRI NETS AND TIMED PETRI NETS BASIC CONCEPTS AND PROPERTIES

Lecture Notes for CS–6726:
Modeling and Analysis of Computer Systems

by

W.M. Zuberek

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

December 2000

Department of Computer Science
Memorial University of Newfoundland
St. John's, Canada A1B 3X5
tel: (709) 737–8627
fax: (709) 737–2009

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

PETRI NETS AND TIMED PETRI NETS BASIC CONCEPTS AND PROPERTIES

A b s t r a c t

Petri nets are formal models of systems which exhibit concurrent activities. Communication networks, multiprocessor systems, manufacturing systems and distributed databases are simple examples of such systems. As formal models, Petri nets are bipartite directed graphs, in which the two types of vertices represent, in a very general sense, conditions and events. An event can occur only when all conditions associated with it (represented by arcs directed to the event) are satisfied. An occurrence of an event usually satisfies some other conditions, indicated by arcs directed from the event. So, an occurrence of one event causes some other event to occur, and so on.

In order to study performance aspects of systems modeled by Petri nets, the durations of modeled activities must also be taken into account. This can be done in different ways, resulting in different types of temporal nets. In timed Petri nets, occurrence times are associated with events, and the events occur in real–time (as opposed to instantaneous occurrences in other models). For timed nets with constant or exponentially distributed occurrence times, the state graph of a net is a Markov chain, in which the stationary probabilities of states can be determined by known methods. These stationary probabilities can be used for the derivation of many performance characteristics of the model.

Analysis of net models based on exhaustive generation of all possible states is called reachability analysis; it provides detailed characterization of model's behavior, but often requires analysis of huge state spaces (the number of states can increase exponentially with some model parameters which is known as "state explosion"). Structural analysis determines the properties of net models on the basis of connections among model elements; structural analysis is usually much simpler than reachability analysis, but can be applied only to models satisfying certain properties. If neither reachability nor structural analysis is feasible, discrete–event simulation of timed nets can be used to study the properties of net models.

This report introduces basic concepts of Petri nets and timed Petri nets, discusses some of their properties, and presents several straightforward applications. It also includes a comprehensive list of references.

A c k n o w l e d g e m e n t

Support of the Natural Sciences and Engineering Research Council of Canada through Research Grant RGPIN–8222 is gratefully acknowledged.

1. INTRODUCTION

Petri nets have been proposed as a simple and convenient formalism for modeling systems that exhibit parallel and concurrent activities [Ag79, Pe81, Re85, Mu89]. The popularity that Petri nets (and their numerous extensions and modifications) have been gaining is due to simple representation of concurrency and synchronization, i.e., those aspect of systems which cannot be expressed easily in traditional formalisms, developed for analysis of systems with sequential behavior.

In order to study performance aspects of Petri net models, the durations of activities must also be taken into account. Several types of Petri nets "with time" have been proposed by assigning "firing times" (or "occurrence times") to the transitions or places of a net. In *timed* nets, 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). In *stochastic* (and *generalized stochastic*) Petri nets [Mo82, AM84, AM00] and their many variants [BP98, CA93, AB89], (exponentially distributed) firing times are associated with transitions, but the tokens remain (for the firing time) in places, and the instantaneous firings occur at the end of firing times (so the "firing times" are actually "enabling times"). In *time* nets [MF76, Aa93] there is an interval associated with a transition, and the (instantaneous) firing must occur within this interval of time.

In timed nets, all firings of enabled transitions are initiated in the same instants of time in which the transitions become enabled. 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").

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–nets, in the second, for the (negative) exponential distribution of firing times, the nets are referred to as M–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 [Zu91].

In (ordinary) nets the tokens are indistinguishable, so their distribution can conveniently be described by a marking function which assigns nonnegative (integer) numbers of tokens to places of the net. In colored Petri nets [Je87], tokens have attributes called colors. Token colors can be quite complex, for example, they can describe the values of (simple or structured) variables or the contents of message packets. Token colors can be modified by (firing) transitions and also a transition can have several different occurrences (or variants) of firing.

The basic idea of colored nets is to "fold" 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.

Any partition of places and transitions will result in a colored net. One of the extreme

partitions will combine all original places into one place, and all original transitions into one transition; this will create a very simple net (one place and one transition only) but with quite complicated rules describing the use of colors. The other extreme partition will create one–element classes of places and transitions, so the colored net will be isomorphic to the original net, with only one color. To be useful in practice, colored nets must constitute a reasonable balance between these two extreme cases.

Analysis of net models can be based on their behavior (i.e., the set of reachable states) or on the structure of the net; the former is called *reachability analysis* while the latter – *structural analysis*. Invariant analysis seems to be the most popular example of the structural approach. Structural methods eliminate the derivation of the state space, so they avoid the "state explosion" problem of reachability analysis, but they cannot provide as much information as the reachability approach does. Quite often, however, all the detailed results of reachability analysis are not really needed, and more synthetic performance measures, that can be obtained by structural methods, are quite satisfactory.

Both reachability and structural analyses are based on quite detailed net characterizations. Consequently, only very simple models can be analyzed manually; for more realistic models, software tools for analysis of net models are needed. It is, therefore, not surprising that many different tools have been developed for analysis of a variety of net types [Fe93]. A collection of software tools developed for analysis of timed Petri net models, TPN–tools, uses the same internal representation of different classes of net models, and a common language for the description of modeling nets [Zu96a].

Timed Petri net models are discrete–event systems as the states change in a discrete manner (by removing or depositing tokens in places). Analysis of timed models by using discrete–event simulation is yet another approach to performance analysis, which imposes very few restrictions on the class of analyzed models [Zu96b].

2. BASIC PETRI NETS

Place/transition Petri nets are based on bipartite directed graphs in which the two types of vertices are called *places* and *transitions*. Place/transition nets are also known as condition/event systems.

A Petri *net structure* $\mathcal{N}$ is a triple $\mathcal{N} = (P, T, A)$ where:

P is a finite set of places (which represent conditions),

T is a finite set of transitions (which represent events), $P \cap T = \emptyset$,

A is a set of directed arcs which connect places with transitions and transitions with places, $A \subseteq P \times T \cup T \times P$, also called the *flow relation* or *causality relation* (and sometimes represented in two parts, a subset of $P \times T$ and a subset of $T \times P$).

For each transition $t \in T$, and each place $p \in P$, the input and output sets are defined as follows:

$$Inp(t) = \{p \in P \mid (p,t) \in A\}, \qquad Inp(p) = \{t \in T \mid (t,p) \in A\},$$
$$Out(t) = \{p \in P \mid (t,p) \in A\}, \qquad Out(p) = \{t \in T \mid (p,t) \in A\}.$$

2.1. Marked nets

The dynamic behavior of nets is represented by markings, which assign nonnegative numbers of tokens to the places of a net. Under certain conditions these tokens can "move" in the net, changing one marking into another.

A *marked Petri net* $\mathcal{M}$ is a pair $\mathcal{M} = (\mathcal{N}, m_0)$, where $\mathcal{N}$ is a net structure, $\mathcal{N} = (P, T, A)$, and m_0 is the initial marking function, $m_0 : P \to \{0, 1, ...\}$ which assigns a nonnegative number of *tokens* to each place of the net. Marked nets are also equivalently defined as $\mathcal{M} = (P, T, A, m_0)$.

Example. Fig.2.1 shows a very simple model of the producer–consumer bounded–buffer system. The cyclic subnet (t_1, p_1, t_2, p_2) represents the producer process which produces an item (t_1) and stores it in the buffer (t_2) provided there is space for it (condition p_5). The cyclic subnet (t_3, p_3, t_4, p_4) represents the consumer process which fetches an item from the buffer (t_3) provided the buffer is nonempty (condition p_6) and consumes it (t_4).

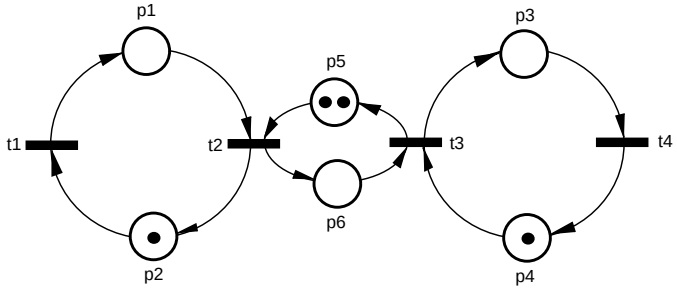

Fig.2.1. Producer–consumer bounded–buffer model.

Let any mapping $m : P \to \{0, 1, ...\}$ be called a marking function in $\mathcal{N} = (P, T, A)$.

In marked nets, a condition represented by a place p is satisfied at a marking m if m assigns a nonzero number of tokens to p, $m(p) > 0$, and then p is said to be marked by m. If all input places of a transition t are marked, t is *enabled*:

$$t \text{ is enabled by } m \;\Leftrightarrow\; \forall\, p \in Inp(t) : m(p) > 0.$$

The set of all transition enabled by a marking m is denoted $En(m)$.

If all (input) conditions of an event are satisfied (i.e., the transition representing this event is enabled), the event can occur (or a transition can occur or fire). An occurrence of an event removes (simultaneously) a single token from all input places of the transition representing this event, and (also simultaneously) adds a single token to all output places of this transition. This creates a new marking function. An occurrence of an event represented by t (i.e., t's firing) is thus a transformation of the (current) marking function m into a new marking function m' which is *directly reachable* from m by firing t, $m \stackrel{t}{\mapsto} m'$:

$$\forall\, p \in P :\; m'(p) \;=\; \begin{cases} m(p) + 1, & \text{if } p \in Out(t) - Inp(t); \\ m(p) - 1, & \text{if } p \in Inp(t) - Out(t); \\ m(p), & \text{otherwise.} \end{cases}$$

Example. In the net shown in Fig.2.1, t_1 is the only transition which is enabled by the initial marking [0,1,0,1,2,0] (in vector notation, so zero tokens are assigned to p_1, one

token to p_2, etc.). An occurrence of t_1 removes one token from p_2 and adds one token to p_1, creating marking [1,0,0,1,2,0]. Since the only transition enabled by this new marking is t_2, its occurrence creates yet another marking [0,1,0,1,1,1]. The set of transitions enabled by this marking includes t_1 and t_3, so any one of these two transitions can occur next.

A marking m_j is *generally reachable* (or just reachable) from a marking m_i in $\mathcal{M}$, $m_i \stackrel{*}{\mapsto} m_j$, if m_j is reachable from m_i by a sequence of directly reachable markings (general reachability relation is the reflexive transitive closure of the direct reachability relation).

The *set of reachable markings*, $\mathbf{M}(\mathcal{M})$, of a marked net $\mathcal{M}$ is the set of all markings which are (generally) reachable from the initial marking m_0:

$$\mathbf{M}(\mathcal{M}) \;=\; \{m \;\mid\; m_0 \stackrel{*}{\mapsto} m\}.$$

A *graph of reachable markings* of $\mathcal{M}$ (not to be confused with a reachability tree) is a directed, arc–labeled graph $\mathcal{R}(\mathcal{M}) = (V, E, \ell)$ in which:

V is a set of vertices which is equal to the set of reachable markings $\mathbf{M}(\mathcal{M})$,

E is a set of directed arcs which represent the direct reachability relation on $\mathbf{M}(\mathcal{M})$, $(m_i, m_j) \in E \Leftrightarrow m_i \mapsto m_j$,

ℓ is a labeling function which assigns subsets of transitions to elements of E, $\ell : E \to 2^T$:

$$\forall\ (m_i, m_j) \in E : \ell(m_i, m_j) = \{t \in T \;\mid\; m_i \stackrel{t}{\mapsto} m_j\}.$$

Example. The graph of reachable marking for the net of Fig.2.1 is shown in Fig.2.2. It can be observed that the graph is finite, strongly–connected (i.e., there is a directed path between any two vertices of the graph), and each cycle contains labels of all transitions from the set T.

The set of reachable markings can be finite or infinite; if it is finite, there is a bound on the number of tokens assigned to any place of the net, and the net $\mathcal{M}$ is *bounded*, otherwise the net is *unbounded*:

$$\mathcal{M} \text{ is bounded} \Leftrightarrow \exists\ k > 0\ \forall\ m \in \mathbf{M}(\mathcal{M})\ \forall\ p \in P : m(p) \leq k.$$

A marked net $\mathcal{M}$ is *safe* if it is bounded and the bound k is equal to 1.

A marking m_j *dominates* marking m_i, $m_j m_i$, iff m_j is componentwise greater than or equal to m_i, and m_j is not equal to m_i (i.e., there exists at least one component of m_j which is greater than the corresponding component of m_i):

$$m_j m_i \;\Leftrightarrow\; m_j \neq m_i \wedge \forall\ p \in P : m_j(p) \geq m_i(p).$$

It can be shown that the set of reachable markings of a marked net $\mathcal{M}$ is infinite iff there exist markings m_i and m_j such that m_i is reachable from m_0, m_j is reachable from m_i, and m_j dominates m_i.

Example. Fig.2.3 presents a simple model of the producer–consumer unbounded–buffer system with p_5 representing the buffer. It should be observed that, in this model,

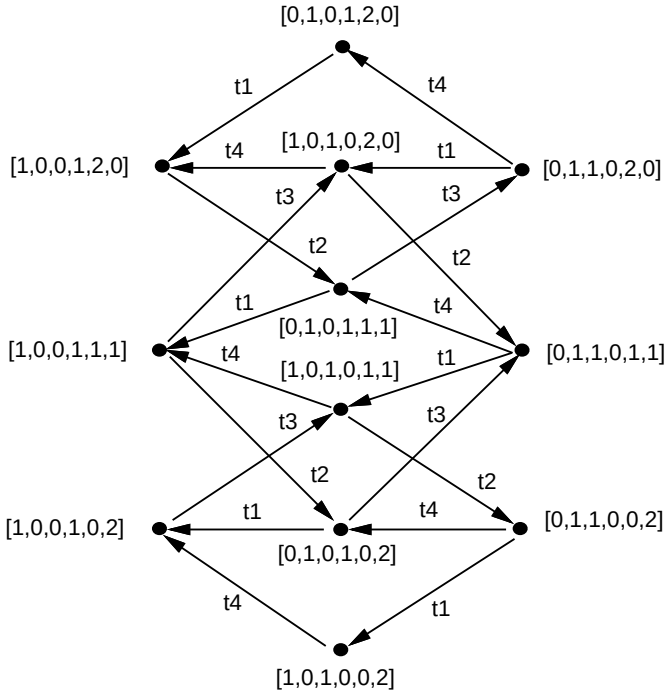

Fig.2.2. Graph of reachable markings for the model of Fig.2.1.

the producer and consumer processes are quite independent. The firing sequence $(t_1 t_2)$ transforms the initial marking [0,1,1,0,0] into marking [0,1,1,0,1] which dominates the initial marking, so the set of reachable markings is infinite; indeed, the firing sequence $(t_1 t_2)$ can be repeated any number of times, systematically increasing the marking of place p_5.

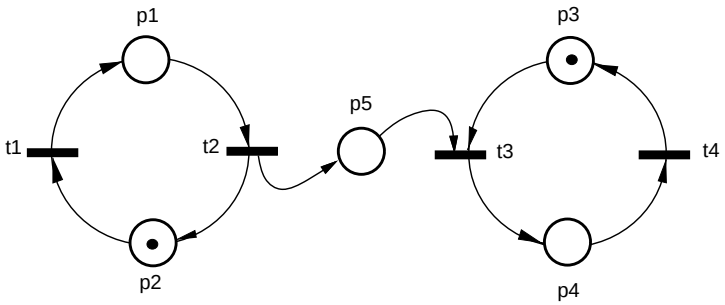

Fig.2.3. Producer–consumer unbounded–buffer model.

For unbounded nets the reachability graphs are infinite, so a different finite representation of net behavior is needed. Such a description is known as *reachability trees*. In reachability trees, the nodes are also markings, but there are no arcs to nodes already existing in the tree (so both dead markings and markings leading to cycles are terminal vertices or "leaves" in reachability trees). Moreover, an additional special symbol, # or ω, is used to represent "any number of tokens". More precisely, if a marking m' is reachable from a marking m and $m'm$, all those elements of m' which are greater than the corresponding elements of m are indicated by #. The special symbol # has the "absorbing" properties,

$\# + k = \#$ and $\# - k = \#$, for any constant k.

Example. Fig.2.4 shows the reachability tree for the producer–consumer unbounded–buffer model shown in Fig.2.3. The firing sequence $(t_1 t_2)$ creates marking $m' = [0, 1, 1, 0, 1]$, which dominates the initial marking $m_0 = [0, 1, 1, 0, 0]$; since $m'(p_5)$ is greater than $m(p_5)$, the element of $m'(p_5)$ is replaced by $\#$ in Fig.2.4 (this also indicates that p_5 is an unbounded place in Fig.2.3).

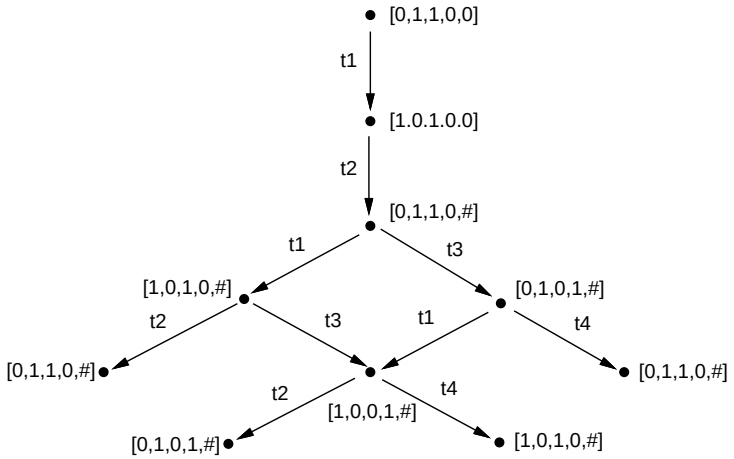

Fig.2.4. Reachability tree for the net in Fig.2.3.

One of the most important properties of many concurrent systems is the absence of *deadlocks*; intuitively, a deadlock is a configuration in which the system cannot continue its operation, it becomes *dead*.

A marking m in net $\mathcal{N}$ is *dead* if no transition is enabled by m, i.e., if $En(m) = \emptyset$. A marked net M contains a *deadlock* if its set of reachable markings contains a dead marking:

$$\mathcal{M} \text{ contains a deadlock } \Leftrightarrow \exists\, m \in \mathbf{M}(\mathcal{M}) : En(m) = \emptyset.$$

Example. Fig.2.5 shows a simple application of Petri nets to modeling resource allocation based on semaphores (with operations $P(s)$ for "dropping" the semaphore s, and $V(s)$ for "rising" the semaphore s). Each resource R_i has a semaphore s_i controlling its allocation; when a process tries to acquire the resource, it performs a $P(s_i)$ operation; after using the resource R_i, the process releases the acquired resource performing operation $V(s_i)$.

Semaphores are modeled by places, which – for single unit resources (e.g., input/output devices) – are initialized (by the initial marking function) to one. Each operation $P(s)$ removes a token from s, so it uses an arc outgoing from s, while each operation $V(s)$ returns a token to s, so it is represented by an arc directed to s.

Fig.2.5 shows two processes sharing two resources, R_1 and R_2, controlled by semaphores s_1 and s_2 (the model can easily be extended to any number of processes and any number of resources). The processes acquire the resources in different order; process–1 first acquires the resource R_1 (performing $P(s_1)$), while process–2 first acquires resource R_2 (performing $P(s_2)$).

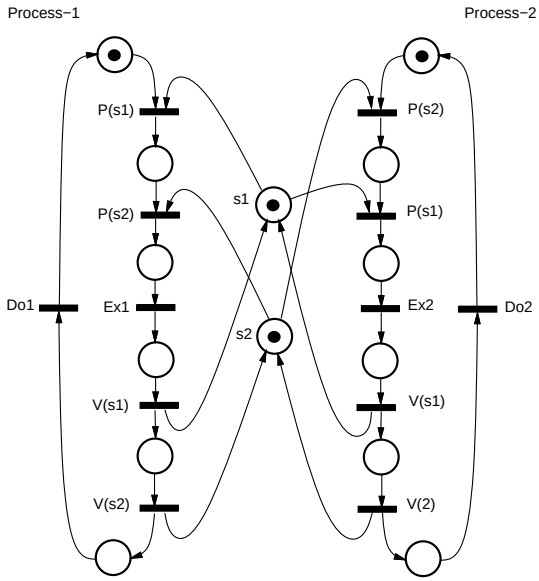

Fig.2.5. Resource allocation model.

A partial graph of reachable markings for this resource allocation model is shown in Fig.2.6. There is a node with no outgoing arcs that represents a deadlock. Indeed, if process–1 acquires R_1, and process–2 acquires R_2, none of the processes can continue without continuation of the other process (and eventual release of the needed resource); such a "cycle" of processes waiting one for another is a characteristic condition of a deadlock. It should be observed that the deadlock occurs only for some sequences of operations, so in a real system the existence of a deadlock may be quite difficult to detect during testing.

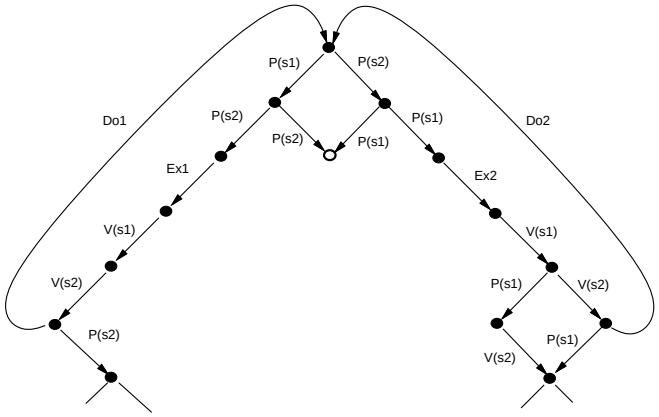

Fig.2.6. Partial graph of reachable markings for net of Fig.2.5.

A marked net $\mathcal{M}$ is *conservative* if the token count for each marking reachable from the initial marking is the same:

$$\mathcal{M} \text{ is conservative} \Leftrightarrow \forall\ m \in \mathbf{M}(\mathcal{M}) : \sum_{p \in P} m(p) = \sum_{p \in P} m_0(p).$$

A marked net $\mathcal{M}$ is *live* iff for any marking m_i reachable from the initial marking m_0

and any transition t, there exists a marking m_j reachable from m_i which enables t (so t can fire):

$$\mathcal{M} \text{ is live } \Leftrightarrow \forall\, m_i \in \mathbf{M}(\mathcal{M})\ \forall\, t \in T\ \exists\, m_j \in \mathbf{M}(\mathcal{M}) : m_i \overset{*}{\mapsto} m_j \wedge t \in En(m_j).$$

Example. The marked net shown in Fig.2.1 is bounded, live and conservative; the net shown in Fig.2.3 is unbounded, live and non–conservative, and the net shown in Fig.2.5 is bounded, non–live (it contains a deadlock) and non–conservative.

2.2. Inhibitor nets

An important extension of the basic net model is addition of *inhibitor arcs* [AF73, Va82]. Inhibitor arcs (which connect places with transitions) provide a "test if zero" condition which is nonexistent in basic Petri nets. 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 unmarked.

An inhibitor (marked) Petri net $\mathcal{M}$ is a pair, $\mathcal{M} = (\mathcal{N}, m_0)$ where $\mathcal{N}$ is a net structure with inhibitor arcs, $\mathcal{N} = (P, T, A, B)$, where B is the set of inhibitor arcs, $B \subseteq P \times T$, $A \cap B = \emptyset$. The set of places connected by inhibitor arcs with transition t is called the inhibitor set of t and is denoted $Inh(t) = \{p \in P \mid (p, t) \in B\}$.

In an inhibitor net $\mathcal{N}$, a transition t is enabled by a marking m iff:

$$t \text{ is enabled by } m \ \Leftrightarrow\ (\forall\, p \in Inp(t) : m(p) > 0) \wedge (\forall\, p \in Inh(t) : m(p) = 0).$$

An occurrence (or firing) of a transition t does not affect the marking of inhibitor places (if they are not in the t's output set).

Example. Fig.2.7 shows a Petri net model (inhibitor arcs have small circles instead of arrowheads) of the readers–writers synchronization problem, in which m reader processes and n writer processes access the same data in such a way, that any number of reader processes can access the data at the same time, but each writer process must have exclusive access to this data to perform an update operation. Moreover, writer processes have priority over reader processes, which means that when any writer process is ready to perform its write operation, no new reader processes can be granted access to the data, but reader processes which were granted their accesses earlier, continue their operation until completion, and then the ready writer process can proceed to access the data.

The cyclic reader processes are represented by the subnet $(p_1, t_1, p_2, t_2, p_3, t_3, p_4, t_4)$. The initial marking of place p_1 represents m, the number of reader processes. t_1 models the granting of access to data, t_2 represents accessing the data, and t_3 models release of the "access right". The subnet $(p_5, t_5, p_6, t_6, p_7, t_7, p_8, t_8, p_9, t_9)$ models the cyclic writer processes, in which t_5 registers (in p_{11}) that there is a writer process ready to perform an update operation, and then the inhibitor arc (p_{11}, t_1) blocks the granting of accesses (t_1) to subsequent reader processes. Each reader process which is granted access to data is "counted" in p_{12}; the inhibitor arc (p_{12}, t_6) delays the writer process (or processes) until all the reader processes complete their read operations (t_2), and release the "access rights" (by

removing a token from p_{12}). The write operation is performed by one process at a time due to a single token in p_{10}.

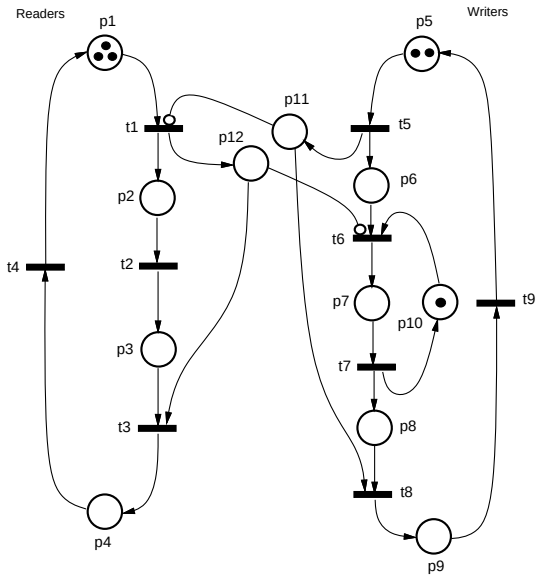

Fig.2.7. Readers–writers model.

It should be noted that nets with inhibitor arcs are more powerful than nets without such arcs [AF73]. Consequently, some results which are valid for nets without inhibitor arcs do not apply to inhibitor nets (for example, the condition on infinite set of reachable markings is not true for inhibitor nets).

2.3. Structural properties of nets

A place is *shared* if it is connected to more than one transition. A shared place is *guarded* if for every pair of transitions sharing it there exists another place which is connected by a directed arc to one of these two transitions and by an inhibitor arc to the other transition:

$$p \text{ is guarded } \Leftrightarrow \forall\, t_i, t_j \in Out(p)\ \exists\, p_k \in P : \\ p_k \in Inp(t_i) \wedge p_k \in Inh(t_j) \vee p_k \in Inp(t_j) \wedge p_k \in Inh(t_i).$$

If a place is guarded, at most one of the transitions sharing it can be enabled by any marking function.

If all shared places of a net are guarded, the net is (structurally) *conflict–free*, otherwise the net contains conflicts. The simplest case of conflicts is known as a *free–choice* (or *generalized free–choice*) structure; a shared place is (generalized) free–choice if all transitions sharing it have identical input and inhibitor sets:

$$p \text{ is free–choice } \Leftrightarrow \forall\, t_i, t_j \in Out(p) : Inp(t_i) = Inp(t_j) \wedge Inh(p_i) = Inh(t_j).$$

An inhibitor net is free–choice if all shared places are either guarded or free–choice. The transitions sharing a free–choice place constitute a free–choice class of transitions. For each

marking function, and each free–choice class of transitions, either all transitions in this class are enabled or none of them is. It is assumed that the selection of transitions for firing within each free–choice class is a random process which can be described by "choice probabilities" assigned to (free–choice) transitions. Moreover, it is usually assumed that the random variables describing choice probabilities in different free–choice classes are independent.

All places which are not conflict–free and not free–choice, are *conflict places*. Transitions sharing conflict places are (directly or indirectly) *potentially in conflict*:

$$t_i, t_j \text{ are potentially in conflict } \Leftrightarrow\ Inp(t_i) \cap Inp(t_j) \neq \emptyset \vee (\exists\ t_k \in T : Inp(t_i) \cap Inp(t_k) \neq \emptyset \wedge t_k, t_j \text{ are potentially in conflict}).$$

A conflict class is the set of all transitions which are potentially in conflict with each other:

$$T_k \subseteq T \text{ is a conflict class } \Leftrightarrow\ \forall\ t_i, t_j \in T_k : t_i, t_j \text{ are potentially in conflict.}$$

It should be observed that conflict classes are disjoint.

Example. Places s_1 and s_2 in Fig.2.5 are conflict places. A conflict of enabled transitions exists after executing $P(s_1)$ by process-1, or after executing $P(s_2)$ by process-2. It is assumed that conflicts are resolved by random choices of occurrences among the conflicting transitions.

The net shown in Fig.2.7 is conflict–free because it does not contain shared places.

A *siphon* in a net $\mathcal{N}$ is such a subset P_i of places, that all input transitions of places in P_i are also output transitions of some places in P_i, and there is a transition $t \in T$ which is an output transition of some place in P_i but is not an input transition of any place in P_i:

$$P_i \subset P \text{ is a siphon } \Leftrightarrow\ Inp(P_i) \subset Out(P_i).$$

If a marked net $\mathcal{M}$ contains a siphon which is not marked by the initial marking m_0, the siphon cannot be marked by any marking reachable from m_0, so the net cannot be live. If $\mathcal{M}$ contains a siphon P_i which is marked by m_0, the occurrence of the transition $t \in Out(P_i) - Inp(P_i)$ will remove the token form P_i, and then no token can enter P_i, so $\mathcal{M}$ also cannot be live.

A *trap* in a net $\mathcal{N}$ is such a subset P_j of places, that all output transitions of places in P_j are also input transitions of some places in P_j, and there is a transition $t \in T$ which is an input transition of some place in P_j but is not an output transition of any place in P_j:

$$P_j \subset P \text{ is a trap } \Leftrightarrow\ Out(P_j) \subset Inp(P_j).$$

If a marked net $\mathcal{M}$ contains a trap which is marked by the initial marking m_0, the trap will remain marked for any marking reachable from m_0. If $\mathcal{M}$ contains a trap P_j which is not marked by m_0, the occurrence of the transition $t \in Inp(P_j) - Out(P_j)$ will input a token into P_j, and P_j will remain marked in all subsequent markings.

Fig.2.8 shows a simple example of a (marked) siphon and a (marked) trap.

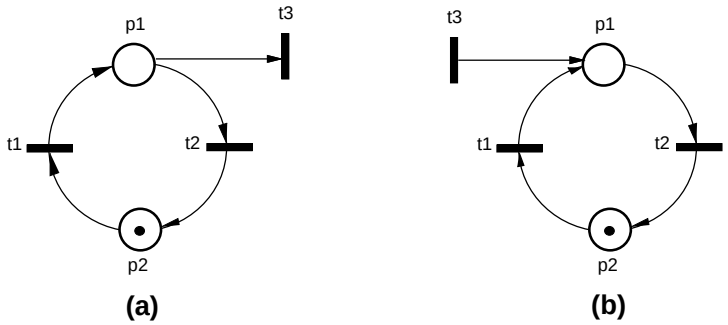

Fig.2.8. Example of a siphon (a) and a trap (b).

Properties of nets based on structural properties are disussed in [Be86, Mu89, Si77], while siphons and traps are discussed in greater detail in [ECS93, Mu89].

2.4. Net invariants

A net $\mathcal{N}_i = (P_i, T_i, A_i, B_i)$ is a P_i-implied *subnet* of a net $\mathcal{N} = (P, T, A, B)$, $P_i \subset P$, iff:

(1) $T_i \;=\; \{t \in T \mid \exists\, p \in P_i : (p,t) \in A \vee (t,p) \in A\}$,

(2) $A_i \;=\; A \cap (P_i \times T \cup T \times P_i)$,

(3) $B_i \;=\; B \cap (P_i \times T)$.

Each net $\mathcal{N} = (P, T, A)$ can be represented by a *connectivity matrix* (or *incidence matrix*) $\mathbf{C} : P \times T \to \{-1, 0, +1\}$ in which places correspond to rows, transitions to columns, and the entries are defined as:

$$\forall\; p_i \in P \;\forall\; t_j \in T : \mathbf{C}[i,j] = \begin{cases} -1, & \text{if } (p_i, t_j) \in A \wedge (t_j, p_i) \notin A, \\ +1, & \text{if } (t_j, p_i) \in A \wedge (p_i, t_j) \notin A, \\ 0, & \text{otherwise.} \end{cases}$$

If a marking m_j is obtained from another marking m_i by firing a transition t_k, then (in vector notation) $m_j = m_i + \mathbf{C}[k]$, where $\mathbf{C}[k]$ denotes the k-th column of $\mathbf{C}$, i.e., the column representing t_k. Similarly, if m_j is reached from m_i by a firing sequence $(t_{i_1} t_{i_2} ... t_{i_k})$, then $m_j = m_i + \mathbf{C}[i_1] + \mathbf{C}[i_2] + ... + \mathbf{C}[i_k]$.

Connectivity matrices ignore inhibitor arcs and disregard "selfloops", that is, pairs of arcs (p,t) and (t,p); any firing of a transition t cannot change the marking of p in such a selfloop, so selfloops are neutral with respect to token count of a net. A *pure net* is defined as a net without selfloops [Re85].

It should be observed that each P_i–implied subnet of $\mathcal{N}$ is described by the P_i subset of rows of the connectivity matrix of $\mathcal{N}$.

A *P–invariant* (place-invariant, sometimes also called S–invariant) of a net $\mathcal{N}$ is any integer positive (column) vector I which is a solution of the matrix equation

$$\mathbf{C}^T \times I = 0,$$

where $\mathbf{C}^T$ denotes the transpose of matrix $\mathbf{C}$. It follows immediately from this definition that if I_1 and I_2 are P–invariants of $\mathcal{N}$, then also any linear (positive) combination of I_1 and I_2 is a P–invariant of $\mathcal{N}$.

A *basic P–invariant* of a net is defined as a P–invariant which does not have simpler invariants. All basic P–invariants I are binary vectors [Re85], $I : P \rightarrow \{0, 1\}$.

It should be observed that in a pure net $\mathcal{N}$, each P–invariant I determines a P_I-implied (invariant) subnet of $\mathcal{N}$, where $P_I = \{p \in P \mid I[p] > 0\}$ is sometimes called the *support* of the invariant I; all nonzero elements of I select rows of $\mathbf{C}$, and each selected row i corresponds to a place p_i with all input (+1) and all output (–1) arcs associated with it.

Finding basic invariants is a classical problem of linear algebra, and there are known algorithms to solve this problem efficiently [KJ87, MS82].

Example: For the net shown in Fig.2.1, the connectivity matrix is:

$$\mathbf{C} = \begin{bmatrix} +1 & -1 & 0 & 0 \\ -1 & +1 & 0 & 0 \\ 0 & 0 & +1 & -1 \\ 0 & 0 & -1 & +1 \\ 0 & -1 & +1 & 0 \\ 0 & +1 & -1 & 0 \end{bmatrix}$$

It can be observed that the sums of rows 1 and 2, 3 and 4, and 5 and 6 are all equal to (vector) zero, so the basic P–invariants I for this net are [1,1,0,0,0,0], [0,0,1,1,0,0] and [0,0,0,0,1,1]; these P–invariants imply simple cyclic subnets (t_1, p_1, t_2, p_2), (t_3, p_3, t_4, p_4), and (t_2, p_6, t_3, p_5).

The connectivity matrix for the net shown in Fig.2.3 is:

$$\mathbf{C} = \begin{bmatrix} +1 & -1 & 0 & 0 \\ -1 & +1 & 0 & 0 \\ 0 & 0 & +1 & -1 \\ 0 & 0 & -1 & +1 \\ 0 & +1 & -1 & 0 \end{bmatrix}$$

There are only two basic P–invariants, [1,1,0,0,0] and [0,0,1,1,0]; p_5 does not belong to any of the P–invariants (p_5 is an unbounded place).

It can be shown that if a net is covered by simple P–invariants (i.e., if each element of a net belongs to one of the basic P–invariant implied subnets), the net is bounded. Moreover, if, in a net without inhibitor arcs, all P–invariant implied subnets are conflict–free and marked, the net is live.

A *T–invariant* (transition-invariant) of a net $\mathcal{N}$ is any integer positive (column) vector J which is a solution of the matrix equation

$$\mathbf{C} \times J = 0,$$

where $\mathbf{C}$ is the connectivy matrix of $\mathcal{N}$. A basic T–invariant is a T–invariant which does not contain simpler T–invariants. If the transitions of $\mathcal{N}$ fire in numbers indicated by the

elements of a T–invariant (in some order; the order is irrelevant), then the resulting marking is the same as the original one. So, each T–invariant corresponds to a sequence of transition firings which create a cycle of reachable markings.

Example. There is only one basic T–invariant for the net shown in Fig.2.1, $J = [1, 1, 1, 1]$. There is also one basic T–invariant for the net shown in Fig.2.3, $J = [1, 1, 1, 1]$. The two basic T–invariants for the net shown in Fig.2.7 are J_1=[1,1,1,1,0,0,0,0,0] and J_2=[0,0,0,0,1,1,1,1,1].

2.5. Simplifications of basic Petri nets

There are two types of net simplifications, structural simplifications and behavioral ones. In the first case, the classes of simplified nets are known as marked graphs, state machines, conflict–free nets, and free–choice nets. In the second case, there are bounded nets, safe nets, and a few other classes of nets.

A Petri net is a *marked graph* if each place has exactly one input and one output transition. Marked graphs can represent synchronization (by transitions with multiple inputs) but cannot represent decisions (represented by places with multiple outputs). Nets shown in Fig.2.1 and Fig.2.3 are marked graphs.

Marked graphs are often used as models of simple cyclic processes and their interactions (as in Fig.2.1). Their properties have been extensively studied in the literature [CC92, Mu77, Mu89, Pe81, TC87].

A Petri net is a *state machine* if each transition has exactly one input and one output place. State machines can represent decisions (by places with multiple outputs) but cannot model synchronization of activities. Since any firing of a transition in a state machine does not change the number of tokens, state machines are bounded and conservative.

State machines are especially useful as subnets covering a net. If a net is covered by a family of state machines, it is bounded. Some properties of state machines are discussed in [Mu89, Pe81, Re85].

Conflict–free nets are discussed in greater detail in [LR78, Ma87], and free–choice nets, in [Be86, DE95, ES91]. More general conflicts are described in [HV87, TS96].

2.6. Extensions of basic Petri nets

A popular extension of the basic model allows multiple arcs connecting places and transitions. A transition is enabled in such nets only if the number of tokens is at least equal to the number of directed arcs between a place and a transition. Formally this extension can be described by a "weight function" w which maps the set of directed arcs A into the set of positive numbers, $\mathcal{N} = (P, T, A, B, w)$, $w : A \to \{1, 2, ...\}$. Sometimes inhibitor arcs also have weights, in which case an inhibiting place can be associated with any number of tokens smaller than the value of the weight to allow the firing of a transition to occur; however, weights of inhibitor arcs are not considered here, which means that the implied weight of all inhibitor arcs is equal to 1.

In a net with multiple arcs (or arc weights), a transition t is enabled by a marking m if:

$$t \text{ is enabled by } m \;\Leftrightarrow\; (\forall\, p \in Inp(t) : m(p) \geq w(p,t)) \wedge (\forall\, p \in Inh(t) : m(p) = 0).$$

A transition t enabled by m can fire, transforming the marking m into m':

$$\forall\ p \in P:\ m'(p)\ =\ \begin{cases} m(p) + w(t,p), & \text{if } p \in Out(t) - Inp(t); \\ m(p) - w(p,t), & \text{if } p \in Inp(t) - Out(t); \\ m(p) + w(t,p) - w(p,t), & \text{if } p \in Out(t) \cap Inp(t); \\ m(p), & \text{otherwise.} \end{cases}$$

For nets with multiple arcs, the connectivity matrix contains the values of the weight function w labeling the arcs (instead of 0's and 1's), but otherwise the concepts are the same as for basic nets.

A *priority net* can be defined as a Petri net with an additional function which assigns a (numerical) level of priority to each transition. It is assumed that transitions with higher priority levels have higher priorities in firing.

A priority marked net is $\mathcal{M} = (\mathcal{N}, m_0)$, where $\mathcal{N} = (P, T, A, o)$, $o : T \to \{1, 2, ...\}$, and $m_0 : P \to \{0, 1, ...\}$. A transition t is enabled by m if all its input places are marked, all inhibiting places are unmarked, and if there is no enabled transition of higher priority:

$$t_i \text{ is enabled by } m\ \Leftrightarrow\ (\forall\ p \in Inp(t_i) : m(p) > 0)\ \wedge\ (\forall\ p \in Inh(t_i) : m(p) = 0)\ \wedge$$
$$(\forall\ t_j \in T : o(t_j) > o(t_i) \Rightarrow (\exists\ p_k \in Inp(t_j) : m(p_k) = 0)\ \vee\ (\exists\ p_\ell \in Inh(t_j) : m(p_\ell) > 0)).$$

Priority nets can be systematically converted into equivalent inhibitor nets [JK99].

Sometimes the definition of basic Petri nets includes *place capacities*, which determine the maximum numbers of tokens that can be assigned to places [Re85]; if an output place of a transition contains the number of tokens equal to its capacity, the transition cannot fire even if it is enabled. In this sense, the basic place/transition nets introduced earlier have infinite capacities.

Place capacities can easily be introduced in basic nets (with infinite capacities) by using *complementary places* with initial marking that complements the marking of the original place to the required capacity of the place. Fig.2.9 illustrates the idea of complementary places.

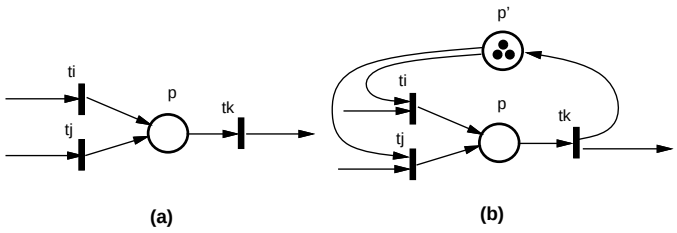

Fig.2.9. Introducing capacity 3 of place p through a complementary place p'.

2.6. Colored Petri nets

In colored Petri nets [Je87, Je92], tokens have attributes called colors. Token colors can be modified by (firing) transitions and also transitions can have several different firings (or variants of firing) for different combinations of colored tokens.

The basic idea of colored nets is to fold identical parts of a place/transition Petri net, and use the colors of tokens to indicate the parts the tokens belong to.

Each colored net can be systematically expanded to an equivalent ordinary (i.e., non–colored) net.

Formal definition of colored nets uses a convenient concept of *multisets* (or *bags*). A multiset is an extension of a set that allows multiple occurrences of the same elements; for any set $\mathbf{A}$, a multiset m on $\mathbf{A}$ is a function, $m : \mathbf{A} \rightarrow \{0, 1, ...\}$ which indicates the numbers of elements $a \in \mathbf{A}$. If the set $\mathbf{A}$ is ordered (e.g., by subscripting its elements, $\mathbf{A} = \{a_1, a_2, ..., a_n\}$), multisets can be represented by vectors, $m = [k_1, k_2, ..., k_n]$, where k_i is the number of elements a_i, $k_i = m(a_i)$, $i = 1, ..., n$.

A colored Petri net $\mathcal{N}$ can be defined as $\mathcal{N} = (P, T, A, C, a)$ where:

(P, T, A) is a Petri net structure,

C is a set of attributes called colors,

a is an arc labeling function, $a : A \rightarrow Expr(C, V)$, which assigns, to each arc of the net, an expression composed of colors (C), free variables (V) on the set of colors, and constants; expressions labeling the arcs determine the numbers and specific colors of tokens which are used for firing the transitions; free variables used in these expressions can represent any colors, but the same variable represents the same color in all arc expressions associated with the same transition; the selections of specific colors for free variables are called *bindings*.

A marked colored net $\mathcal{M}$ is defined as a pair, $\mathcal{M} = (\mathcal{N}, m_0)$, where $\mathcal{N}$ is a colored net, and the initial marking function m_0 assigns nonnegative numbers of (colored) tokens to places of $\mathcal{N}$, $m_0 : P \rightarrow C \rightarrow \{0, 1, ...\}$.

Example. The initial marking, in Fig.2.10, assigns 6 tokens to p_1 (one token of color a, two tokens of color b and three tokens of color c), and 4 tokens to p_2. Arc expressions associated with transition t require (at least) two tokens of (some) color x and one token of (some) color y in p_1, and (at least) one token of (the same) color x and two tokens of color y in p_2; if t fires, one token of color x and one of color y will be deposited in p_3.

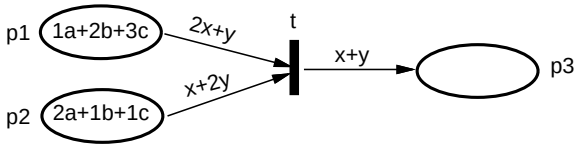

Fig.2.10. Occurrences in colored nets.

For the initial marking shown in Fig.2.10, there are two possible bindings for x and y: (1) $x = b$, $y = a$, and (2) $x = c$, $y = a$. After t's firing, the marking of p_3 becomes $1a + 1b$ for the first binding, or $1a + 1c$ for the second binding.

Colored nets are very convenient models of systems which contain many similar components, for example multiprocessor or distributed systems, because the components can be folded into a single subnet, significantly simplifying the model (but not its analysis).

Example. Fig.2.11 shows a model of "five dining philosophers". All philosophers, represented by colors a, b, c, d and e, follow the same cyclic behavior of thinking and eating. Place p_3 represents the (available) forks, in this case modeled by colors A, B, C, D and E. The two functions, "lf(x)" and "rf(x)" assign the left and right fork to each philosopher x, so, lf(a)=A, rf(a)=B, lf(b)=B, rf(b)=C, and so on.

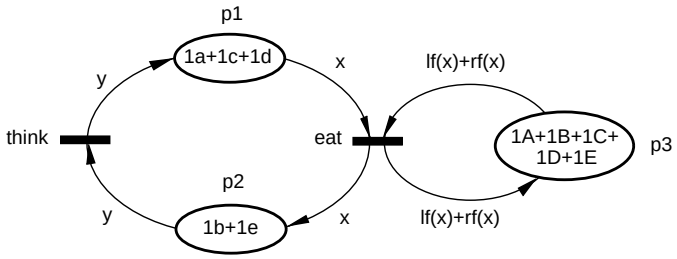

Fig.2.11. Colored net model of "five dining philosophers".

Colored Petri nets are quite convenient for modeling and analysis of distributed algorithms [Re99]. The Dijkstra's distributed termination detection algorithm [Di83] is used as an illustration of modeling using colored Petri nets.

The algorithm assumes that the N processors, $P_0, ..., P_{N-1}$, are connected in a ring, $P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3, ..., P_{N-1} \rightarrow P_0$, as shown in Fig.2.12(a), in which a token is transmitted from one processor to another checking if all processors have terminated their tasks. The token uses two colors, Black and White, to represent two states of the distributed system: the White color corresponds to the situation when all processors are found idle; the Black color represents the situation where some activity existed prior to the moment of checking, and, therefore, it cannot be concluded that the system is idle. The two token colors are distinguished, in Fig.2.12(a), by two connections between processors, one for White tokens (labeled by "w") and the other for Black tokens (labeled by "b"); the Black connection to P_1 is never used.

Each processor indicates its state, idle or active, by its color, White or Black, respectively. Whenever a processor induces any activity in the system by sending a data message, it also sets its color to Black. Processor P_0, whenever it becomes idle, initiates the termination detection by sending a White token to P_1. Each processor P_i, except of P_0, forwards the received token to P_{i+1} changing its color to Black if the processor is active, and preserving the token's color if the processor is idle. The token returning to P_0 is thus White only if all processors are idle, and this indicates the termination by the whole system; otherwise another termination detection cycle is initiated.

The "token control" in processor P_0 is shown in Fig.2.12(b). Place p_1 indicates that processor P_0 is active. Firing t_1 represents the completion of the execution of processor's task(s), and then firing t_2 sends a White "testing" token to processor P_1. When the "testing" token returns as Black, firing t_4 initiates another cycle of termination detection. If the returning "testing" token is White, firing t_3 indicates that the whole distributed system terminated is job.

Fig.2.12(c) shows the token control for all processors except of P_0. Again, place p_1 indicates that the processor is active, and then if the received "testing" token is White, it is forwarded as a Black token by firing t_2; if the received token is Black, it is forwarded as Black by firing t_3. The termination of processor's tasks is indicated by firing t_1, after which

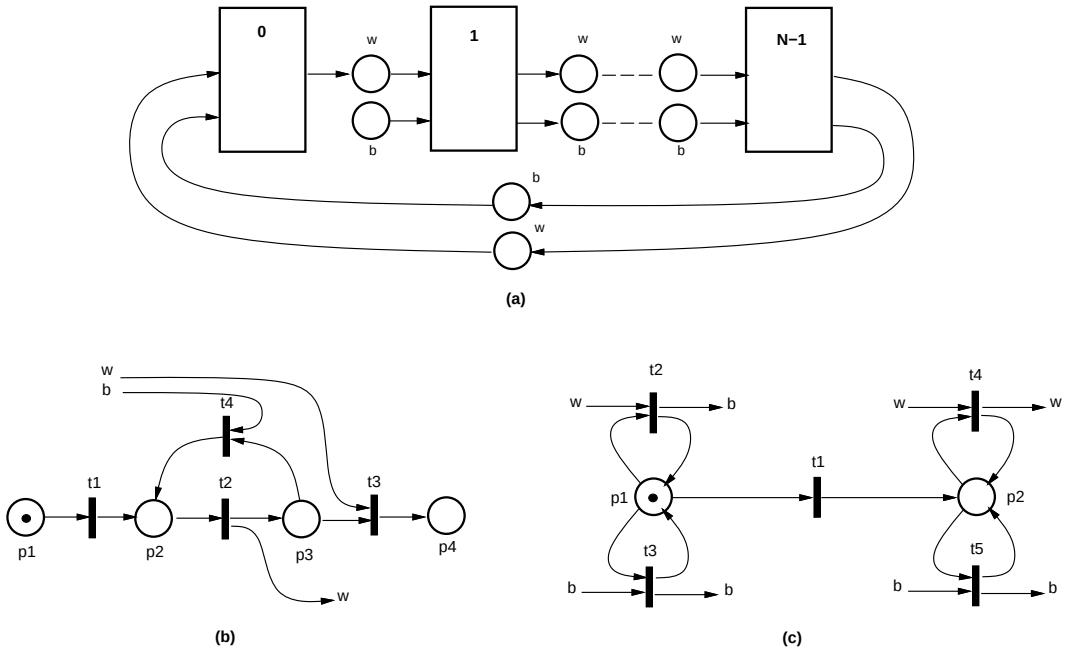

Fig.2.12. Termination detection in a distributed system.

the "testing" token is forwarded without changing its color, by firing t_4 or t_5, for White and Black colors, respectively.

A colored Petri net of the whole distributed system is shown in Fig.2.13; processor P_0 is represented by the upper part of the model (with t_1, t_2, t_3 and t_4 performing the same operations as in Fig.2.12(b)), while the lower part represents all remaining processors.

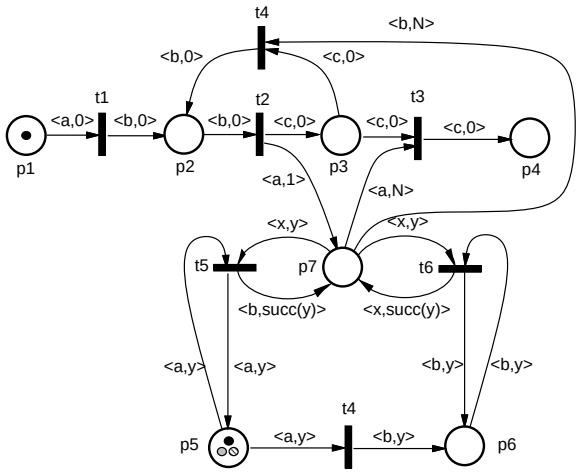

Fig.2.13. Colored net model of a distributed system (Fig.2.12(a)).

The color attributes of tokens are ordered pairs, $\langle x, i\rangle$, where x represents the active ("a") or idle ("b") processors and also the color of the "testing" token ("a" represents White, and "b" Black); moreover, "c" (in processor P_0) is used for the termination testing; the second component, i, identifies the processor, $i = 0, 1, ..., N-1$, and "succ(i)" is the

successor function.

Place p_7 represents the ring connection for passing the "testing" token. A White token is inserted into p_7 by firing t_2, and then this token is modified by consecutive processors by either firing t_5 if the processor is active (in which case the color of the token is changed to Black), or by firing t_6 if the processor is idle.

The initial marking assigns one token $\langle a, 0\rangle$ to p_1, and $N-1$ tokens, $\langle a, 1\rangle, \langle a, 2\rangle, ..., \langle a, N-1\rangle$ to p_5.

Since the information about the status of each processor is represented by the color ("a", "b" or "c"), Fig.2.13 can be further simplified by "merging" places p_1, p_2 and p_3 and also p_5 and p_6, as shown in Fig.2.14.

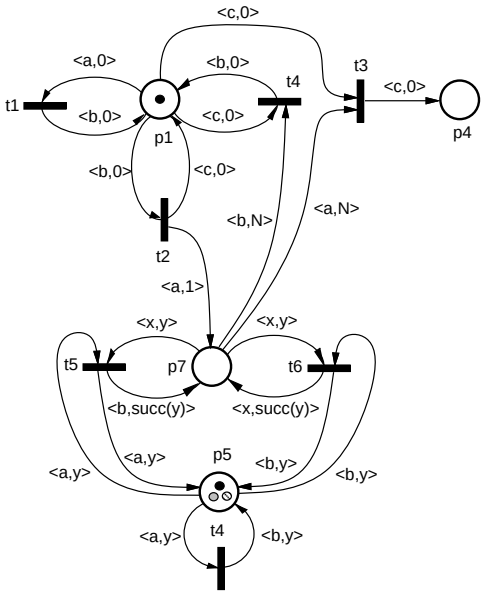

Fig.2.14. Simplified colored net model of a distributed system.

3. TIMED PETRI NETS

In timed nets, firing times are associated 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 transition cannot initiate their firing; for example, all transitions in a free–choice class can be enabled, but only one can fire). If, during the firing period of a transition, the transition becomes enabled again (as a result of completion of some other firing), 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 terminations. Sometimes, however, an initiated firing should be discontinued, as in the case of modeling processes with preemptions; if a lower–priority job is executing on a processor, and a higher–priority job needs the same processor for its execution, the execution of the lower–priority job must be suspended, and the processor allocated to the higher–priority job to allow its execution without any delay. The preempted job can continue only when the higher–priority job is finished (and no other higher–priority job is waiting). An extension to the basic model is needed to interrupt firing transitions; a special type of inhibitor arcs, called interrupt arcs, is used for this purpose. If, during the firing period of a transition, any place connected with this transition by an interrupt arc (such a place is called an interrupting place) receives a token, the firing discontinues, and the tokens removed from the transition's input places at the beginning of firing, are returned to these places (if there are several firings of the transition, the least recent one is discontinued; if there are several interrupting tokens, the corresponding number of the least recent firings are discontinued). Interrupt arcs are "special" inhibitor arcs, so they also disable transition's firings in the same way as inhibitor arcs do. Formally, the set of interrupt arcs, D, is added to the structure of the net as a subset of the set of inhibitor arcs, so $\mathcal{N} = (P, T, A, B, D)$, $D \subseteq B$. It should be noted that an effect similar to an interruption of a firing transition can be obtained by using a more complicated net with inhibitor arcs, so interrupt arcs are not a necessary extension; it is rather a convenient addition which simplifies the modeling process.

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; only when no more immediate transitions are enabled, the firings of (enabled) timed transitions are initiated (still in the same instant of time). 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. Also, the free–choice and conflict classes of transitions must be "uniform", i.e., all transitions in each such class must be either immediate or timed.

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

c is the conflict–resolution function, $c : T \to [0, 1]$, which assigns the probabilities of firings to transitions in free–choice classes of transitions, and relative frequencies of firings to transitions in conflict classes,

f is the firing–time function, $f : T \to \mathbf{R}^+$, which assigns the (average) firing times (or occurrence times) to transitions of the net.

The firing times of transitions can be constant (i.e., deterministic) or can be random variables with some probability distribution function; the (negative) exponential distribution is by far the most popular distribution for randomly distributed firing times.

3.1. D–timed Petri nets

In D–timed Petri nets, the firing times (or occurrence times) of transitions are constant, as defined by the firing–time function f. The behavior of (conflict–free) D–timed nets can be represented by timing diagrams, which illustrate the firing periods of transitions. Fig.3.1 shows such a diagram for the net of Fig.2.1, assuming that the firing time of t_1 is equal to 2 time units, $f(t_1) = 2$, that of t_2 and t_3 are equal to 0.5 time units, $f(t_2) = f(t_3) = 0.5$, and that of t_4 is equal to 2.5 time units. Fig.3.1 shows only the initial part of the diagram.

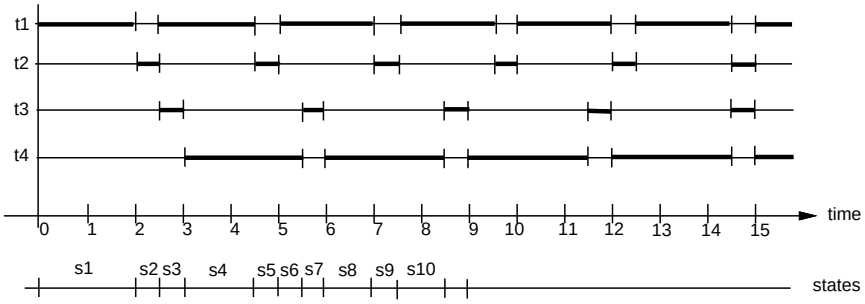

Fig.3.1. Timing diagram for the net shown in Fig.2.1
($f(t_1) = 2$, $f(t_2) = f(t_3) = 0.5$, $f(t_4) = 2.5$).

The behavior of a D–timed net can be described by states and state transitions. In Fig.3.1, states correspond to different configurations of the net, and state transitions occur when a firing of a transition terminates and possibly some new firings are initiated.

A state s of a D–timed net can be described by three functions [Zu91], $s = (m, n, r)$, where m is a marking function describing the distribution of tokens which are not involved in the firings of transitions (the remaining tokens), n is the firing–rank function which, for each transition of the net, indicates the number of its current firings, $n : T \rightarrow \{0, 1, ...\}$, and r is the remaining–firing–time function, which for each firing described by n specifies the time remaining to the completion of the firing (at the time instant in which the state begins).

Example. For the timing diagram in Fig.3.1, the first state, s_1 corresponds to the firing of transition t_1, and is described by (the components m, n and r are separated by semicolons):

$$s_1 = [0, 0, 1, 0, 2, 0; 1, 0, 0, 0; 2.0, 0, 0, 0].$$

When the firing of t_1 terminates, a token is deposited to p_1, and this enables t_2 which immediately initiates its firing, so the next state is:

$$s_2 = [0, 0, 1, 0, 1, 0; 0, 1, 0, 0; 0, 0.5, 0, 0].$$

After 0.5 time units the state changes to:

$$s_3 = [0, 0, 0, 0, 1, 0; 1, 0, 1, 0; 2.0, 0, 0.5, 0]$$

in which two transitions, t_1 and t_3 are occurring. t_3 first completes its firing, which enables t_4, so the next state is:

$$s_4 = [0, 0, 0, 0, 1, 0; 1, 0, 0, 1; 1.5, 0, 0.2.5]$$

and so on. The behavior of this model is cyclic, but there are 33 states before the cycle of three states is reached. The cycle time is determined by the subnet (t_3, p_3, t_4, p_4) in Fig.2.1, and is equal to 3 time units.

The set of all states that can be derived for a D–timed net $\mathcal{T}$ is called the set of reachable states, $\mathbf{S}(\mathcal{T})$. This set can be finite or infinite. It can be shown that if a marked net $\mathcal{M}$ is bounded, then all its timed extensions $\mathcal{T} = (\mathcal{M}, c, f)$ have finite sets of reachable states. On the other hand, if $\mathcal{M}$ is unbounded, than the set of reachable states can be finite or infinite, depending upon the firing times associated with transitions by the function f.

Example. For the unbounded net of Fig.2.3, with $f(t_1) = 2$, $f(t_2) = f(t_3) = 0.5$, and $f(t_4) = 1.5$, the sequence of states is shown in the following table (the component r of the state descriptions is not shown), in which column $h(s_i)$ shows the holding time of state s_i (i.e., the time spent in state s_i), and column j indicates the next state:

	m_i					n_i					
i	1	2	3	4	5	1	2	3	4	$h(s_i)$	j
1	0	0	1	0	0	1	0	0	0	2.0	2
2	0	0	1	0	0	0	1	0	0	0.5	3
3	0	0	0	0	0	1	0	1	0	0.5	4
4	0	0	0	0	0	1	0	0	1	1.5	2

The cycle time is equal to 2.5 time units and, in this case, is determined by the subnet (t_1, p_1, t_2, p_2).

It should be observed that the condition of (timed) boundedness for this net is that the consumer is not "slower" than the producer, i.e., $f(t_1) + f(t_2) \geq f(t_3) + f(t_4)$.

A *state graph* of a D–timed net $\mathcal{T}$ is a vertex and arc labeled directed graph $\mathcal{G} = (V, E, h, q)$ where:

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

E is a set of directed arcs, $E \subseteq V \times V$, such that $(s_i, s_j) \in E$ if and only if s_j is directly reachable from s_i,

h is a vertex labeling function which assigns the holding time $h(s)$ to each vertex $s = (m, n, r)$ of the graph, $h(s) = \min(r(t) : t \in T \wedge n(t) > 0)$,

q is the transition probability function, $q : E \rightarrow [0, 1]$.

A detailed description of the derivation of states can be found in [Zu91].

It should be observed that the state graph of a D–timed net is an embedded Markov chain [Fe78], so the stationary probabilities of the states can be obtained in the standard way [St94]. Many performance characteristics can be derived from the state graph of a net.

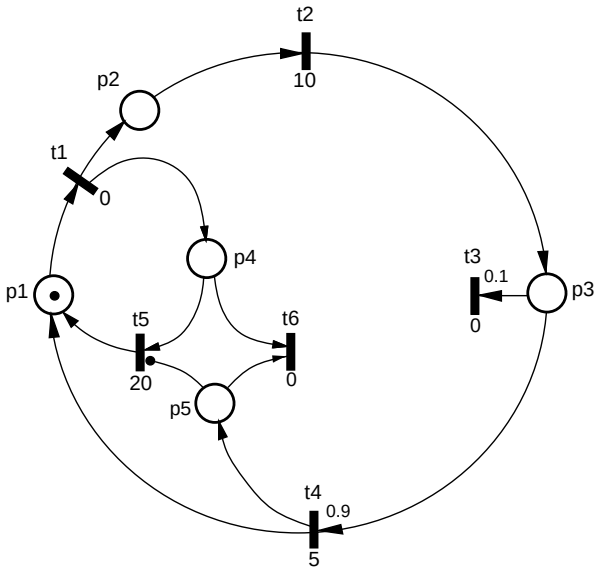

Fig.3.2. Petri net model of a simple communication protocol.

Example. A model of a very simple protocol with a timeout mechanism is shown in Fig.3.2 (interrupt arcs have blacks dots instead of arrowheads).

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 a firing of t_1, after which a single token is deposited in p_2 (the message) and in p_4 (the timeout). The firing time of t_2 represents the "transmission delay" of sending a message, and firing time of t_5, the timeout time. When the firing of t_2 is completed, 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 characteritransmission by "choice probabilities" assigned to t_3 and t_4 (0.1 and 0.9, respectively). t_3 represents (in a simplified way) the loss or distortion of the message or its acknowledgement; if 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_5 will complete its firing with no token in p_5; the termination of t_5's firing regenerates the lost token in p_1, so the message can be retransmitted. If the message is received correctly, t_4 is selected for firing rather than t_3, and after another transmission delay (modeled by t_4), tokens are deposited in p_5 and p_1 (so another message can be sent to the receiver). The token in p_5 interrupts the firing of t_5, so the "timeout token" is returned to p_4 and immediately removed by firing t_6.

The firing times of transitions must be selected in such a way that the timeout time ($f(t_5)$) is greater than the sum of the delays of sending a message ($f(t_2)$) and an acknowledgement ($f(t_4)$).

The set of reachable states for the net of Fig.3.2 is given in the following table, which, for each state s_i, shows the holding time $h(s_i)$, the next state j and the transition probability q_{ij}:

	m_i					n_i								
i	1	2	3	4	5	1	2	3	4	5	6	$h(s_i)$	j	q_{ij}
1	0	0	0	0	0	1	0	0	0	0	0	0.0	2	1.0
2	0	0	0	0	0	0	1	0	0	1	0	10.0	3	0.1
													4	0.9
3	0	0	0	0	0	0	0	1	0	1	0	0.0	5	1.0
4	0	0	0	0	0	0	0	0	1	1	0	5.0	6	1.0
5	0	0	0	0	0	0	0	0	0	1	0	10.0	1	1.0
6	0	0	0	0	0	1	0	0	0	0	1	0.0	2	1.0

The state graph for the net of Fig.3.2 is shown in Fig.3.3(a), in which the states with zero holding times (e.g., firing of t_1 or t_3) are represented by 'white' circles. The holding times of other states are shown as labels of the states. Transition probabilities are also shown where needed. The cycle time and other performance characteristics can easily be derived from this graph.

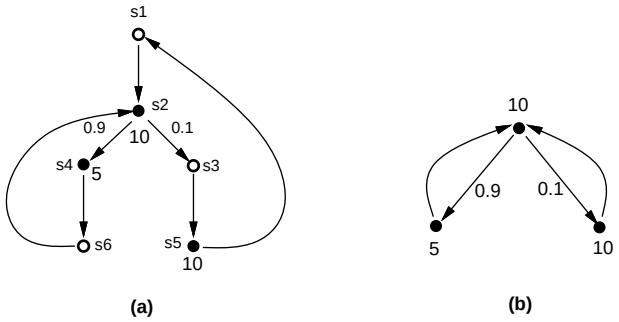

Fig.3.3. State graphs for the net shown in Fig.3.2; original (a) and reduced (b).

It should be noted that only a small modification of the net in Fig.3.2 is needed to represent a "sliding window" protocol, i.e., a protocol with several messages in different stages of transmission/acknowledgement or recovery.

States with holding times equal to zero (sometimes called *vanishing states*) do not contribute to the timed behavior of the net, so all such states can be eliminated from the state graph without any effect on the performance of the model. Such simplified model is shown in Fig.3.3(b). The vanishing states can be removed from the state graph, but it is also possible to eliminate them earlier, during the generation of the state graph. This second approach is used in *enhanced nets* [Zu91], in which the set of transitions is divided into two classes, timed and immediate transitions; immediate transitions fire in zero time (i.e., instantaneously), and it is assumed that the immediate transitions have priority over timed ones (so, during all changes of states, first one or more transitions complete their firings and deposit tokens to their output places, then all possible firings of immediate transitions occur, and finally, when no immediate transitions are enabled, the firings of timed transitions are initiated). Immediate transitions usually simplify the analysis by reducing, sometimes very significantly, the number of states of net models.

An enhanced timed net $\mathcal{T}$ is defined (similarly as before) as $\mathcal{T} = (\mathcal{M}, c, f)$, $\mathcal{M} = (\mathcal{N}, m_0)$, $\mathcal{N} = (P, T_i, T_t, A, B, D)$, and $f : T_t \rightarrow \mathbf{R}^+$, where T_i is the set of immediate

transitions, T_t is the set of timed transitions, and $T = T_i \cup T_t$. It is also assumed that all free–choice and conflict classes of transitions are "uniform", i.e., they are either immediate or timed, but not mixed.

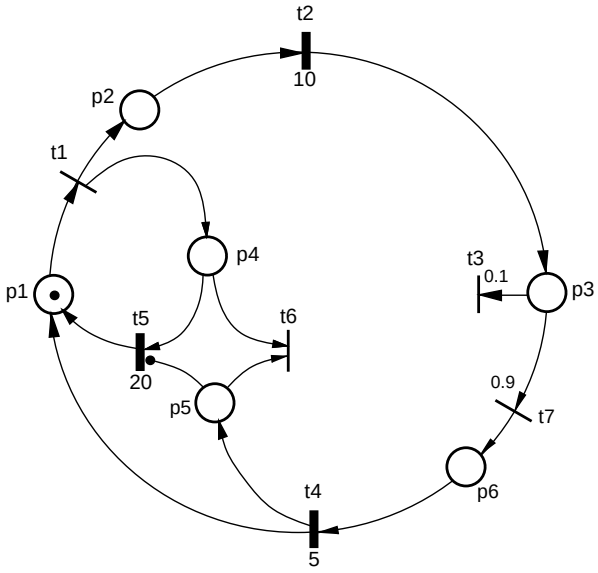

Fig.3.4. Enhanced Petri net model of a simple communication protocol.

Example. Fig.3.4 shows an enhaced version of the model shown in Fig.3.2 (immediate transitions are usually represented by "thin" bars while the timed ones by "thick" bars); the additional (immediate) transition t_7 and place p_6 may seem redundant, but actually they are needed to make the free–choice class (t_3, t_7) uniform. The state graph of this net is shown in Fig.3.3(b).

In some cases the performance of a net model can be derived from structural properties of nets, without the derivation of the state space (i.e., without the reachability analysis). In particular, if the net is covered by a set of simple basic P–invariants, then its cycle time is determined by the maximum cycle time of the subnets implied by the P–invariants:

$$\tau_0 \;=\; \max(\tau_1, ..., \tau_k)$$

where, for each simple subnet $\mathcal{N}_i = (P_i, T_i, A_i)$, the cycle time is:

$$\tau_i \;=\; \frac{\sum_{t \in T_i} f(t)}{\sum_{p \in P_i} m_0(p)}$$

Example. For the net shown in Fig.2.1, the cycle times of the three subnets implied by basic invariants are:

$$\tau_1 = 2.5,$$
$$\tau_2 = 0.25,$$
$$\tau_3 = 3.0,$$

so the cycle time of the model $\tau_0 = 3.0$.

Another approach, which sometimes can significantly simplify the analysis, is based on net transformations that preserve the behavior of the net. There is a variety of such transformations [B87, ES91]. Two more specialized transformations are shown in Fig.3.5. It should be noticed that these transformations preserve the state graphs of the original nets.

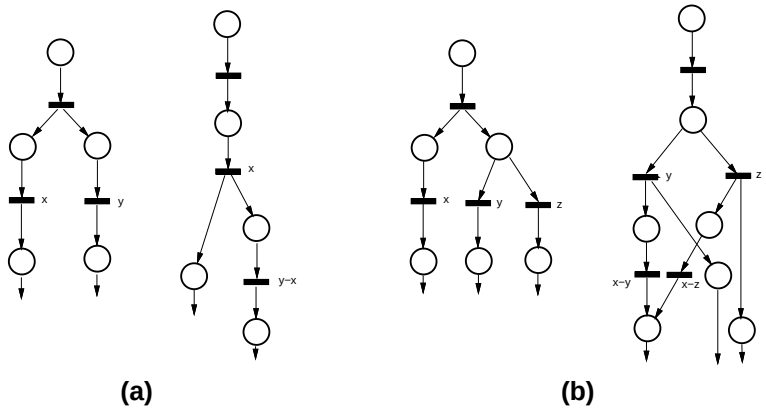

Fig.3.5. Simple net transformations; (a) $y > x$, (b) $x > y > z$.

Fig.3.6 shows a sequence of net transformations applied to the model of Fig.3.4.

Fig.3.6(a) is the result of applying the transformation of Fig.3.5(a) to transition t_1; the firing time of t_5 is adjusted by 10 (because of the firing time of t_2). Then the transformation shown in Fig.3.5(b) can be applied to transition t_2 in Fig.3.6(a), and the transformation shown in Fig.3.6(a) to transition t_7. The resulting model is shown in Fig.3.6(b). It can be observed that, in Fig.3.6(b), any firing of t_4 deposits tokens in p_4 and p_5, enabling the immediate transition t_6, which removes the deposited tokens from the net; consequently, t_6 and p_5 with all incident arcs, and also arc (t_4, p_4), can be deleted without any effect on the net's behavior. The remaining net is shown in Fig.3.6(c). The remaining transformation simply deletes the immediate transitions and their places since they are connected serially with timed transitions. The final net shown in Fig.3.6(d) is very simple, its state graph is shown in Fig.3.3(b), the stationary probabilities of states are 0.645, 0.290 and 0.065, for s_1 (at the top), s_2 (bottom left) and s_3 (bottom right), respectively, and then the throughput, θ, as the number of correctly delivered messages per time unit, can be obtained from the utilization of transition t_4 (which represents the acknowledgements, and which is firing in s_2), so $\theta = 0.290/5.0 = 0.058$ messages per time unit.

3.2. M–timed Petri nets

In M–timed Petri nets (or Markovian timed nets), the firing times (or occurrence times) of transitions are exponentially distributed random variables with the average times described by the values $f(t), t \in T$.

Example. Fig.3.7 shows a very simple model of an interactive computer system, in which p_1 represents the (idle) processor, t_1 models a processor executing a job, p_2 is the

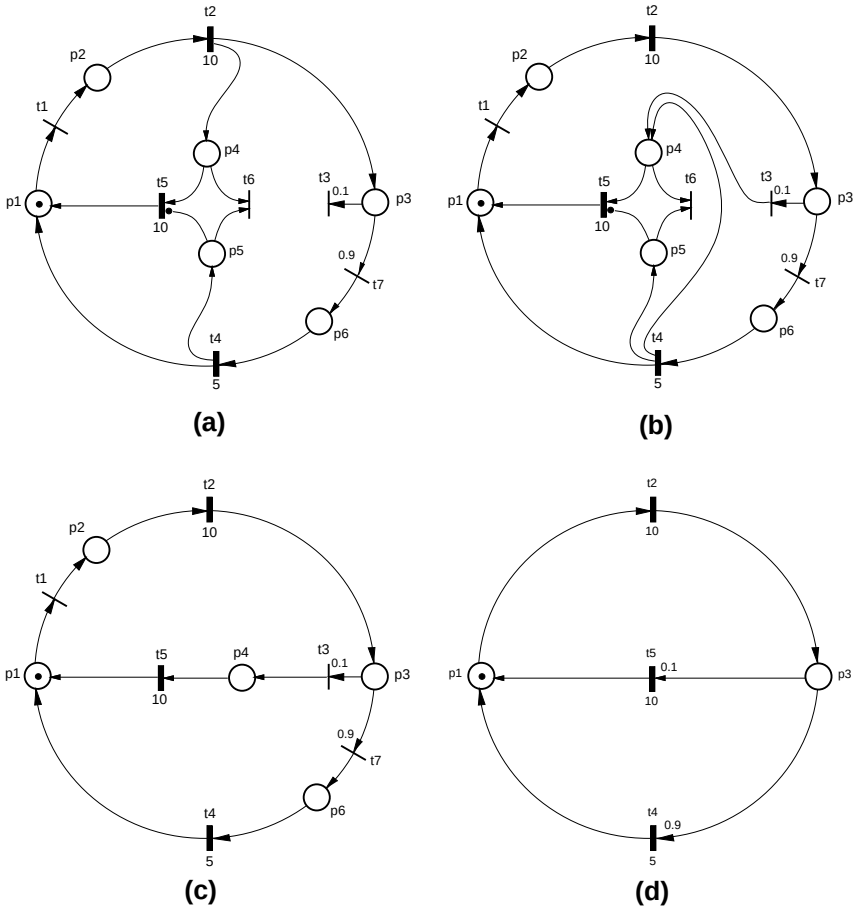

Fig.3.6. Transformations of the protocol model.

queue of jobs waiting for execution, t_2 represents the "thinking time" of users, and 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).

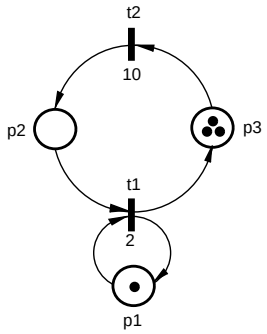

Fig.3.7. A simple model of an interactive system.

The initial marking function indicates one processor ($m_0(p_1)$) and three users ready to start their thinking phases (p_3).

The three initial tokens in p_3 initiate three independent firings of t_2, all exponentially distributed with parameter 0.1 (since the average firing time of t_2, $f(t_2)$, is equal to 10 time units). When one of these firings completes, a token is deposited in p_2, and this immediately (since p_1 is marked) starts a firing of t_1, which is also exponentially distributed (with the average time equal to 2 time units). If another firing of t_2 completes before the end of t_1's firing, the token will be deposited in p_2 waiting for its access to t_1, and so on. One of possible execution traces is shown as a timing diagram in Fig.3.8.

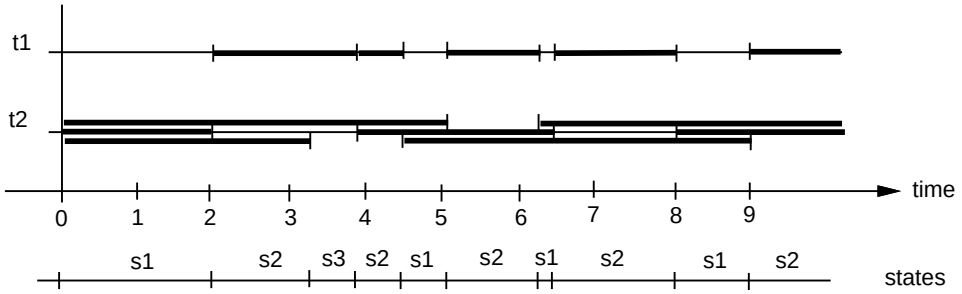

Fig.3.8. A sequence of possible events in the net shown in Fig.3.7.

It should be observed that, in the net shown in Fig.3.7, 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, with one token assigned to p_1, can have at most a single firing at any instant of time.

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.

A state of an M–timed net can be described by a pair of functions [Zu91], $s = (m, n)$, where m is a marking function describing the distribution of tokens which are not involved in firings of transitions, $m : P \rightarrow \{0, 1, ...\}$, and n is the firing–rank function which, for each transition of the net, indicates the number of its current firings, $n : T \rightarrow \{0, 1, ...\}$.

Example. For the net shown in Fig.3.7, the first state corresponds to three firings of t_2, so the first state is:

$$s_1 = [1, 0, 0; 0, 3].$$

When one of t_2's firings terminates, a new firing of t_1 is initiated, so the next state is:

$$s_2 = [0, 0, 0; 1, 2].$$

If, in s_2, the firing of t_1 completes before another firing of t_2, a token is deposited in p_3, and this immediately initiates another firing of t_2, so the state is again s_1. If, on the other hand, another firing of t_2 completes before that of t_1 (as shown in Fig.3.8), a token is deposited in t_2, and the state becomes:

$$s_3 = [0, 1, 0; 1, 1].$$

In s_3 there are also two possibilities, either t_1 first completes its firing, and then the next state is again s_2, or the remaining firing of t_2 completes first, and then the state becomes:

$$s_4 = [0, 2, 0; 1, 0]$$

in which the only possibility is to complete the firing of t_1, i.e., to return to state s_3.

As before, the set of all states that can be derived for an M–timed net $\mathcal{T}$ (i.e., the state space for $\mathcal{T}$) is denoted $\mathbf{S}(\mathcal{T})$.

A *state graph* of an M–timed net $\mathcal{T}$ is a directed arc–labeled graph $\mathcal{G} = (V, E, \ell)$ where:

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

E is a set of directed arcs, $E \subseteq V \times V$, such that $(s_i, s_j) \in E$ if and only if s_j is directly reachable from s_i,

ℓ is the transition rate function, $\ell : E \to \mathbf{R}^+$.

It should be noticed that state graphs of M–timed Petri nets are continuous–time Markov chains, so the stationary probabilities of states can be obtained using the standard techniques, and then many performance measures can be easily derived from stationary probabilities.

The rate of transitions between the states depend upon the probabilities of transitions, and these are composed of two effects:

- the probability that a particular firing will complete first (if there are more than one simultaneous firings); since all firing times are exponentially distributed, the probability that firing x will complete first is equal to the ratio of the rate of firings x and the sum of all rates of simultaneous firings;
- the probability of initiating new firings (if there are any new free–choice or conflict firings involved).

Example. The state graph for the net of Fig.3.7 is shown in Fig.3.9.

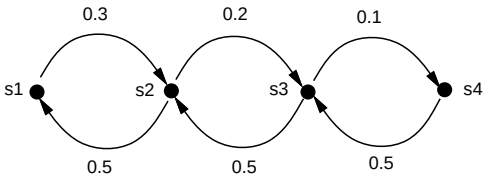

Fig.3.9. State graph for the net shown in Fig.3.7.

In the state s_1 in Fig.3.9 (and Fig.3.8), there are three simultaneous firings of transition t_2. It does not matter which one of these firings will complete first because they are identical; so the rate of transitions to state s_2 is equal to $3{*}0.1 = 0.3$. In s_2, either one of the remaining two firings of t_2 will complete first (as shown in Fig.3.8), or the firing of t_1 completes first; the probability that t_1's firing will complete first is equal to 0.5/0.7 (the rate of t_1's firings

is equal to 0.5, and the rate of each t_2's firings is equal to 0.1), so the probability that s_2 will change into s_1 is 5/7 and the rate of transitions from s_2 to s_1 is 0.5, the rate of firing t_1, while the rate of transitions from s_2 to s_3 is equal to $2 * 0.1 = 0.2$. The following table summarizes the states and state transitions, with column $h(s_i)$ showing the holding time of the state s_i (i.e., the average time spent in s_i), column j indicating the next state, and column q_{ij} showing the probability of transitions from state s_i to state s_j (the transition rates shown in Fig.3.9 are simply the ratios of q_{ij} over $h(s_i)$).

	m_i			n_i				
i	1	2	3	1	2	$h(s_i)$	j	q_{ij}
1	1	0	0	0	3	3.333	2	1.000
2	0	0	0	1	2	1.429	1	0.714
							3	0.286
3	0	1	0	1	1	1.667	2	0.833
							4	0.167
4	0	2	0	1	0	2.000	3	1.000

The state graph in Fig.3.9 is the Markov chain representing the behavior of the model shown in Fig.3.7.

The exponentially distributed firing times of transitions can be combined into hypo- and hyper-exponential distributions (and used for approximations of other distributions). Fig.3.10(a) shows a model of a two–stage hypo–exponential server, and Fig.3.10(b) a two–stage hyper–exponential server in which the two transitions form a free–choice structure, with "choice probabilities" describing random selections [Zu91].

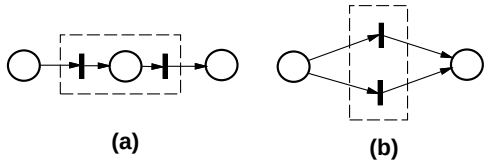

Fig.3.10. A model of a hypo-exponential (a) and hyper–exponential (b) server.

A different type of modification of the basic interactive model is shown in Fig.3.11; in this case, there are two classes of jobs (and users), say A and B; class A is represented by subnet (t_1, p_3, t_2, p_2) and class B by subnet (t_3, p_5, t_4, p_4). The processor is shared by both classes; either t_2 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 jobs of class A waiting in p_2 (non-preemptive priority scheduling).

It should be noted that if the inhibitor arc in Fig.3.11 is replaced by an interrupt arc, 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) when any job of class A becomes ready for execution.

Yet another modification of the basic model of an interactive system is shown in Fig.3.12; 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).

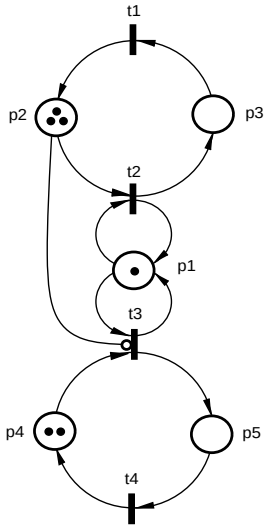

Fig.3.11. A system with two classes of jobs.

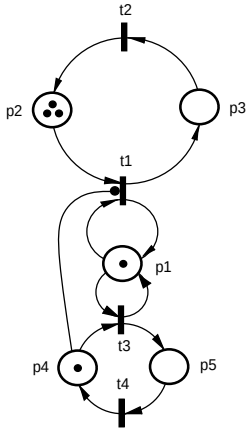

Fig.3.12. A system with unreliable processor.

The "operative–inoperative" cycle is represented by the subnet (t_4, p_4, t_3, 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; 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.

It should be observed that the net shown in Fig.3.12 is structurally similar to the net shown in Fig.3.11 (with an interrupt arc instead of the inhibitor arc); the model of processor failures is thus similar to a higher priority jobs that (conceptually) preempt the processor (for a failure and its repair).

3.4. Timed colored Petri nets

A timed colored net $\mathcal{T}$ is defined as a triple, $\mathcal{T} = (\mathcal{M}, c, f)$, where $\mathcal{M}$ is a marked colored net, $\mathcal{M} = (\mathcal{N}, m_0)$, c is the conflict–resolution function which assigns the choice probabilities to free–choice firings of transitions and relative frequencies to conflict firings of transitions in $\mathcal{N}$, $c : T \to C \to [0, 1]$, and f is the firing–time function which assigns the (average) firing time (or the occurrence time) to each occurrence of each transition of $\mathcal{N}$, $f : T \to C \to \mathbf{R}^+$.

A timed net model of a distributed memory multithreaded architecture [BH95, BR92, Mo96] is used as an illustration of using colored net models. The distributed memory system is composed of a number processors connected by an interconnection network; Fig.3.14(a) outlines a 16–processor system with a two–dimensional torus–like interconnection network.

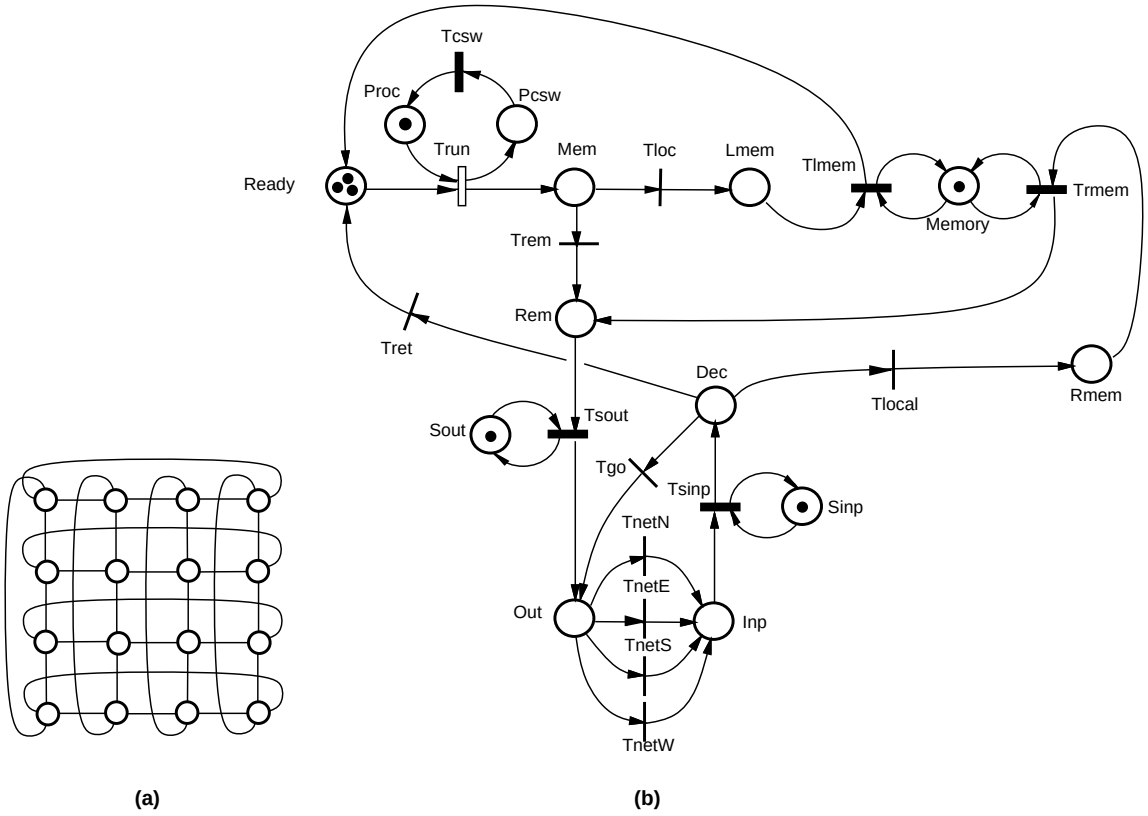

Fig.3.14. A colored net of a multithreaded multiprocessor.

Each node in Fig.3.14(a) is a multithreaded processor. Multithreading is an architectural approach to tolerating long–latency memory accesses and synchronization delay in distributed memory systems. Its general idea is very simple: instead of waiting for the completion of long–latency memory accesses (which in distributed memory systems can required hundreds and even thousands of processor cycles), the processor suspends the execution of the current thread and switches to another thread if such a thread is available (this process is called "context switching"). Since the threads are executed in the same address space,

context switching can be performed very efficiently in just a few processor cycles, especially if different sets of (hardware) registers are allocated to different threads.

Fig.3.14(b) shows a model of a multithreaded processor as well as its connection with the interconnection network (using two switches, $Tsinp$ for messages coming from the network, and $Tsout$ for the messages outgoing to other nodes). The interconnection network is represented by transitions $TnetN$, $TnetE$, $TnetS$ and $TnetW$, which model – for this particular interconnection network – connections to four neighboring nodes, NORTH, EAST, SOUTH and WEST, respectively. The processor shown in Fig.3.14(b) performs context switching for each long–latency memory access (local or remote); Petri net models of some other variants of multithreading are discussed in [Zu00].

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 and its average value represents the runlength of threads, ℓ_t, i.e., the average number of instructions executed before context switching occurs. Context switching is represented by transition $Tcsw$ with its firing time t_{cs}.

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. The memory cycle time, t_m, is assigned to both $Tlmem$ and $Trmem$.

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 adjacent nodes (transitions $TnetN$, ..., $TnetW$). Similarly, the traffic incoming to the node 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" in the interconnection network); 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. The delays introduced by the switches, t_s, are represented by firing times assigned to $Tsout$ and $Tsinp$.

Colors are used to fold the processors into a single model shown in Fig.3.14(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 [ZGS98].

The model shown in Fig.3.14 contains one transition with exponentially distributed firing times (*Trun*); all remaining timed transitions have deterministic firing times associated with them. Although it is possible to derive the state space for such a model, it should be observed that even for a small number of processor (e.g., 16), this space is very large.

Therefore, event–driven simulation was used to obtain performance characteristics of this model [Zu96b]. An example of such characteristics, presented in Fig.3.15, shows the utilization of each processor as a function of the number of available threads (i.e., the initial marking of *Ready*), and the probability of long–latency accesses to local memory, p_ℓ (i.e., the free–choice probability of $Tloc$), with fixed values of the remaining modeling parameters.

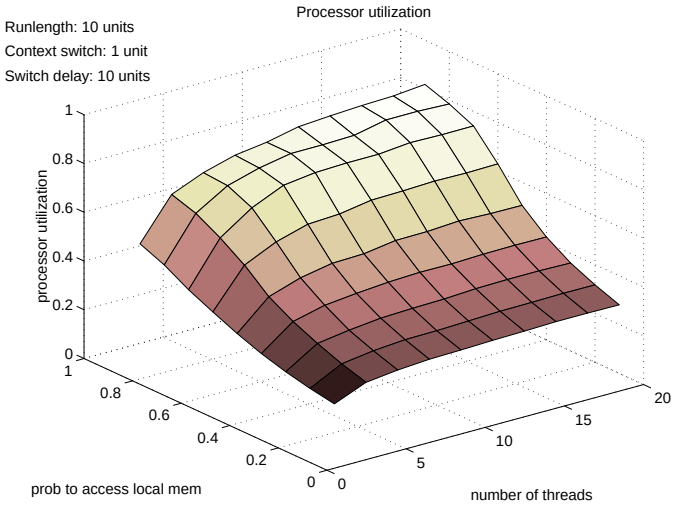

Fig.3.15. Processor utilization in a 16–processor system; $t_{cs} = 1, \ell_t = t_m = t_s = 10$.

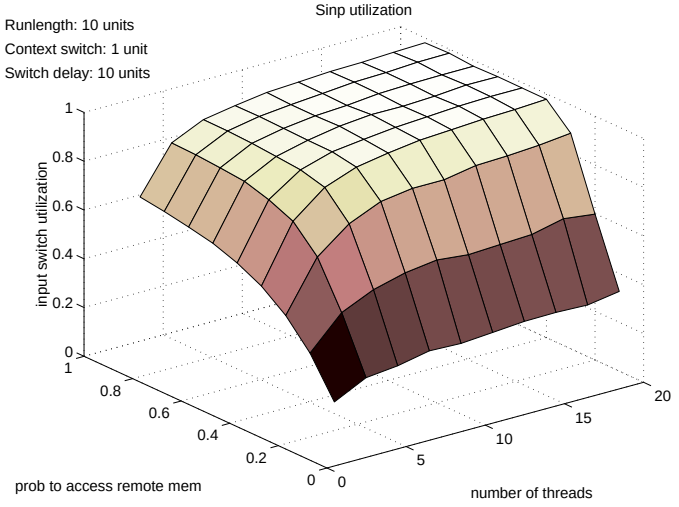

Fig.3.16. Input switch utilization in a 16–processor system; $t_{cs} = 1, \ell_t = t_m = t_s = 10$.

It can be observed that, for values of p_ℓ close to 1, the utilization increases with the number of available threads n_t, and tends to the bound 0.91 which, in this case, is determined by the ratio of $\ell_t/(\ell_t + t_{cs})$ since the context switching time, t_{cs}, is the overhead of multithreading. For smaller values of p_ℓ, the utilization of the processor "saturates" very quickly and is practically insensitive to the number of available threads. This is a clear indication that some other component of the system is the bottleneck, i.e., that it is utilized

in practically 100 % limiting the performance of the whole system.

It appears that for the analyzed 16–processor system, the input switch becomes the bottleneck for $p_\ell < 0.75$ [Zu00]. Indeed, Fig.3.16 shows the utilization of the input switch (for the same values of modeling parameters as in Fig.3.15); it should be noted that Fig.3.16 uses the probability of accessing remote memory, p_r, rather than p_ℓ used in Fig.3.15, so the "front part" of Fig.3.16 corresponds to the "back part" of Fig.3.15.

Fig.3.16 shows that the input switch enters its saturation quite quickly when the number of threads increases or when the value of p_r increases (i.e., the value of p_ℓ decreses). The "boundary" corresponding to $p_r = 0.25$ is clearly visible in Fig.3.16. The input switch is simply "too slow" if the probability of accesses to remote memory can be greater than 0.25.

Fig.3.17 shows the utilization of the processor for the case when the switch delay is one half of that used in Fig.3.15, i.e., $t_s = 5$; the processor's utilization is significantly better than in Fig.3.15, but the limiting effects of the input switch can still be observed for small values of p_ℓ.

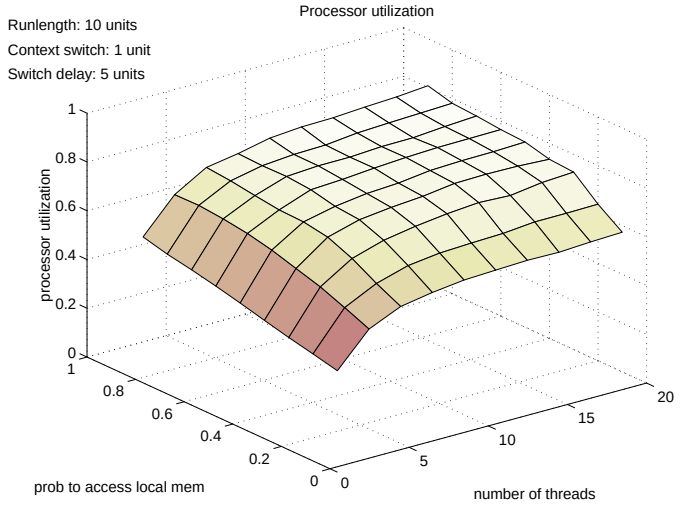

Fig.3.17. Processor utilization in a 16–processor system; $t_{cs} = 1, \ell_t = t_m = 10, t_s = 5$.

For performance analysis of derived models, the interconnection network is characterized by the average number of hops, n_h. Consequently, different networks characterized by the same value of n_h will yield the same performance characteristics of the nodes. For example, Fig.3.18 shows a hypercube network for a 16–processor system that can be used instead of a 2–dimension torus network shown in Fig.3.14(a). Since each node in Fig.3.18 is connected to 4 neighbors (as is the case in Fig.3.14(a)), the average numbers of hops for the two networks are the same, and then the performance characteristics for the two types of interconnection networks are identical.

One of the assumptions made to obtain the presented results was that accesses to memory are uniformly distributed over the nodes of the system. If this assumption is not realistic and some sort of "locality" is present, the only change that needs to be done is an adjustment of the value of n_h; for example, if the probability of accessing nodes decreases with the distance (i.e., nodes which are close are more likely to be accessed that the distant ones), the average value of n_h should be decreased.

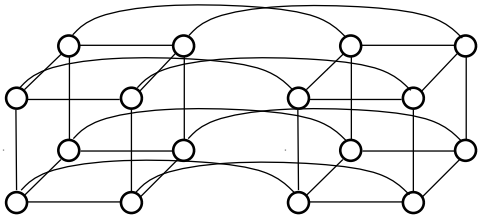

Fig.3.18. Outline of a 16–processor system.

Moreover, models of systems with different numbers of processors (e.g., 25, 36, etc.) require only minor adjustment of a few model parameters (the free–choice probabilities describing the traffic of messages in the interconnection network); otherwise the models are as presented earlier.

Further discussion of multithreaded models and their performance can be found in [Zu00, ZGS98].

4. CONCLUDING REMARKS

Because of complexity of real–life net models, high-level Petri nets are becoming increasingly popular in practical applications of Petri nets [FC99, GC98, RFH00, Wu99]. Compositionality of models, usually expressed by process algebras, often with temporal enhacements for performance analysis, is expected to provide elegant formal methods for complex realistic applications [An00, BDF00, HH00, Hi96, Ko00, La98, Si99, Wi00].

Available literature on theoretical and applied aspects of Petri nets is growing very quickly; a database of references to Petri net publications is maintained by the University of Hamburg, Germany:

`http://www.informatik.uni-hamburg.de/TGI/pnbib`

and general information on Petri nets, including available tools for analysis of net models – by University of Aarhus, Denmark:

`http://www.daimi.au.dk/PetriNets`

For more than 20 years, the "Annual Conference on Applications and Theory of Petri Nets" has been one of the focal points for Petri net researchers; for many years its proceedings have been published by Springer-Verlag in the series of Lecture Notes in Computer Science (vol.1825, 1639, 1420, 1248, 1091, and so on). Springer-Verlag, also in the series Lecture Notes in Computer Science, has been publishing "Advances in Petri Nets", an annual collection of selected contributions to the area of Petri nets. The "Conference on Petri Nets and Performance Models" (PNPM), organized every second year, is another survey of recent developments in the area of performance–related aspects of Petri net models. Several other conferences have special tracks or special sessions devoted to Petri nets; "IEEE Annual Conference on Systems, Man, and Cybernetics", "International Conference on Application of Concurrency to System Design", "IEEE Annual Conference on Emerging Technologies and Factory Automation" and "Annual High–Performance Computing Symposium" are good examples of such conferences. In addition, workshops are being organized

on specialized aspects of Petri nets, for example, "Workshop on Practical Use of Colored Petri Nets and Design/CPN" or "Workshop on Hardware Design and Petri Nets".

Finally, there is an increasing number of monographs on Petri nets and their applications, so the well–known Peterson's book [Pe81] and the Reisig's monograph [Re85] are now supplemented by several books on application of Petri nets to manufacturing systems [DA95, DC93, PX96, Zh95], on modeling using stochastic Petri nets [AM95, BK96, Li98], on colored Petri nets [Je91, Je95, Re99], and also on general aspects of some classes of Petri nets [DE95, Wa98].

References

[Aa93] van der Aalst, W.M.P., "Interval timed colored Petri nets and their analysis"; in: **Applications and Theory of Petri Nets 1993** (Lecture Notes in Computer Science 691); pp.453–472, Springer-Verlag 1993.

[Ag79] Agerwala, T., "Putting Petri nets to work"; IEEE Computer Magazine, vol.12, no.12, pp.85–94, 1979.

[AF73] 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.

[AB89] Ajmone Marsan, M., Balbo, G., Bobbio, A., Chiola, G., Conte, G., Cumani, A., "The effect of execution policies on the semantics and analysis of stochastic Petri nets"; IEEE Trans. on Software Engineering, vol.15, no.7, pp.832-846, 1989.

[AM00] Ajmone Marsan, M., Balbo, G., Conte, G., "The early days of GSPNs"; in: **Performance Evaluation: Origins and Directions** (Lecture Notes in Computer Science 1769), pp.505-512, Springer-Verlag 2000.

[AM95] Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G., **Modeling with Generalized Stochastic Petri Nets**; J. Wiley & Sons 1995.

[AM84] Ajmone Marsan, M., Conte, G., Balbo, G., "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.

[An00] Andova, S., "Time and probability in process algebra"; in: **Algebraic Methodology and Software Technology** (Lecture Notes in Computer Science 1816), pp.323–338, Springer-Verlag 2000.

[BDF00] Ballarini, P., Donatelli, S., Franceschinis, G., "Parametric stochastic well-formed nets and compositional modeling"; in: **Application and Theory of Petri Nets 2000** (Lecture Notes in Computer Science 1825), pp.43–52, Springer-Verlag 2000.

[BK96] Bause, F., Kritzinger, P.S., **Stochastic Petri Nets – and Introduction to the Theory**, Vieweg Verlag 1996.

[B87] Berthelot, G., "Transformations and decompositions of nets"; in: **Petri Nets: Central Models and Their Properties – Advances in Petri Nets 1986**, vol.1 (Lecture Notes in Computer Science 254), pp.359–376, Springer-Verlag 1987.

[Be86] Best, E., "Structural theory of Petri nets: the free–choice hiatus"; in: **Advances in Petri Nets 1986** (Lecture Notes in Petri Nets 254), pp.168–206, 1987.

[BP98] Bobbio, A., Puliafito, A., Telek, M., Trivedi, K.S., "Recent developments in non-Markovian stochastic Petri nets"; Journal of Circuits, Systems, and Computers, vol.8, no.1, pp.119–158, 1998.

[BR92] Boothe, B. and Ranade, A., "Improved multithreading techniques for hiding communication latency in multiprocessors"; Proc. 19-th Annual Int. Symp. on Computer Architecture, pp.214–223, 1992.

[BH95] Byrd, G.T., Holliday, M.A., "Multithreaded processor architecture"; IEEE Spectrum, vol.32, no.8, pp.38-46, 1995.

[CC92] Campos, J., Chiola, G., Colom, J.M., Silva, M., "Properties and performance bounds for timed marked graphs"; IEEE Trans. on Circuits and Systems, vol.39, pp.386–401, 1992.

[CA93] Chiola, G., Ajmone Marsan, M., Balbo, G., Conte, G., "Generalized stochastic Petri nets: a definition at the net level and its implications"; IEEE Trans. on Software Engineering, vol.19, no.2, pp.89-107, 1993.

[DE95] Desel, J., Esparza, J., **Free Choice Petri Nets** (Cambridge Tracts in Theoretical Computer Science 40); Cambridge University Press 1995.

[DA95] Desrochers, A.A., Al-Jaar, R.Y., **Applications of Petri Nets in Manufacturing Systems**; IEEE Press 1995.

[DC93] DiCesare, F., Harhalakis, G., Proth, J.M., Silva, M., Vernadat, F.B., **Practice of Petri Nets in Manufacturing**; Chapman and Hall 1993.

[Di83] Dijkstra, E., Feijen, W., van Gasteren, A., "Derivation of a termination detection algorithm for distributed computations"; Information Processing Letters, vol.16, no.5, pp.217–219, 1983.

[ES91] Esparza, J., Silva, M., "On the analysis and synthesis of free choice systems"; in: **Advances in Petri Nets 1990** (Lecture Notes in Computer Science 483), pp.243–288, Springer-Verlag 1991.

[ECS93] Ezpeleta, J., Couvreur, J.M., Silva, M., "A new technique for finding a generating family of siphons, traps and ST-components – application to colored Petri nets"; in: **Advances in Petri nets 1993** (Lecture Notes in Computer Science 674), pp.126–147, Springer-Verlag 1993.

[Fe93] Feldbrugge, F., "Petri net tool overview 1992"; in: **Advances in Petri Nets 1993** (Lecture Notes in Computer Science 674), pp.169-209, Springer-Verlag 1993.

[FC99] Feldmann, K., Colombo, A.W., "Monitoring of flexible production systems using high-level Petri net specifications"; Control Engineering Practice, vol.7, no.12, pp.1449-1466, 1999.

[Fe78] Ferrari, D., **Computer Systems Performance Evaluation**; Prentice–Hall 1978.

[HH00] Hermanns, H., Herzog, U., Klehmet, U., Mertsiotakis, V., Siegle, M., "Compositional performance modeling with the TIPPtool"; Performance Evaluation, vol.39, no.1-4, pp.5–35, 2000.

[Hi96] Hillston, J., **A Compositional Approach to Performance Modeling**; Cambridge University Press 1996.

[HV87] 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.

[GC98] Gonzalez, A., Crespo, A., "Modeling Ada95 components with high-level Petri nets"; Proc. IFAC/IFIP Workshop on Real-Time Programming (WRTP'98), pp.147-150, 1998.

[JK99] Janicki, R., Koutny, M., "On causality semantics of nets with priorities"; Fundamenta Informaticae, vol.38, no.3, pp.223–255, 1999.

[Je87] Jensen, K., "Coloured Petri nets"; in: **Advanced Course on Petri Nets 1986** (Lecture Notes in Computer Science 254), pp.248–299, Springer-Verlag 1987.

[Je92] Jensen, K., **Colored Petri Nets – Basic Concepts, Analysis Methods and Practical Use**, vol.1; Springer-Verlag 1992.

[Je95] Jensen, K., **Colored Petri Nets – Basic Concepts, Analysis Methods and Practical Use**, vol.2; Springer-Verlag 1995.

[Ko00] Koutny, M., "A compositional model of time Petri nets"; in: **Application and Theory of Petri Nets 2000** (Lecture Notes in Computer Science 1825), pp.303–322, Springer-Verlag 2000.

[KJ87] Krueckeberg, F., Jaxy, M., "Mathematical methods for calculating invariants in Petri nets"; in: **Advances in Petri Nets 1987** (Lecture Notes in Computer Science 266), pp.104–131, Springer-Verlag 1987.

[La98] Lamport, L., "Composition: a way to make proofs harder"; in: **Composability: The Significant Difference** (Lecture Notes in Computer Science 1536), pp.402–423, Springer-Verlag 1998.

[LR78] Landweber, L.H., Robertson, E.L., "Properties of conflict free and persistent nets"; Journal of the ACM, vol.25, no.3, pp.352–364, 1978.

[Li98] Lindemann, C., **Performance Modeling with Deterministic and Stochastic Petri Nets**; Wiley and Sons 1998.

[Ma87] Magott, J., "Performance evaluation of concurrent systems using conflict–free and persistent Petri nets"; Information Processing Letters, vol.26, no.1, pp.77–80, 1987.

[MS82] Martinez, J., Silva, M., "Simple and fast algorithm to obtain all invariants of a generalized Petri net"; in: **Applications and Theory of Petri Nets** (Informatik Fachberichte 52); pp.301–310, Springer-Verlag 1982.

[MF76] Merlin, P.M., Farber, D.J., "Recoverability of communication protocols – implications of a theoretical study"; IEEE Trans. on Communications, vol.24, no.9, pp.1036–1049, 1976.

[Mo82] M.K. Molloy, "Performance analysis using stochastic Petri nets"; IEEE Trans. on Computers, vol.31, no.9, pp.913–917, 1982.

[Mo96] Moore, S.W., **Multithreaded Processor Design**; Kluwer Academic Publishers 1996.

[Mu77] Murata, T., "Circuit theoretic analysis and synthesis of marked graphs"; IEEE Trans. on Circuits and Systems, vol.24, no.7, pp.400–405, 1977.

[Mu89] Murata, T., "Petri nets: properties, analysis and applications"; Proceedings of IEEE, vol.77, no.4, pp.541–580, 1989.

[Pe81] Peterson, J.L., **Petri Net Theory and the Modeling of Systems**; Prentice–Hall 1981.

[PX96] Proth, J.M., Xie, X., **Petri Nets**; Wiley & Sons 1996.

[Re85] Reisig, W., **Petri Nets - an Introduction** (EATCS Monographs on Theoretical Computer Science 4); Springer-Verlag 1985.

[Re99] Reisig, W., **Elements of Distributed Algorithms – Modeling and Analysis with Petri Nets**; Springer-Verlag 1999.

[RFH00] Rokyta, P., Fengler, W., Hummel, T., "Electronic system design automation using high level Petri nets"; in: **Hardware Design and Petri Nets**, pp.193-204, Kluwer Academic Publ. 2000.

[Si77] Sifakis, J., "Structural properties of Petri nets"; in: **Mathematical Foundations of Computer Science 1978** (Lecture Notes in Computer Science 64), pp.474–483, Springer–Verlag 1978.

[Si99] Sifakis, J., "The compositional specification of timed systems – a tutorial"; in: **Computer Aided Verification** (Lecture Notes in Computer Science 1633), pp.2–7, Springer-Verlag 1999.

[St94] Stewart, W.J., **Introduction to the Numerical Solution of Markov Chains**; Princeton University Press 1994.

[TS96] Teruel, E., Silva, M., "Structure theory of equal conflict systems"; Theoretical Computer Science, vol.153, no.1-2, pp.271–300, 1996.

[TC87] Thulasiraman, K., Comeau, M.A., "Maximum-weight markings in marked graphs: algorithms and interpretations based on simplex method"; IEEE Trans. on Circuits and Systems, vol.34, no.12, pp.1535–1545, 1987.

[Va82] Valk, R., "Test on zero in Petri nets"; in: **Applications and Theory of Petri Nets** (Informatik–Fachberichte 52), pp.193–197, Springer-Verlag 1982.

[Wa98] Wang, J., **Timed Petri nets**; Kluwer Academic Publ. 1998.

[Wi00] Winkowski, J., "Processes of timed Petri nets"; Theoretical Computer Science, vol.243, no.1-2, pp.1–34, 2000.

[Wu99] Wu, Z., "CEM/T net, a high level Petri net for FMS modeling"; International Journal of Intelligent Control Systems, vol.3, no.3, pp.377-387, 1999.

[Zh95] Zhou, M-C., **Petri Nets in Flexible and Agile Automation**; Kluwer Academic Publishers 1995.

[Zu91] Zuberek, W.M., "Timed Petri nets – definitions, properties and applications"; Microelectronics and Reliability (Special Issue on Petri Nets and Related Graph Models), vol.31, no.4, pp.627–644, 1991 (available through anonymous `ftp` at `ftp.cs.mun.ca` as `/pub/publications/91-MaR.ps.Z`).

[Zu96a] Zuberek, W.M., "Modeling using timed Petri nets – model description and representation"; Technical Report #9601, Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada A1B 3X5, 1996 (available through anonymous `ftp` at `ftp.cs.mun.ca/pub/techreports/tr-9601.ps.Z`).

[Zu96b] Zuberek, W.M., "Modeling using timed Petri nets – discrete–event simulation"; Technical Report #9602, Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada A1B 3X5, 1996 (available through anonymous `ftp` at `ftp.cs.mun.ca/pub/techreports/tr-9602.ps.Z`).

[Zu00] Zuberek, W.M., "Performance modeling of multithreaded distributed memory architectures"; in: **Hardware Design and Petri Nets**, pp.311–331, Kluwer Academic Publishers 2000.

[ZGS98] Zuberek, W.M., Govindarajan, R., Suciu, F., "Timed colored Petri net models of distributed memory multithreaded multiprocessors"; Proc. Workshop on Practical Use of Colored Petri Nets and Design/CPN, Aarhus, Denmark, pp.253-270, 1998.