Petri Nets and Timed Petri Nets in Modeling and Analysis of Concurrent Systems – An Overview

Włodek M. Zuberek

Abstract— **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 standard methods. These stationary probabilities are 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 generation and analysis of huge state spaces (in some models the number of states increases 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 paper overviews basic concepts of Petri nets, introduces timed Petri nets, and provides brief summaries of several case studies of performance analysis which are discussed in greater detail in other publications of the author.

Keywords—**Petri nets, timed Petri nets, performance analysis, reachability analysis, structural analysis, net transformations, multithreaded multiprocessors, distributed–memory multiprocessors, event–driven simulation.**

I. Introduction

PETRI nets have been proposed (by Carl Adam Petri [49]) as a simple and convenient formalism for modeling systems that exhibit concurrent activities [2], [47], [48], [54]. 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.

Petri nets are bipartite directed graphs, in which the two types of vertices, called places and transitions, represent, in a very general sense, conditions and events (sometimes Petri nets are also called condition–event systems). 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. In effect, an occurrence of one event causes some other event(s) to occur, and so on.

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 "occurrence times" (or "firing times") to the transitions or places of a net. In *timed* nets [65], [70], [83], 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). In *stochastic* (and *generalized stochastic*) Petri nets [5], [7], [44] and their many variants [4], [13], [18], (exponentially distributed) firing times are associated with transitions, but the tokens remain (for the occurrence time) in places, and the instantaneous occurrences occur at the end of occurrence times (so the "occurrence times" are actually "enabling times"). In *time* nets [1], [43] there is an interval associated with a transition, and the (instantaneous) occurrence 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 a 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 models [78], [79], [83].

Part of the Record of Research Forum 2003 organized on November 13, 2003 on the occasion of the 25-th Anniversary of the Department of Computer Science at Memorial University,

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada through Grant RGPIN-8222.

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 [35], [36], [37] 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 its firing for different combinations of token colors.

The basic idea of colored nets is to "fold" an ordinary Petri net into a simpler one. 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 a large number of colors and 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* and 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, 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 unless software tools for analysis of such models are available. It is, therefore, not surprising that many different tools have been developed for analysis of a variety of net types. 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 [86].

Timed Petri nets are discrete–event models which can be continuos–time (M–timed nets) or discrete–time (D–timed nets). Analysis of timed models by event–driven simulation of their behavior is yet another approach to performance analysis, which imposes very few restrictions on the class of analyzed models [87], [94] (e.g., both continuos–time and discrete–time elements can be used in the same model).

This paper first reviews basic concepts of Petri nets, then introduces timed Petri nets, and, as an illustration of their applications, summarizes a few case studies which are described in greater detail in other publications of the author. A brief information on activities of Petri net community concludes the paper.

II. Basic Concepts of Petri Nets

Place/transition Petri nets are 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 (sometimes also called *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:

$$\begin{aligned} Inp(t) &= \{p \in P \mid (p,t) \in A\}, \\ Inp(p) &= \{t \in T \mid (t,p) \in A\}, \\ Out(t) &= \{p \in P \mid (t,p) \in A\}, \\ Out(p) &= \{t \in T \mid (p,t) \in A\}. \end{aligned}$$

A. 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).

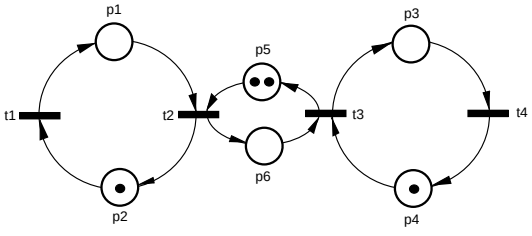

Fig.2.1. Producer–consumer bounded–buffer model.

The capacity of the buffer is represented by the total (initial) marking of places p_5 and p_6, so it is 2 in this case.□

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 $m(p) > 0$, 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 transitions 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. 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, $m \overset{t}{\mapsto} m'$:

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

A marking m_j is *generally reachable* (or just reachable) from a marking m_i in $\mathcal{M}$, $m_i \overset{*}{\mapsto} m_j$, 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:

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

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, $\ell : E \to 2^T$:

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

Example. The graph of reachable markings 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, the net $\mathcal{M}$ is *bounded*, otherwise the net is *unbounded*:

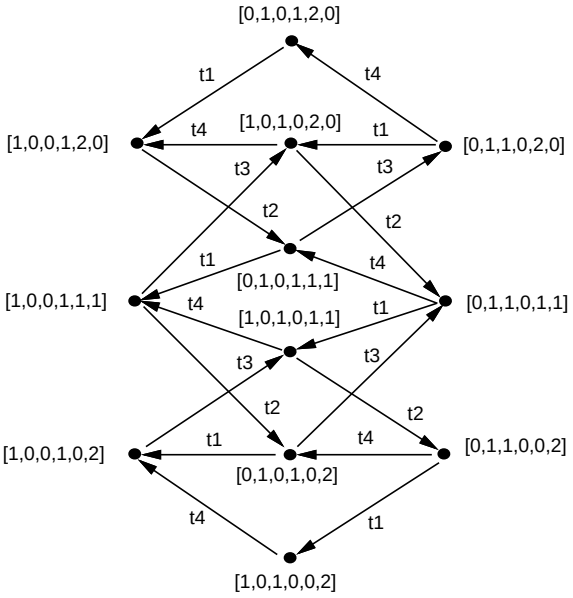

Fig.2.2. Graph of reachable markings for the net of Fig.2.1.

$\mathcal{M}$ is bounded $\Leftrightarrow$

$$\exists\; k > 0\; \forall\; m \in \mathbf{M}(\mathcal{M})\; \forall\; p \in P : m(p) \le k.$$

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, $m_j \rhd m_i$, 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):

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

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, the producer process does not depend upon the consumer process. 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. □

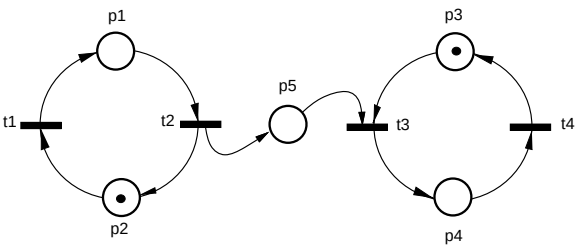

Fig.2.3. Producer–consumer unbounded–buffer model.

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 $\mathcal{M}$ 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.2.4 shows a simple Petri net model of 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 $V(s_i)$.

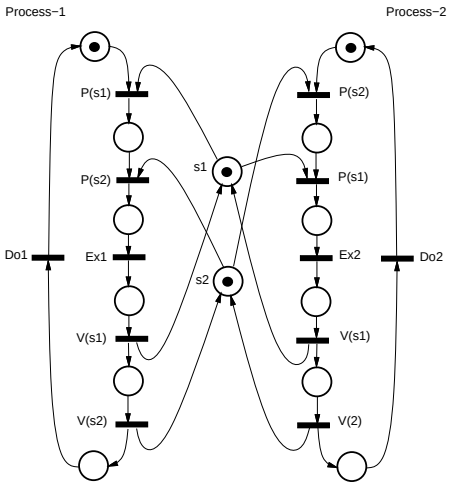

Fig.2.4. Resource allocation model.

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.2.4 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 $P(s_2)$).

A partial graph of reachable markings for this resource allocation model is shown in Fig.2.5. The graph contains a node with no outgoing arcs which 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. An extensive discussion of Petri net models of synchronization mechanisms is given in [91]. □

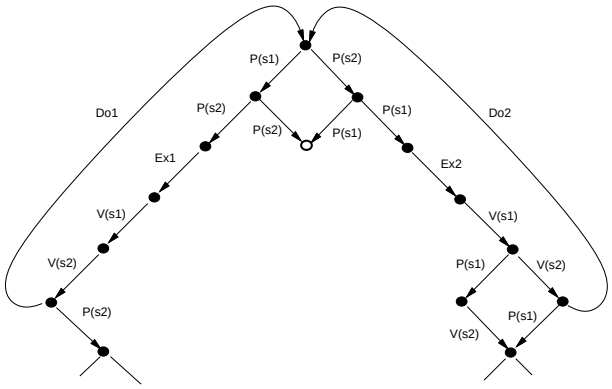

Fig.2.5. Partial graph of reachable markings for Fig.2.4.

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

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

Conservative nets are (obviously) bounded.

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 occur):

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

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.2.4 is bounded, non–live (it contains a deadlock) and non–conservative. □

A net which does not contain a deadlock is not necessarily live; it may contain a *livelock*, i.e., a subset of transitions which can occur, but which exclude occurrences of other transitions.

B. Inhibitor Nets

An important extension of the basic net model is addition of *inhibitor arcs* [3], [33], [64]. Inhibitor arcs (which connect places with transitions) provide a "test if zero" condition which does not exist in basic Petri 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. Nets with inhibitor arcs are usually called *inhibitor nets*.

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.2.6 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 sometimes earlier, continue their operation until completion, and then the ready writer process can proceed to access the data.

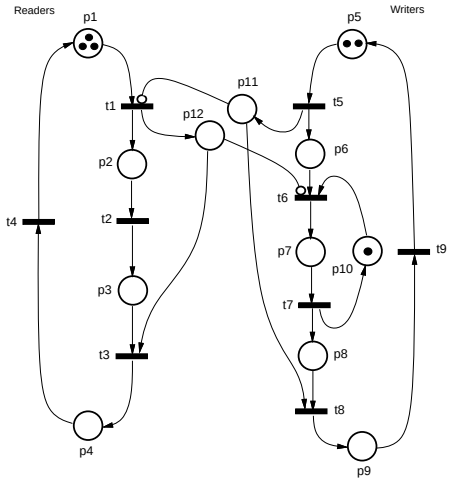

Fig.2.6. Readers–writers model.

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 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}. □

It should be noted that nets with inhibitor arcs are more powerful than nets without such arcs [3]. 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).

C. 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:

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

All conflict classes are disjoint. It is assumed that conflicts are resolved by random choices of occurrences among the conflicting transitions. These random choice are independent in different conflict classes.

Example. Transitions t_i and t_j, which are explicitly potentially in conflict, are shown in Fig.2.7(a), while Fig.2.7(b) shows t_i and t_j which are implicitly potentially

in conflict. Whether or not transitions, which are potentially in conflict, are actually in conflict depends upon the marking; in Fig.2.7(a), if only p_a and p_b are marked, the transition t_i is conflict–free; if p_c is also marked, t_i is a conflict transition.

Places s_1 and s_2 in Fig.2.4 are conflict places.

The net shown in Fig.2.6 is conflict–free because it does not contain shared places. □

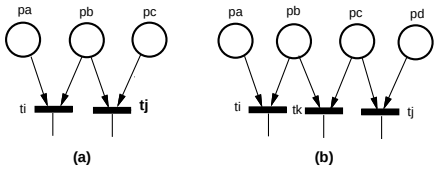

Fig.2.7. Examples of conflicts.

Properties of nets based on structural properties are discussed in greater detail in [12], [23], [24], [47], [59], [63].

An important structural concept is known as place–invariants.

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:

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

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 [54].

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 [54], $I : P \to \{0, 1\}$.

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 (elements "+1") and all output (elements "–1") arcs associated with it.

There are efficient algorithms to find all basic invariants of a net [39], [42].

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). □

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) $A_i = A \cap (P_i \times T \cup T \times P_i)$, and
(3) $B_i = B \cap (P_i \times T)$.

Each P_i–implied subnet of $\mathcal{N}$ is described by the P_i subset of rows of the connectivity matrix of $\mathcal{N}$.

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 connectivity 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 represents 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.2.6 are J_1=[1,1,1,1,0,0,0,0,0,0] and J_2=[0,0,0,0,1,1,1,1,1,1]. □

D. 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 [46], [47], [48], [62].

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 conservative and thus bounded.

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 [47], [48], [54].

Conflict–free nets are discussed in greater detail in [41], and free–choice nets, in [12], [19], [23]. More general conflicts are described in [30], [63].

E. 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 weight of the inhibitor arc to allow the transition to occur; in this paper, however, all inhibitor arcs are assumed to have weights 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':

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

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.

Priority nets can be systematically converted into equivalent inhibitor nets [34].

Sometimes the definition of basic Petri nets includes *place capacities*, which determine the maximum numbers of tokens that can be assigned to places [54]; 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.

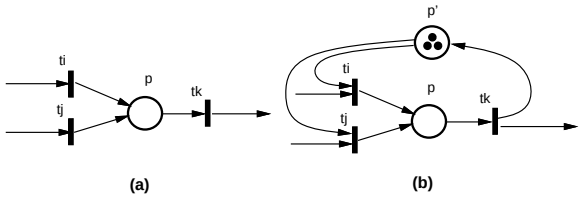

Fig.2.9. Introducing capacity 3 of place p through a complementary place p'.

F. Colored Petri Nets

In colored Petri nets [35], [36], tokens have attributes called colors. Token colors can be modified by (firing) transitions and also transitions can have several different occurrences (or variants of firing) 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, $m : \mathbf{A} \to \{0, 1, ...\}$ which indicates the numbers of elements a in m, $a \in \mathbf{A}$. 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, $a : A \to Expr(C, V)$, 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 $\mathcal{N}$, $m_0 : P \to C \to \{0, 1, ...\}$.

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.

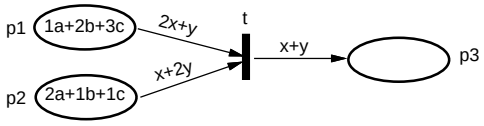

Fig.2.10. Occurrences in colored nets.

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. □

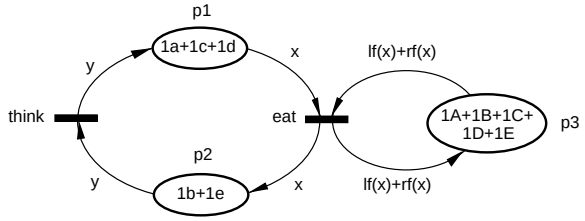

Fig.2.11. Colored net model of "five dining philosophers".

Colored Petri nets are quite convenient for modeling and analysis of distributed algorithms [55]. The Dijkstra's distributed termination detection algorithm [22] 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, $P_0 \to P_1 \to P_2 \to P_3, ..., P_{N-1} \to P_0$, 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.

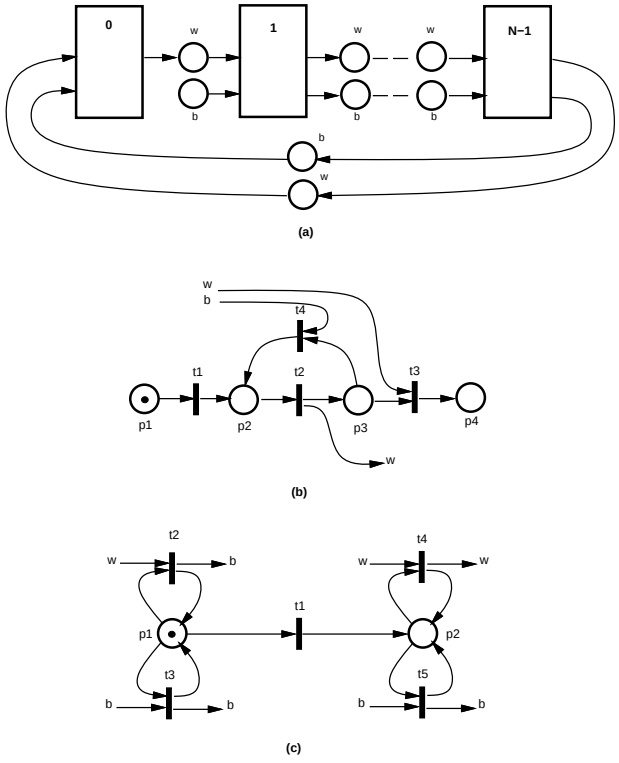

Fig.2.12. Termination detection in a distributed system.

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 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. 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.

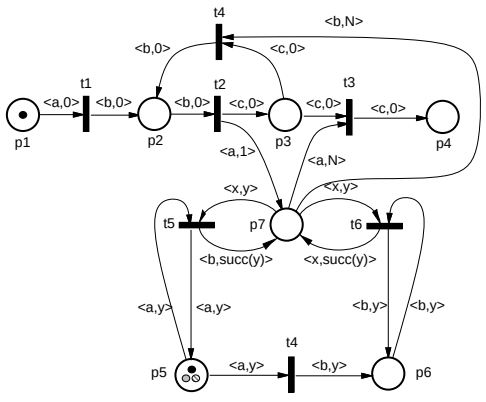

Fig.2.13. Colored net model of a distributed system (as shown in Fig.2.12(a)).

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.

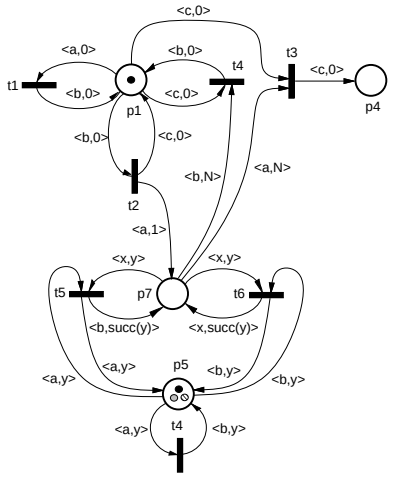

Fig.2.14. Simplified colored net model of Fig.2.13.

III. 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, in *enhanced timed Petri nets* the set of transitions is split 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:
- $\mathcal{M}$ is a marked net, $\mathcal{M} = (\mathcal{N}, m_0)$;
- 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.

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 $f : T_t \to \mathbf{R}^+$, 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.

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.

A. D–timed Petri Nets

In D–timed Petri nets [75], [77], [79], [80], [83], 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 the firing times of t_2 and t_3 are equal to 0.5 time units, $f(t_2) = f(t_3) = 0.5$, and that the firing time of t_4 is equal to 2.5 time units. Fig.3.1 shows only the initial part of the diagram.

Formally, 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 [79], [83], $s = (m, n, r)$, 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, $n : T \to \{0, 1, ...\}$, 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 state 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:

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

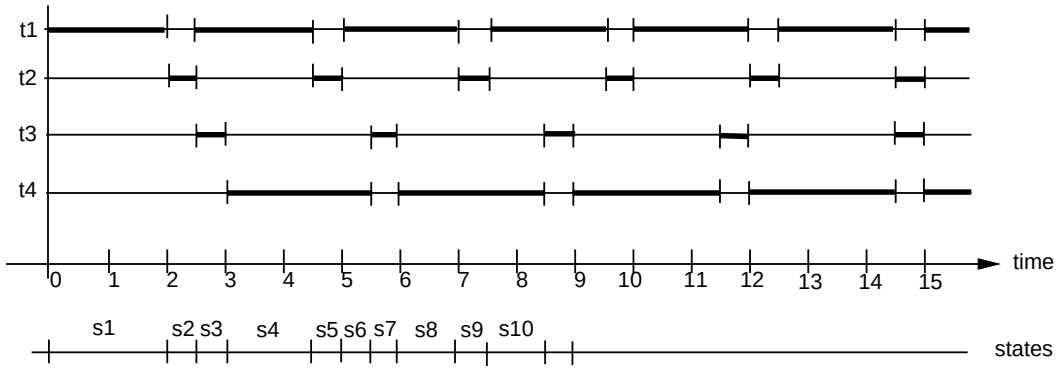

Fig.3.1. Timing diagram for the net shown in Fig.2.1 ($f(t_1) = 2$, $f(t_2) = f(t_3) = 0.5$, $f(t_4) = 2.5$).

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, $q : E \to [0, 1]$.

The state graph of a D–timed net is an embedded Markov chain, so the stationary probabilities of the states can be obtained in the standard way [61]. Many performance characteristics can be derived from the state graph of a net.

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).

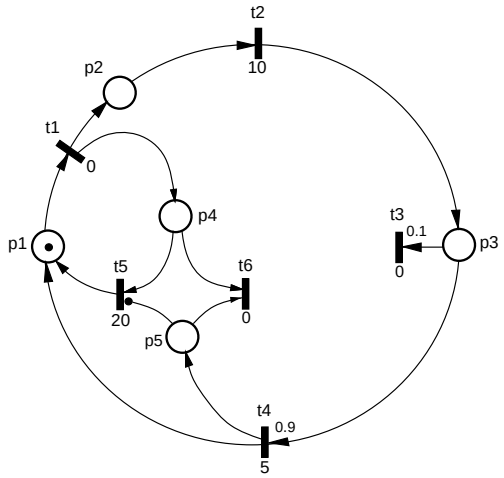

Fig.3.2. Petri net model of a simple protocol.

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 characterized 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.

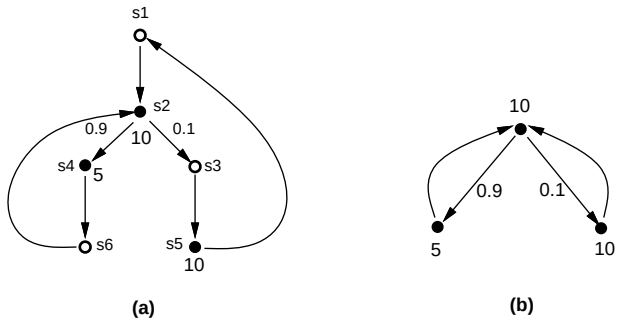

Fig.3.3. State graphs for the net shown in Fig.3.2; original graph (a) and reduced graph (b).

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* [83], 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.

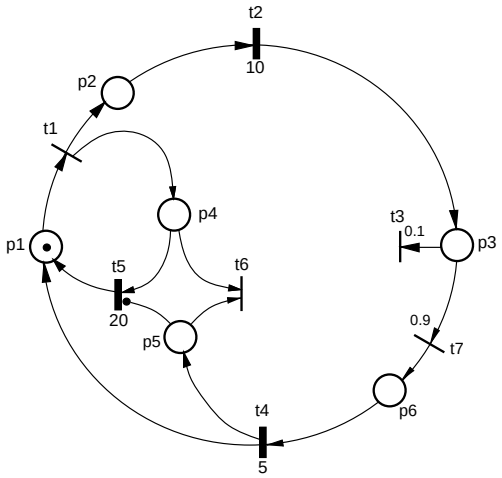

Fig.3.4. Enhanced Petri net model of a simple protocol.

Example. Fig.3.4 shows an enhanced 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, and for $f(t_1) = 2, f(t_2) = f(t_3) = 0.5, f(t_4) = 2.5$, 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 [11], [120]. Two more specialized transformations are shown in Fig.3.5. It should be noted that these two transformations preserve the state graphs of the original nets.

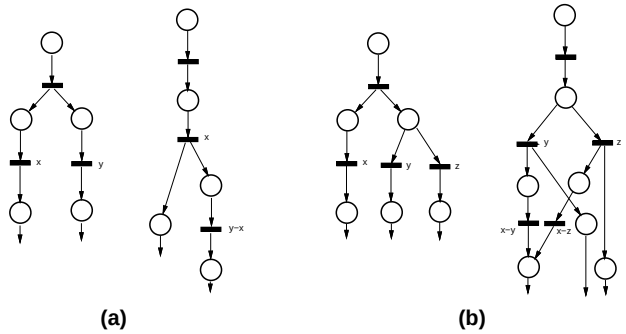

Fig.3.5. Simple net transformations; (a) $y > x$, (b) $x > y > z$.

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, and its state graph is shown in Fig.3.3(b). □

An application of D–timed nets to modeling ATM LAN's is described in [53], while [71] analyzes LeLann's distributed control protocol. An approach to analysis of unbounded timed nets is proposed in [81].

B. M–timed Petri Nets

In M–timed Petri nets (or Markovian timed nets) [73], [76], [78], [83], the occurrence times (or firing times) of transitions are exponentially distributed random variables with the average times described by the values $f(t), t \in T$.

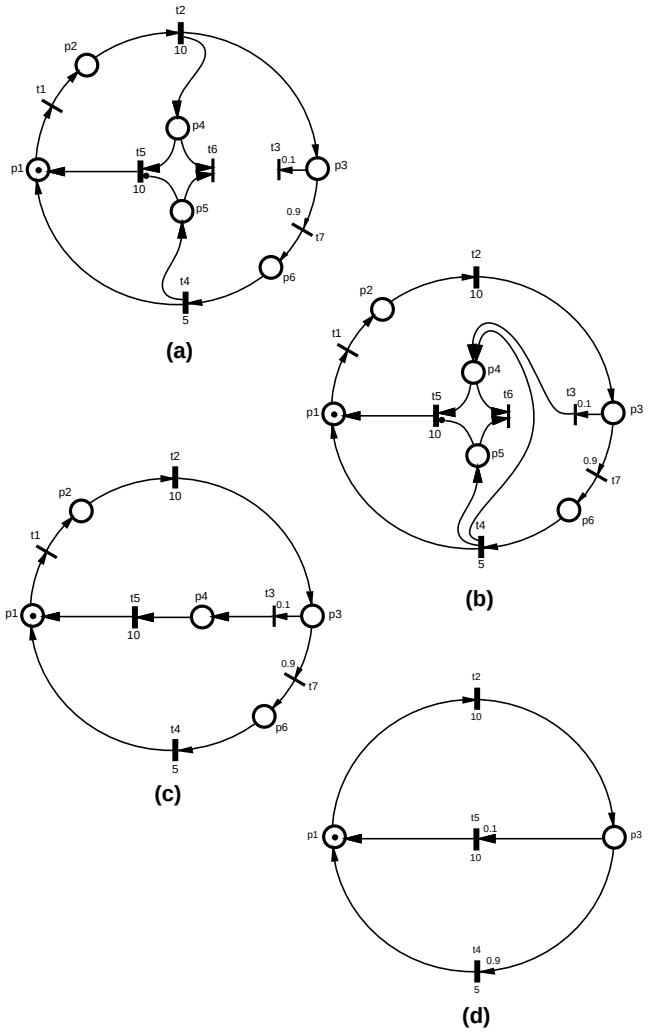

Fig.3.6. Transformations of the protocol model.

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

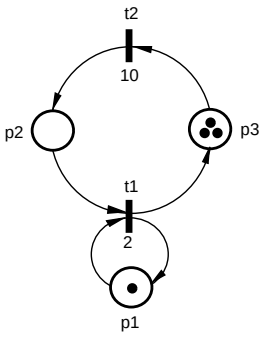

Fig.3.7. A simple model of an interactive system.

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 (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.

It should be observed that, in the net shown in Fig.3.7, the number of simultaneous firings of t_2 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.

More details about Petri net models of computer systems are given in [73], [83], [93], [95], [102]. □

A state of an M–timed net can be described by a pair of functions [83], $s = (m, n)$, where m is a marking function describing the distribution of tokens which are not involved in firings of transitions, $m : P \to \{0, 1, ...\}$, and n is the firing–rank function which, for each transition of the net, indicates the number of its current firings, $n : T \to \{0, 1, ...\}$.

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 ends 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 terminates 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, $\ell : E \to \mathbf{R}^+$.

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 [61], 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.

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}

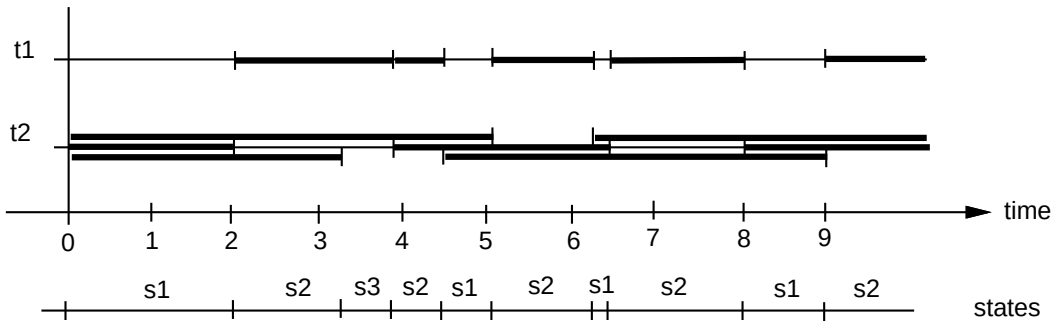

Fig.3.8. A sequence of possible events in the net shown in Fig.3.7.

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. □

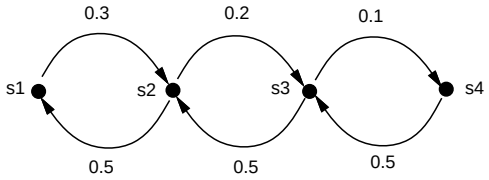

Fig.3.9. State graph for the net shown in Fig.3.7.

The exponentially distributed firing times of transitions can be combined into hypo- or 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 [83].

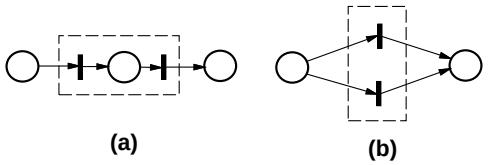

Fig.3.10. A model of a hypo-exponential (a) and hyper–exponential (b) server.

The models shown in Fig.3.10 can be used to refine the model of Fig.3.7 if other than exponential distribution is needed.

Fig.3.11 shown a different type of modification of the model shown in Fig.3.7. 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).

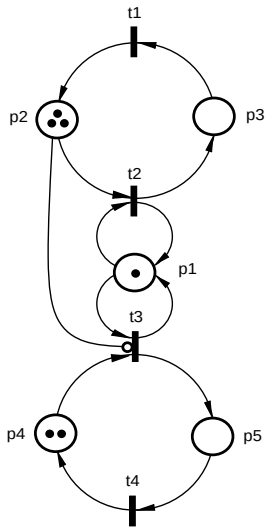

Fig.3.11. A system with two classes of jobs.

Detailed reachability analysis of the net shown in Fig.3.11 is given in [78]. Simple models of other computer systems are described in [74], [82], [83].

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 the execution of class B jobs 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 Fig.3.7 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).

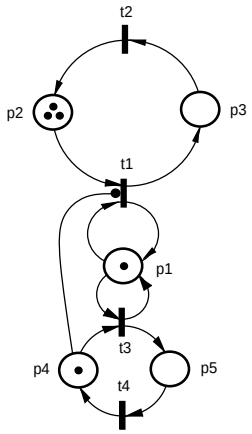

Fig.3.12. A system with unreliable processor.

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).

C. Timed Colored Petri Nets

Timed colored nets [82] are a straightforward combination of timed nets and colored 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 $\mathcal{N}$, $c : T \to V_C \to [0, 1]$, where V_C is the set of bindings for the set of colors C, and
- f is the firing–time function which assigns the (average) firing time (or the occurrence time) to each occurrence of each transition of $\mathcal{N}$, $f : T \to V_C \to \mathbf{R}^+$.

For the model of three dining philosophers (as shown in Fig.2.11, but restricted to colors A, B, C, a, b and c), assuming that all eating and thinking times are exponentially distributeds (with some rates), the set of reachable states contains 13 states shown in the following table (this model has two initial states, 1 and 2):

	m_i			n_i		
i	p_1	p_2	p_3	*eat*	*think*	j
1	1c		1C	1a	1b	3, 4
2	1a		1B	1c	1b	5, 6
3			1B	1c	1a+1b	7, 2, 9
4	1b+1c		1C	1a		8, 9
5			1C	1a	1b+1c	7, 10, 1
6	1a+1b		1B	1c		10, 11
7			1B+1C		1a+1b+1c	5, 12, 2
8	1b		1B	1c	1a	12, 6
9	1c		1A	1b	1a	3, 13
10	1b		1C	1a	1c	12, 4
11	1a		1A	1b	1c	5, 13
12			1A	1b	1a+1c	7, 11, 9
13	1a+1c		1A	1b		1,2

The graph of reachable states is a Markov chain representing the model's behavior, so its analysis is similar to analysis of ordinary nets.

IV. Case Studies

A. Manufacturing Systems

Modeling and analysis of manufacturing systems is one of the most popular applications of Petri nets [20], [21], [51], [69], however, the use of Petri net models for performance analysis of such systems has received little attention in the past; deadlock detection and deadlock prevention seem to be the dominant aspects of application of Petri nets to manufacturing systems.

A large class of manufacturing systems can be represented as collections of manufacturing cells connected by transportation (and possibly storage) systems. Simple examples of such systems are shown in Fig.4.1, where "A" is a supply of unprocessed parts, and "D" is a storage for the final products.

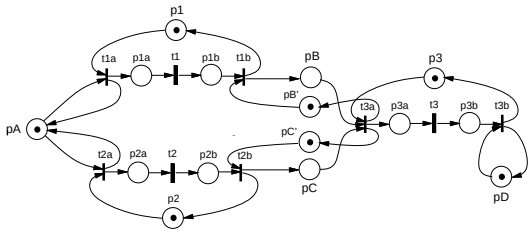

Fig.4.1. Outlines of simple manufacturing systems.

A Petri net model of the system from Fig.4.1(a) is shown in Fig.4.2. The three cells are represented by identical subnets, each of which contains one timed transition, t_i, which models the total operations performed by the corresponding cell; the two immediate transitions simply represent the operations of bringing a new part to a cell and removing a completed part from a cell. Places p_i, if marked, indicate that the corresponding cell is idle, and waiting for another part.

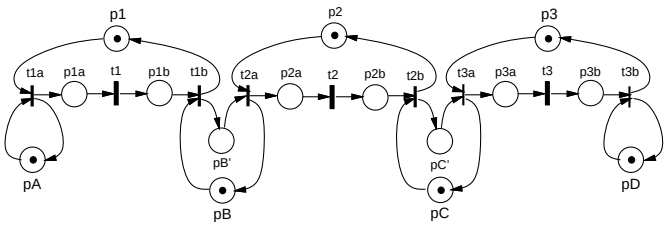

Fig.4.2. Petri net model of system outlined in Fig.4.1(a).

Place p_A represents the source of unprocessed parts, and it is assumed that there is always sufficient supply of these parts; this is the reason that, in Fig.4.2, whenever a part is taken from "A" (transition t_{1a}), a part is also "returned" to p_A, so there is another part "ready", when needed. Similarly, it is assumed that storage "D" can always accept another completed product. If these assumptions are not realistic, the model needs to be expanded to take these additional constraints into account.

The connections between the cells are represented by buffers "B" and "C" with capacity 1; this capacity is indicated by the initial markings of places p_B and p_C. If a different capacity of these buffers is needed, the initial marking of these two places needs to be changed accordingly.

Fig.4.3 shows a Petri net model corresponding to the outline from Fig.4.1(b). This model differs in two aspects from that in Fig.4.2; the storage "A" is connected to both stages "1" and "2" in Fig.4.3, and the connection with stage "3" is different because of a different flow of parts in the system.

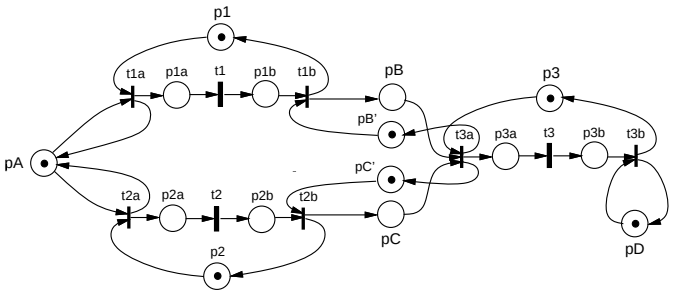

Fig.4.3. Petri net model of system outlined in Fig.4.1(b).

The models shown in Fig.4.2 and Fig.4.3 are composed of simple cyclic subnets, so the structural approach can be used for their analysis.

The net shown in Fig.4.2 has 5 simple P–invariants, which imply subnets with the following subsets of transitions (these subnets correspond to the cyclic subnets which can easily be identified in Fig.4.2); three subnets corresponding to the three cells, and two subnets corresponding to the two buffers:

inv	t_{1a}	t_1	t_{1b}	t_{2a}	t_2	t_{2b}	t_{3a}	t_3	t_{3b}
1	1	1	1	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	0	1	1	1	0	0	0
4	0	0	0	0	0	1	1	0	0
5	0	0	0	0	0	0	1	1	1

The cycle time is thus:

$$\tau_0 \;=\; \max(\tau_1, \tau_2, \tau_3, \tau_4, \tau_5)$$

where τ_i, $i = 1, .., 5$, are cycle times of the subnets:

$$\begin{aligned}
\tau_1 &= f(t_{1a}) + f(t_1) + f(t_{1b}),\\
\tau_2 &= f(t_{1b}) + f(t_{2a}),\\
\tau_3 &= f(t_{2a}) + f(t_2) + f(t_{2b}),\\
\tau_4 &= f(t_{2b}) + f(t_{3a}),\\
\tau_5 &= f(t_{3a}) + f(t_3) + f(t_{3b}).
\end{aligned}$$

The net shown in Fig.4.3 also has 5 simple P–invariants with the following sets of transitions of the subnets implied by these invariants:

inv	t_{1a}	t_1	t_{1b}	t_{2a}	t_2	t_{2b}	t_{3a}	t_3	t_{3b}
1	1	1	1	0	0	0	0	0	0
2	0	0	1	0	0	0	1	0	0
3	0	0	0	1	1	1	0	0	0
4	0	0	0	0	0	1	1	0	0
5	0	0	0	0	0	0	1	1	1

The difference with respect to the previous model (Fig.4.2) is only in subnet (2), for which the cycle time now is:

$$\tau_2 = f(t_{1b}) + f(t_{3a}).$$

The times of storing and retrieving parts $f(t_{1a})$, $f(t_{1b})$, etc., can be estimated on the basis of physical measurements; the times $f(t_1)$, $f(t_2)$ and $f(t_3)$ are usually derived from a more detailed analysis of the corresponding manufacturing cells.

Each manufacturing cell typically contains a number of versatile machines, M1, ...,Mk, an input and output conveyor, and a robot which moves the manufactured or assembled parts from one machine to another, and also from the input conveyor to the first machine and from the last machine to the output conveyor.

An outline of a simple manufacturing cell with 4 machines is shown in Fig.4.4.

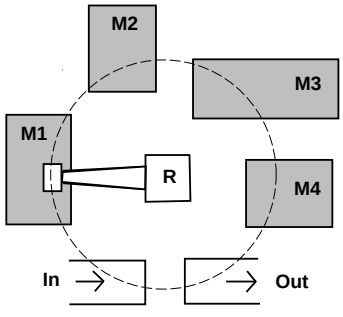

Fig.4.4. An outline of a 4–machine manufacturing cell.

A sequence of operations performed (cyclically) by the robot is called a *schedule*. It is known that there are $m!$ schedules for a cell with m machines [58]. The best schedule is the one which maximizes the throughput (or minimizes the cycle time) of a cell. For a given cell, all schedules can be systematically derived, as Petri net models, and evaluated using P–invariants [88].

Fig.4.5 shows a Petri net model of the simplest, sequential schedule for a 4–machine cell.

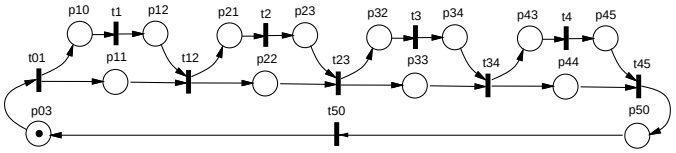

Fig.4.5. Net model of schedule 1234.

The model is composed of four sections modeling the machines of the cell, each section composed of a transition t_i, $i = 1, 2, 3, 4$, and two places, one representing the condition that the part has been loaded, so the machine can begin its operation, and the other indicating that the machine's operation has been completed, so the part can be moved by the robot to another machine or the output conveyor.

The sequence of robot's operations is described by the following sequence of transitions:

t_i	*robot's operations*
t_{01}	pick a part from In, move to M_1 and load
t_{12}	unload M_1, move to M_2 and load
t_{23}	unload M_2, move to M_3 and load
t_{34}	unload M_3, move to M_4 and load
t_{45}	unload M_3, move to Out and drop
t_{50}	move from Out to In

The model shown in Fig.4.5 contains several parallel paths which can be simplified without affecting the performance of the model [100], [120]; all places p_{ii}, $i = 1, 2, 3, 4$, with their arcs can be removed, creating the simple cyclic model shown in Fig.4.6.

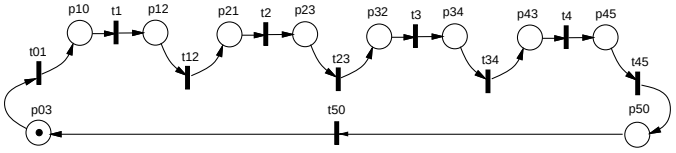

Fig.4.6. Simplified net model of schedule 1234.

The cycle time of the model shown in Fig.4.6 is simply:

$$\tau_0^{(1)} = f(t_{01}) + f(t_1) + f(t_{12}) + f(t_2) + f(t_{23}) + f(t_3) + f(t_{34}) + f(t_4) + f(t_{45}) + f(t_{50}).$$

This cycle time can easily be expressed in terms of elementary operations (and their durations) performed by the robot. Assuming that:

- u denotes the (average) pickup time,
- v – the (average) unload time,
- w – the (average) load time,
- x – the (average) drop time and
- y – the average 'travel' time between two adjacent machines (to simplify the description, it is assumed that this time is the same for all adjacent machines, and also the same for $M4$ to Out, Out to In and In to $M1$ moves),

the operations associated with transitions have the following (average) executions times:

t_i	$f(t_i)$
t_{01}	$u + w + y$
t_{12}	$v + w + y$
t_{23}	$v + w + y$
t_{34}	$v + w + y$
t_{45}	$v + x + y$
t_{50}	y

The cycle time, assuming that the (average) operation times of machines M1 to M4 are denoted by o_1 to o_4, is:

$$\tau_0^{(1)} = o_1 + o_2 + o_3 + o_4 + u + 4v + 4w + x + 6y.$$

A different schedule, with two concurrent activities, is shown in Fig.4.7; the initial marking of place p_{32} indicates that, when the next part is being picked from the input conveyor, the previous part is loaded on machine M3 and will be processed concurrently with the new part loaded on machine M1.

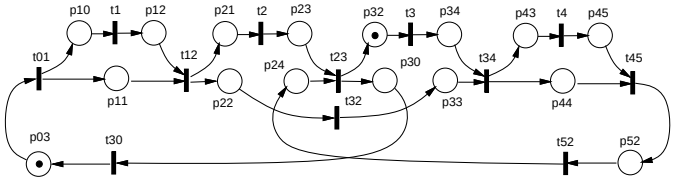

Fig.4.7. Net model of schedule 1243.

The sequence of robot's operations, with their execution times, is as follows:

t_i	$f(t_i)$
t_{01}	$u + w + y$
t_{12}	$v + w + y$
t_{32}	y
t_{34}	$v + w + y$
t_{45}	$v + x + y$
t_{52}	$3y$
t_{23}	$v + w + y$
t_{30}	$3y$

Similarly as before, the model can be simplified by removing places p_{11} and p_{44} and the arcs incident with them. The resulting net is shown in Fig.4.8.

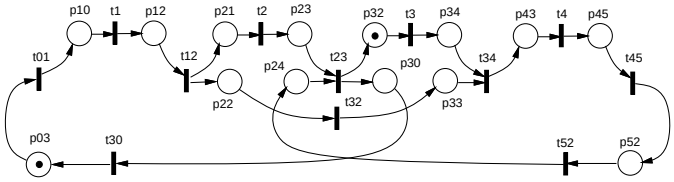

Fig.4.8. Simplified net model of schedule 1243.

The net in Fig.4.8 has 3 simple P–invariants with the following sets of transitions of subnets implied by these P–invariants:

inv.	*transitions*
1	$t_{01}, t_1, t_{12}, t_2, t_{23}, t_{30}$
2	$t_{23}, t_3, t_{34}, t_4, t_{45}, t_{52}$
3	$t_{01}, t_1, t_{12}, t_{32}, t_{34}, t_4, t_{45}, t_{52}, t_{23}, t_{30}$

so the cycle time is:

$$\tau_0^{(2)} = \max(\tau_1^{(2)}, \tau_2^{(2)}, \tau_3^{(2)})$$

where:

$$\tau_1^{(2)} = o_1 + o_2 + u + 2v + 3w + 6y,$$
$$\tau_2^{(2)} = o_3 + o_4 + 3v + 2w + x + 6y,$$
$$\tau_3^{(2)} = o_1 + o_4 + u + 3v + 3w + x + 12y.$$

The cycle time $\tau_0^{(2)}$ is usually smaller that $\tau_0^{(1)}$ but the result of comparison depends upon the specific values of parameters o_1 to o_4 and y.

The schedule with the maximum concurrency is shown of Fig.4.9; in this case the three machines, M2, M3 and M4, are loaded with (previous) parts when a new part is picked and loaded on machine M1.

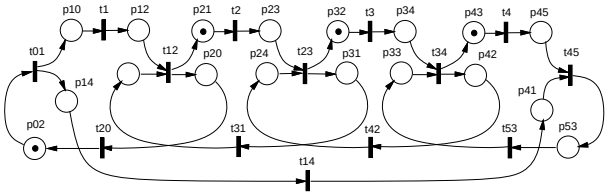

Fig.4.9. Net model of schedule 1432.

There is only one possible sequence of robot's operations for this model, and it is described by the following transitions (and their execution times):

t_i	$f(t_i)$
t_{01}	$u + w + y$
t_{14}	$3y$
t_{45}	$v + x + y$
t_{53}	$2y$
t_{34}	$v + w + y$
t_{42}	$2y$
t_{23}	$v + w + y$
t_{31}	$2y$
t_{12}	$v + w + y$
t_{20}	$2y$

The net shown in Fig.4.9 has 5 P–invariants which imply subnets with the following sets of transitions:

inv.	*transitions*
1	$t_{01}, t_1, t_{12}, t_{20}$
2	$t_{12}, t_2, t_{23}, t_{31}$
3	$t_{23}, t_3, t_{23}, t_{42}$
4	$t_{34}, t_4, t_{45}, t_{53}$
5	$t_{01}, t_{14}, t_{45}, t_{53}, t_{34}, t_{42}, t_{23}, t_{31}, t_{12}, t_{20}$

so the cycle time of this model is:

$$\tau_0^{(3)} = \max(\tau_1^{(3)}, \tau_2^{(3)}, \tau_3^{(3)}, \tau_4^{(3)}, \tau_5^{(3)})$$

where:

$$\tau_1^{(3)} = o_1 + u + v + 2w + 4y,$$
$$\tau_2^{(3)} = o_2 + 2v + 2w + 4y,$$
$$\tau_3^{(3)} = o_3 + 2v + 2w + 4y,$$
$$\tau_4^{(3)} = o_4 + 2v + w + x + 4y,$$
$$\tau_5^{(3)} = u + 4v + 4w + x + 16y.$$

The derived cycle times of manufacturing cells can be used in the model of manufacturing system, replacing the operations times of the cells; in particular, if the model shown in Fig.4.5 is representing cell "1" in Fig.4.1, the model shown in Fig.4.7 – cell "2" in Fig.4.1, and model shown in Fig.4.9 – cell "3", the cycle time for the manufacturing system is equal:

$$\tau_0 = \max(\tau_0^{(1)}, \tau_0^{(2)}, \tau_0^{(3)})$$

where the terms $\tau_0^{(i)}$, $i = 1, 2, 3$, are defined above.

The described approach first analyzes the performance of the manufacturing system at the abstract level of cells and storage elements, and then considers the cells one at a time. For complex manufacturing systems, even more structured approach can be used. Instead of dealing with all the cells at the same time, an additional level of subsystems can be introduced, and first the performance of the system can be expressed in terms of subsystems, then the performance of subsystems in terms of cells, and finally, performance of cells in terms of individual machines and their interconnections.

Such hierarchical approach can be used to model and analyse a large class of manufacturing systems. The approach is based on stepwise refinement of timed Petri net models, and structural analysis used for performance evaluation of derived models. The results are obtained in symbolic form, which provides very efficient analysis of specific configurations, described by sets of numerical parameters.

Hierarchical modeling by Petri nets is described in greater detail in [88], [89], [90], [96], [99], [114]. Modeling and analysis of manufacturing cells is presented in [84], [85], [103], [117], while performance analysis of other manufacturing systems is discussed in [98], [100], [101].

B. Multithreaded Multiprocessors

In modern computer systems, the performance of memory is increasingly often becoming the factor limiting the performance of the whole computer system. Due to continuous progress in manufacturing technologies, the performance of processors has been doubling every 18 months (the so–called Moore's law [28]). However, the performance of memory chips has been improving by only 10% per year [56], creating a "performance gap" in matching processor's performance with the required memory bandwidth. In effect, it is becoming more and more often the case that the performance of applications depends on the performance of machine's memory hierarchy [15].

Instruction–level multithreading, and in particular block–multithreading [14], [17], [45] tolerates long–latency memory accesses and synchronization delays by switching the threads rather than waiting for the completion of a long–latency operation which, in a distributed–memory system, can require thousands of processor cycles. Since the threads are executed in the same address space, switching from one thread to another (which is called "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.

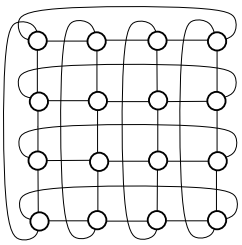

Fig.4.10. Outline of a 16–processor system.

The distributed memory system is composed of a number of processors connected by an interconnection network. Fig.4.10 outlines a 16–processor system with a two–dimensional torus–like interconnection network, used as a running example. Each node in Fig.4.10 is a multithreaded processor.

Fig.4.11 shows a model of a multithreaded processor (at the instruction execution level) 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 $TnetN$, $TnetW$, $TnetE$ and $TnetS$, which model – for this particular interconnection network – connections to four neighboring nodes, NORTH, WEST, EAST and SOUTH, respectively. The interconnecting network is characterized by fwo parameters, the average number of hops, n_h, between the source and a target of a message, and the delay of a single hop. The delays of messages forwarded in the interconnecting network are usually associated with the switches that control the transfer of messages from one node to another. It is assumed that these delays, denoted by t_s, are constant since they are rather insensitive to the length of messages, at least for reasonably short messages [68]. The delays introduced by the switches are represented by firing times assigned to $Tsout$ and $Tsinp$.

The processor shown in Fig.4.11 performs context switching for each long–latency memory access (local or remote); Petri net models of some other variants of multithreading are discussed in [93], [107], [111].

The execution of each instruction of the 'running' thread is modeled by transition $Trun$, a timed transition with the firing time representing one processor cycle. Place $Proc$ represents the (available) processor (if marked) and place $Ready$ – the queue of threads waiting for execution. The initial marking of $Ready$ represents the (average) number of available threads, n_t.

If the processor is available (i.e., $Proc$ is marked) and $Ready$ is not empty, a thread is selected for execution by firing the immediate transition $Tsel$. Execution of consecutive instructions of the selected thread is performed in the loop $Pnxt$, $Trun$, $Pend$ and $Tnxt$. $Pend$ is a free–choice place with the choice probabilities determined by the runlength, ℓ_t, of the thread. In general, the free–choice probability assigned to $Tnxt$ is equal to $(\ell_t - 1)/\ell_t$, so if ℓ_t is equal to 10, the probability of $Tnxt$ is 0.9; if ℓ_t is equal to 5, this probability is 0.8, and so on. The free–choice probability of $Tend$ is just $1/\ell_t$.

If $Tend$ is chosen for firing rather than $Tnxt$, the execution of the thread ends, a request for a long–latency access to (local or remote) memory is placed in Mem, and a token is also deposited in $Pcsw$. The timed transition $Tcsw$ represents the context switching and is associated with the time required for the switching to a new thread, t_{cs}. When its firing is finished, another thread is selected for execution (if it is available).

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 $TnetN$, ..., $TnetS$). 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.

Colors are used to fold the processors into a single model shown in Fig.4.11. Since transitions $TnetN$, ..., $TnetS$ 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 [116].

It is convenient to assume that all timing characteristics are expressed in processor cycles (which is assumed to be 1 unit of time). The basic model parameters and their typical values are as follows:

symbol	*parameter*	*values*
n_t	number of available threads	2,...,20
ℓ_t	thread runlength	5,10,20
t_{cs}	context switching time	1,2,5
t_m	memory cycle time	10,20
t_s	switch delay	5,10
p_ℓ, p_r	probability of accesses to local/remote memory	0.1, ..., 0.9

All timed transitions in Fig.4.11 have deterministic fir-

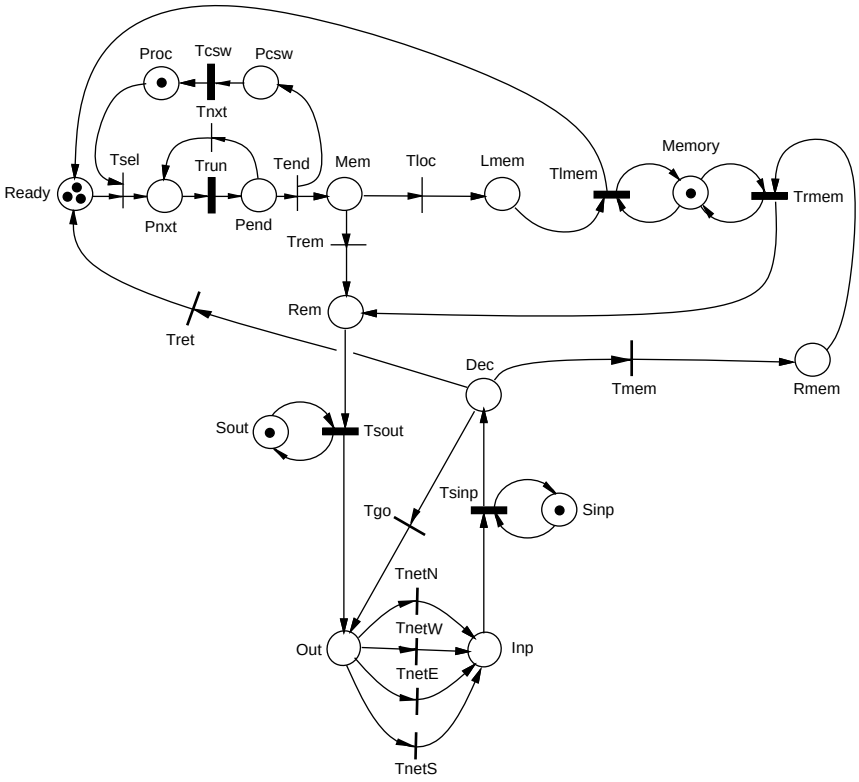

Fig.4.11. A colored net model of a multithreaded multiprocessor.

ing 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 processors (e.g., 16), this space is very large. Therefore, event–driven simulation was used to obtain performance characteristics of this model [87], [94]. An example of such characteristics, presented in Fig.4.12, shows the utilization of each processor as a function of the number of available threads, nt (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.

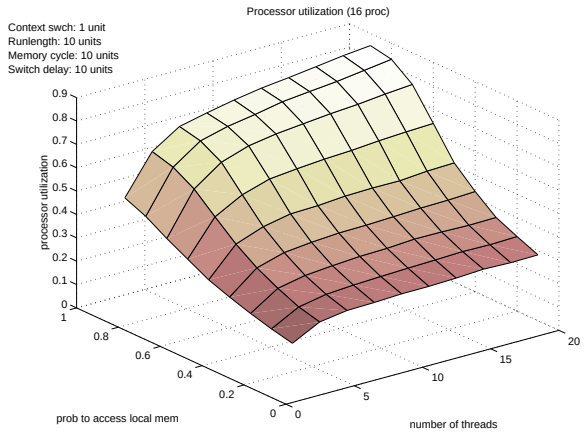

Fig.4.12. Processor utilization in a 16–processor system; $t_{cs} = 1, \ell_t = t_m = t_s = 10$.

Fig.4.12 shows 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 processors "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 $p_\ell < 0.75$ [106], [**?**], [115]. Indeed, Fig.4.13 shows the utilization of the input switch (for the same values of modeling parameters as in Fig.4.12); it should be noted that, for presentation reasons, Fig.4.13 uses the probability of accessing remote memory, p_r, rather than p_ℓ used in Fig.4.12, so the "front part" of Fig.4.13 corresponds to the "back part" of Fig.4.12.

Fig.4.13 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 decreases). The "boundary" corresponding to $p_r = 0.25$ is clearly visible in Fig.4.13. The input switch is simply "too slow" if the probability of accesses to remote memory can be greater than 0.25.

Fig.4.14 shows the utilization of the processor for the case when the switch delay is one half of that used in Fig.4.12, i.e., $t_s = 5$; the processor's utilization is signif-

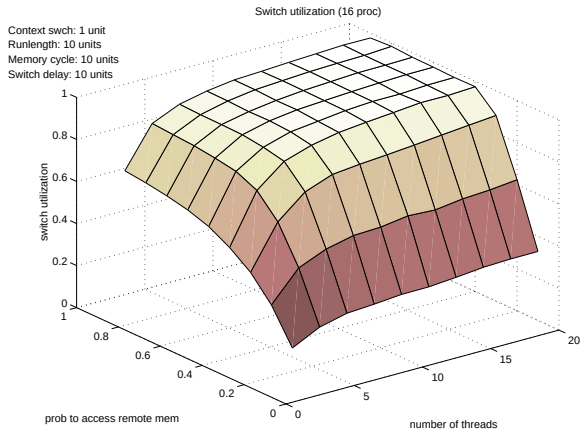

Fig.4.13. Input switch utilization in a 16–processor system; $t_{cs} = 1, \ell_t = t_m = t_s = 10$.

icantly better than in Fig.4.12, but the limiting effects of the input switch can still be observed for small values of p_ℓ.

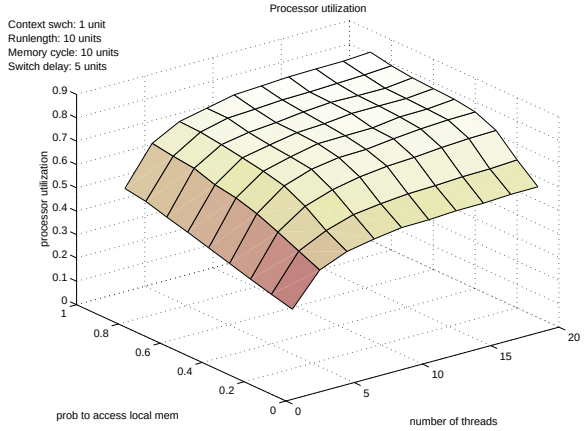

Fig.4.14. Processor utilization in a 16–processor system; $t_{cs} = 1, \ell_t = t_m = 10, t_s = 5$.

For performance analysis of derived models, the interconnection network is characterized by the average number of hops, n_h, and the delay of switches, t_s. Consequently, different networks characterized by the same value of n_h (and the same delay t_s) will yield the same performance characteristics of the nodes. For example, Fig.4.15 shows a hypercube network for a 16–processor system that can be used instead of a 2–dimensional torus–like network shown in Fig.4.10. Since each node in Fig.4.15 is connected to 4 neighbors (as is the case in Fig.4.10), 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 also 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

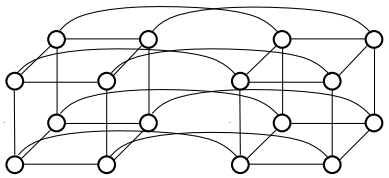

Fig.4.15. Outline of a 16–processor system.

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.

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 is given in [92], [93], [97], [104], [108], [109], [110], [113], [116].

C. Performance–Equivalent Multiprocessors

Since 1990s there has been an increasing trend to use networks of workstations [8] as the high–performance platform instead of expensive and specialized parallel supercomputers [16], [50]. Among the driving forces that have enabled such a transition has been the improvement and availability of commodity high–performance workstation and networks. These technologies are making networks of computers (PCs and workstations) an appealing vehicle for parallel processing and low–cost "commodity supercomputing" [9].

Multiprocessor systems are usually classified as shared–memory or message–passing systems [31]. Shared–memory systems are believed to be easier to program, but distributed–memory systems scale in a better way; systems with large numbers of processors are usually distributed–memory systems [66].

Multiprocessor systems studied in this project are distributed–memory systems, which execute transaction–processing–like jobs. These jobs are composed of (possibly cycles of) a processing phase followed by a communication phase (to access information in a distributed database or to exchange information with other jobs). For simplicity, it is assumed that the information is distributed uniformly over the nodes of the system, so all nodes are accessed with the same probability. It is also assumed that the job processing times are exponentially distributed, so they can be characterized by a single parameter, the average processing time (this assumption is not very important and is made primarily to simplify job descriptions). Similarly, it is assumed that the execution times of remote operations requested by a job are also exponentially distributed, with another parameter describing their average durations. It appears that the specific values of the average processing time and the average operation time are not as important as their sum which specifies the "service demand" for processors.

An outline of a single processor, with its local memory,

and two network interfaces, is shown in Fig.4.16. The outbound interface is used for outgoing traffic, i.e., requests to remote nodes originating at this node as well as results of remote operations requested from this node; the inbound interface handles incoming traffic, i.e., results of remote operations that 'return' to this node and remote requests of operations performed at this node.

Fig.4.16 also shows two queues of jobs, the principal queue of jobs waiting for execution, Ready Queue, and a secondary queue of remote requests, Remote Queue; whenever the currently executing job requests an operation on another node, the current job becomes suspended, and a job from one of these queues is randomly selected for execution; this execution continues until the end of the operation or until another remote operation is initiated. When a remote operation is completed and its results are returned to the 'home' node, the status of the suspended job changes to 'ready', and the job joins the queue of ready jobs, waiting for another execution phase on the processor.

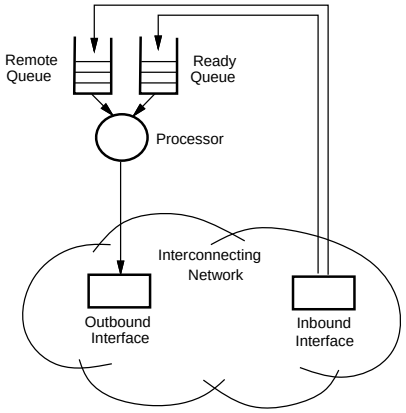

Fig.4.16. Outline of a single processor.

The average execution time of a job, until it makes a request for a remote operation, is one of the basic modeling parameters denoted by t_e; the average time of performing a remote operation is denoted by t_r. The (average) number of jobs available at each node, n_j, is another modeling parameter. For very small values of n_j, queueing effects can be practically neglected, so the performance can be predicted by taking into account only the delays of system's components. On the other hand, for large values of n_j, the system can be considered in saturation, which means that one of its components will be utilized in almost 100%, limiting the utilization of other components as well as the whole system. Identification of such limiting components (called the bottlenecks [32]) and improving their performance is the key to the improved performance of the entire system.

A timed Petri net model of a single node is shown in Fig.4.17.

The execution of the 'running' job is modeled by $Trun$; its execution time is exponentially distributed with the average value t_e. The execution of an operation requested by another node is represented by $Trem$; its execution time is also exponentially distributed, with the average value t_r.

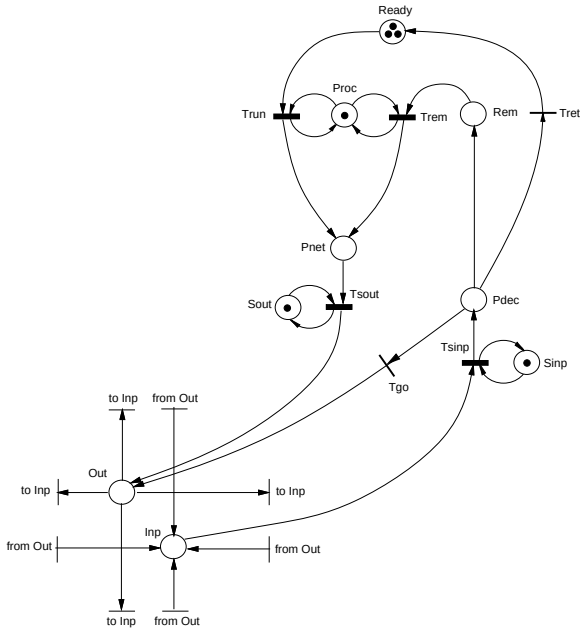

Fig.4.17. Job–level model of a node.

At the end of job execution (by $Trun$), a remote operation is requested by depositing a token in $Pnet$, and the current job becomes suspended; at the end of remote operation (by $Trem$), the token deposited in $Pnet$ represents the results of the operation which are returned to the 'home' node. If there is another job waiting for execution (in $Ready$), or a request from another node (in Rem), the execution is initiated concurrently with forwarding the token to the outgoing interface; after a sequential delay (the outbound switch $Sout$) the request is forwarded to Out, where a random selection is made of one of the four (in this case) adjacent nodes (all nodes are selected with equal probabilities). Similarly, the incoming traffic is collected from all neighboring nodes in Inp, and, after a sequential delay (the inbound switch $Sinp$), forwarded to $Pdec$. $Pdec$ is a decision point with three possibilities: the *Ready Queue*, which represents the results of remote operations reaching their 'home' nodes; Out, which represents requests as well as responses forwarded to another node (another 'hop' in the interconnecting network); and *Remote Queue*, which represents remote operations reaching their destination node; these remote operations are executed when the processor becomes available. The choice probabilities associated with $Pdec$ characterize the interconnecting network.

As in the case of multithreaded multiprocessors, the interconnecting network is characterized by two parameters, the average number of hops between the source and a target of a message, n_h, and the delay of a single hop, t_s.

The utilizations of components in complex systems is directly related to service demands for these components. The service demand, d_i, for the component i is the product of the rate of requests (sometimes also called the 'visit rate'), v_i, and the (average) service time of this component, s_i, i.e., $d_i = v_i * s_i$.

For the multiprocessor systems, the components and

their service demands are:
- processors with job service demands $d_{e,j}$, $j = 1, ..., n_p$;
- processors with remote service demands $d_{r,j}$, $j = 1, ..., n_p$;
- inbound network switches with service demands $d_{i,j}$, $j = 1, ..., n_p$;
- outbound network switches with service demands $d_{o,j}$, $j = 1, ..., n_p$.

If all processors are the same, the steady–state service demands at all nodes are identical, and the second subscripts can be dropped.

For a single cycle of job processing (i.e., a job going through the phases of execution, suspension, and then waiting for another execution), the job service demand for the processor is simply the average execution time t_e. The analyzed cycle of job processing contains one request for a remote operation (to be executed on one of remote nodes); this is equivalent to an execution of a single request from any one of the remaining $(n_p - 1)$ nodes with the probability $1/(n_p - 1)$ (due to the uniform distribution of requests over the nodes of the system). The total service demand for processor is thus $t_e + t_r$.

The service demand for the inbound switch (in each processor) is equal to $2 * n_h * t_s$; the factor 2 is due to sending requests and then returning the results of remote operations. The service demand for the outbound switch is just $2 * t_s$.

The service demands are thus:

$$\begin{aligned} d_p &= t_e + t_r; \\ d_i &= 2 * n_h * t_s; \\ d_o &= 2 * t_s. \end{aligned}$$

Two (multiprocessor) systems are equivalent with respect to performance (or performance equivalent) if the maximum service demands are the same and are associated with corresponding components of the systems. A straightforward consequence of this definition is that component utilizations in performance–equivalent systems are the same; this is the essence of the concept of performance equivalence.

Performance–equivalent systems can be used to simplify the simulation–based performance analysis of distributed–memory multiprocessor systems (as well as other systems which have a similar structure). More specifically, since the simulation time required for the analysis of multiprocessor systems depends upon the number of processors, instead of simulating a system containing n_p processors, a much simpler performance–equivalent system can be used, significantly reducing the required simulation time, and providing reasonably accurate results. For performance analysis of the 16–processor system (Fig.4.10), a 4–processor system can be used with the same parameters t_e and t_r, and with the switch delay $t_s^{(4)}$ adjusted to the value which compensates for the difference in the values of n_h between the 16–processor, $n_h^{(16)}$, and 4–processor, $n_h^{(4)}$, systems, i.e., such that:

$$n_h^{(16)} * t_s^{(16)} = n_h^{(4)} * t_s^{(4)}.$$

Since $n_h^{(16)} = 2$ and $n_h^{(4)} = 4/3$ [111], performance–equivalence is obtained im this case for $t_s^{(4)} = 1.5 * t_s^{(16)}$.

Since the average number of hops in a 16–processor hypercube interconnecting network is the same as in 16–processor two–dimensional torus–like network, the performance characteristics of both networks are the same (although these two interconnecting networks scale in different ways).

Similarly, for a 64–processor system with a hypercube interconnecting network, the average number of hops is equal to 3 [111]; such a system is performance equivalent to a 16–processor system with a two–dimensional torus–like interconnecting network in which the switch delays are 1.5 greater than those in the 64–processor system, and also to a 4–processor system in which the switch delays are 2.25 greater then those in the 64–processor system. A 64–processor system with the hypercube interconnection network is performance–equivalent to a 64–processor system with a two–dimensional torus-like network if the switch delays of the latter network are 0.75 of the switch delays of the hypercube system; on the other hand, the 64–processor hypercube system is performance–equivalent to a 64–processor system with a three-dimensional torus–like interconnecting network with the same switch delays (since the value of n_h for such a network is also equal to 3), and so on.

All performance results presented here as an illustration of performance equivalence have been obtained by simulating the behavior of Petri net models of distributed–memory multiprocessor systems. All temporal characteristics of the models are expressed in "abstract time units" (with no reference to real units of time) which does not affect the results.

Fig.4.18 shows the utilization of processors in a 16–processor system as a function of the number of available jobs, n_j, (i.e., the initial marking of *Ready* in Fig.4.17).

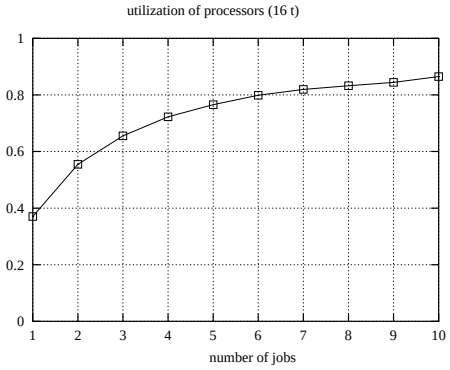

Fig.4.18. Processor utilization – 16 processors (2D torus); $t_e = 10$, $t_r = 2$, $t_s = 2$.

For small number of jobs, the utilization increases quite significantly with the number of jobs; it increases by about 50% by introducing the second available job, and it practically doubles when the number of available jobs increases to 5. Further increases of the processor utilization, for more than 5 or 6 jobs, are, however, rather insignificant.

The utilization of the processors in a performance–equivalent 4–processor system is shown in Fig.4.19; performance equivalence for the 4–processor system is obtained by using the same values of parameters t_e and t_r, and increasing t_s to 3 to compensate for the decreased value of $n_h^{(4)}$.

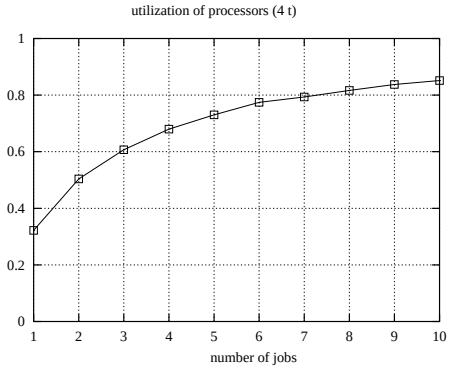

Fig.4.19. Processor utilization – 4 processors (2D torus); $t_e = 10$, $t_r = 2$, $t_s = 3$.

Fig.4.20 shows the utilization of processors in a 32-processor system with a hypercube interconnecting network which is performance-equivalent to a 16–processor system connected by a two–dimensional torus–like network (as in Fig.4.10); again, parameters t_e and t_r are the same as in Fig.4.18, while t_s is reduced to compensate for the increased value of n_h.

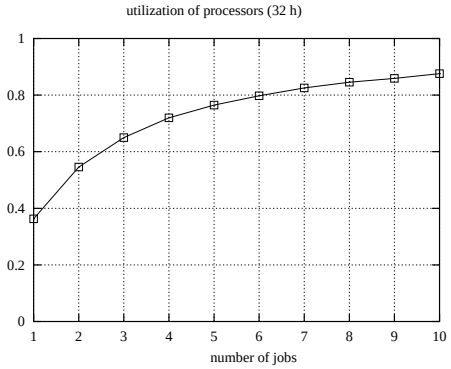

Fig.4.20. Processor utilization – 32 processors (hypercube); $t_e = 10$, $t_r = 2$, $t_s = 1.333$.

The results shown in Fig.4.18, Fig.4.19 and Fig.4.20 are practically the same.

Similarly, it can be shown [111] that the utilization of processors does not depend upon specific values of t_e and t_r, but upon their sum (which determines the service demand for the processor), and that the utilizations of components do not depend upon the specific values of parameters but on the ratio of the service demand for processors to the service demand for the switches (this is the reason that abstract time units can be used in specifying temporal properties of the model). All temporal properties of the analyzed systems can thus be characterized by just one parameter, the computation–to–communication ratio, $r_{comp/comm}$, which, in this case is equal to $(t_e + t_r)/(2 * n_h * t_s)$; this ratio remains the same for cases illustrated in Fig.4.18, Fig.4.19, and Fig.4.20.

The utilization of processors as a function of the number of available jobs, n_j, and the computation–to–communication ratio, $r_{comp/comm}$, is shown in Fig.4.21.

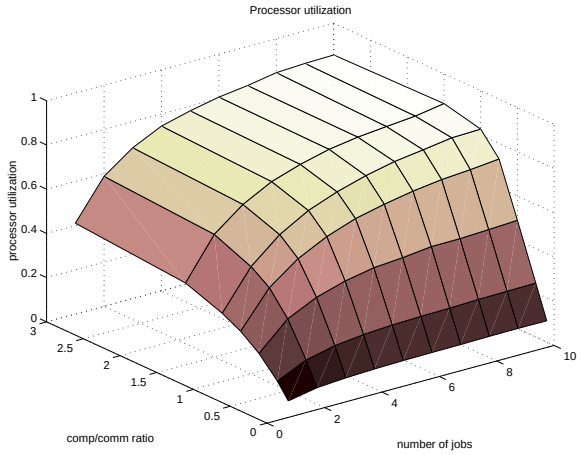

Fig.4.21. Processor utilization – 16 processors (2D torus).

The utilization increases almost linearly with the values of $r_{comp/comm}$ when $r_{comp/comm} < 1$, i.e., when the communication is the system bottleneck; the utilization is practically independent of $r_{comp/comm}$ when $r_{comp/comm} > 1$, i.e., when the computation (i.e., the processors) becomes the system bottleneck. It should be observed that Fig.4.18 to Fig.4.21 illustrate the crossection of Fig.4.21 at $r_{comp/comm} = 1.5$.

The presented performance results for distributed–memory multiprocessor systems indicate that significant simulation–time reductions can be achieved by using simpler models which are equivalent with respect to performance to the original systems. Since the simulation time of complex models usually increases superlinearly with the size of the model, the gains in the simulation time also increase more than linearly with the size of the (original) model.

A slightly different approach to performance equivalence is presented in [105] where instead of changing the delays of switches, the net model is modified in such a way that the value of n_h is preserved at the level of the original system, independently of the number of processors, so, again, much simpler model can be simulated to reduce the simulation time.

Results of other studies of the performance of distributed systems are presented in [52], [72], [118], [119].

V. Concluding Remarks

Because of the complexity of real–life net models, high-level Petri nets are becoming increasingly popular in practical applications of Petri nets [25], [57], [67]. Compositionality of models, usually expressed by process algebras, often with temporal enhancements for performance analysis, is

expected to provide elegant formal methods for complex realistic applications [29], [38], [60].

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 "Lecture Notes in Computer Science" (vol. 2679, 2360, 2075, 1825, 1639, 1420, and so on). Springer-Verlag, also in the series "Lecture Notes in Computer Science", has been publishing "Advances in Petri Nets", initially an annual collection of selected contributions to the area of Petri nets, and recently, collections of contributions is specialized areas, such as communication systems, workflow modeling, etc. 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 popular Peterson's book [48] and the Reisig's monograph [54] are now supplemented by several books on application of Petri nets to manufacturing systems [20], [21], [51], [69], to workflow management systems [26], on modeling using stochastic Petri nets [6], [10], [27], [40], on colored Petri nets and their applications [36], [37], [55], and also on properties of some classes of Petri nets [19], [65].

References

[1] Aalst van der, W.M.P., "Interval timed colored Petri nets"; in: **Advances in Petri nets 1993** (Lecture Notes in Computer Science 674), pp.126–147, Springer-Verlag 1993.

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

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] Anderson, T., O. Culler, and D. Patterson, "A case for NOW (Network of Workstations)", IEEE Micro , vol.15, pp.54-64, 1995.

[9] Baker, M., and R. Buyya, "Cluster computing: the commodity supercomputer"; Software Practice and Experience, vol.19, pp.551-576, 1999.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] Burger, D., Goodman, J.R., Kaegi, A., "Memory bandwidth limitations of future microprocessors"; *Proc. 23-rd Annual Int. Symp. on Computer Architecture*, Philadelphia, PA, pp.78-89, 1996.

[16] Buyya, R., **High performance cluster computing: systems and architectures**, Prentice-Hall 1999.

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

[18] 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.

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

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

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] Girault, C., Valk, R., **Petri Nets for Systems Engineering**; Springer-Verlag 2002.

[27] Haas, P.J., **Stochastic Petri Nets**; Springer-Verlag 2002.

[28] Hamilton, S., "Taking Moore's law into the next century"; *IEEE Computer Magazine*, vol.32, no.1, pp.43-48, 1999.

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

[30] 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.

[31] Hwang, K., **Advanced computer architecture – parallelism, scalability, programmability**, McGraw-Hill 1993.
[32] Jain, R., **The art of computer systems performance analysis**, J. Wiley & Sons 1991.
[33] Janicki, R., Koutny, M., "Semantics of inhibitor nets"; Information and Computation, vol.123, no.1, pp.1–16, 1995.
[34] Janicki, R., Koutny, M., "On causality semantics of nets with priorities"; Fundamenta Informaticae, vol.38, no.3, pp.223–255, 1999.
[35] Jensen, K., "Coloured Petri nets"; in: **Advanced Course on Petri Nets 1986** (Lecture Notes in Computer Science 254), pp.248–299, Springer-Verlag 1987.
[36] Jensen, K., **Colored Petri Nets – Basic Concepts, Analysis Methods and Practical Use**, vol.1; Springer-Verlag 1992.
[37] Jensen, K., **Colored Petri Nets – Basic Concepts, Analysis Methods and Practical Use**, vol.2; Springer-Verlag 1995.
[38] 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.
[39] 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.
[40] Lindemann, C., **Performance Modeling with Deterministic and Stochastic Petri Nets**; Wiley and Sons 1998.
[41] 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.
[42] 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.
[43] 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.
[44] M.K. Molloy, "Performance analysis using stochastic Petri nets"; IEEE Trans. on Computers, vol.31, no.9, pp.913–917, 1982.
[45] Moore, S.W., **Multithreaded Processor Design**; Kluwer Academic Publishers 1996.
[46] Murata, T., "Circuit theoretic analysis and synthesis of marked graphs"; IEEE Trans. on Circuits and Systems, vol.24, no.7, pp.400–405, 1977.
[47] Murata, T., "Petri nets: properties, analysis and applications"; Proceedings of IEEE, vol.77, no.4, pp.541–580, 1989.
[48] Peterson, J.L., **Petri Net Theory and the Modeling of Systems**; Prentice–Hall 1981.
[49] Petri, C.A., "Kommunikation mit Automaten"; Ph.D. Dissertation, University of Bonn, Bonn, Germany 1962; also: Memorandum MAC-M-212, Project MAC, Massachusetts Institute of Technology, Cambridge, MA.
[50] Phister, G., **In search of clusters**, Prentice-Hall 1998.
[51] Proth, J.M., Xie, X., **Petri Nets**; Wiley & Sons 1996.
[52] Rada, I., Zuberek, W.M., "Distributed generation of state space for timed Petri nets"; High Performance Computing Symposium (HPC'01), Seattle, WA, pp.219–227, 2001.
[53] Reid, M., Zuberek, W.M., "Timed Petri net models of ATM LANs"; in: **Applications of Petri Nets in Telecommunication Systems – Advances in Petri Nets** (Lecture Notes in Computer Science 1605), J. Billington, M. Diaz, G. Rozenberg (eds.), pp.150–175, Springer-Verlag 1999.
[54] Reisig, W., **Petri Nets - an Introduction** (EATCS Monographs on Theoretical Computer Science 4); Springer-Verlag 1985.
[55] Reisig, W., **Elements of Distributed Algorithms – Modeling and Analysis with Petri Nets**; Springer-Verlag 1999.
[56] Rixner, S., Dally, W.J., Kapasi, U.J., Mattson, P. and Ovens, J.D., "Memory access scheduling"; *Proc. 27-th Annual Int. Symp. on Computer Architecture*, Vancouver, Canada, pp.128-138, 2000.
[57] 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.
[58] Sethi, S.P., Sriskandarajah, C., Sorger, G., Blazewicz, J., Kubiak, W., "Sequencing of parts and robot moves in a robotic cell"; Int. Journal of Flexible Manufacturing Systems, vol.4, pp.331–358, 1992.
[59] 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.
[60] 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.
[61] Stewart, W.J., **Introduction to the Numerical Solution of Markov Chains**; Princeton University Press 1994.
[62] Tamura, H., Abe, T., Shinoda, S., "Obtaining a marked graph from a synchronic distance matrix"; Electronics and Communications in Japan; Part III – Fundamental Electronic Science; vol.79, no.3, pp.53–63, 1996.
[63] Teruel, E., Silva, M., "Structure theory of equal conflict systems"; Theoretical Computer Science, vol.153, no.1-2, pp.271–300, 1996.
[64] Valk, R., "Test on zero in Petri nets"; in: **Applications and Theory of Petri Nets** (Informatik–Fachberichte 52), pp.193–197, Springer-Verlag 1982.
[65] Wang, J., **Timed Petri nets**; Kluwer Academic Publ. 1998.
[66] Wilkinson, B., **Computer architecture – design and performance**, Prentice Hall 1996.
[67] 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.
[68] Xu, Z., K. Hwang, "Modeling communication overhead: MPI and MPL performance on the IBM SP2", IEEE Parallel and Distributed Technology, Vol.29. pp.9-23, 1996.
[69] Zhou, M-C., **Petri Nets in Flexible and Agile Automation**; Kluwer Academic Publishers 1995.
[70] Zuberek, W.M., "Timed Petri nets and preliminary performance evaluation"; Proc. 7-th Annual Symposium on Computer Architecture, pp.89–96, 1980.
[71] Zuberek, W.M., "Analysis of Le Lann's distributed control protocol by Petri nets"; Proc. 2–nd European Workshop on Theory and Applications of Petri Nets (EWTAPN'81), Bad Honnef, Germany, pp.555–568, 1981.
[72] Zuberek, W.M., "Application of timed Petri nets to analysis of multiprocessor realizations of digital filters"; Proc. 25-th Midwest Symp. on Circuits and Systems, Houghton, MI, pp.134–139, 1982.
[73] Zuberek, W.M., "Augmented M–timed Petri nets, modeling and performance evaluation of computer systems"; Transactions of the Society for Computer Simulation, vol.2, no.2, pp.135–153, 1985.
[74] Zuberek, W.M., "Generalized M–timed Petri nets and performance evaluation of computer systems"; INFOR Journal – Special Issue on Computer Systems Performance Evaluation, vol.23, no.3, pp.344–362, 1985.
[75] Zuberek, W.M., "Inhibitor D–timed Petri nets and performance analysis of communication protocols"; INFOR Journal – Special Issue on Communication System Performance, vol.24, no.3, pp.231–249, 1986.
[76] Zuberek, W.M., "M–timed Petri nets and Markov chains in modelling of computer systems"; Proc. 14-th ACM Computer Science Conf. (CSC'86), Cincinnati, OH, pp.101–106, 1986.
[77] Zuberek, W.M., "Modified D–timed Petri nets, timeouts, and modelling of communication protocols"; Proc. 6-th Int. Conf. on Distributed Computer Systems, Cambridge, MA, pp.452–457, 1986.
[78] Zuberek, W.M., "M–timed Petri nets, priorities, preemptions, and performance evaluation of systems"; in: "Advances in Petri Nets 1985" (Lecture Notes in Computer Science 222), pp.478–498, Springer–Verlag 1986.
[79] Zuberek, W.M., "D–timed Petri nets and modelling of timeouts and protocols"; Transactions of the Society for Computer Simulation, vol.4, no.4, pp.331–357, 1987.
[80] Zuberek, W.M., "Preemptive D–timed Petri nets, timeouts, modelling and analysis of communication protocols"; Proc. IEEE Sixth Annual Conf. on Global Networks (INFOCOM'87), San Francisco, CA, pp.721–730, 1987.
[81] Zuberek, W.M., "Performance evaluation using unbounded timed Petri nets"; Proc. 3-rd Int. Workshop on Petri Nets and Performance Models (PNPM'89), Kyoto, Japan, pp.180–186, 1989.
[82] Zuberek, W.M., "Performance evaluation using timed colored Petri nets"; Proc. 33-rd Midwest Symp. on Circuit and Systems (Special Session on Petri Net Models), Calgary, Alberta, pp.779–782, 1990.
[83] 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.

[84] Zuberek, W.M., "Schedules of flexible manufacturing cells and their timed colored Petri net models"; Proc. IEEE Int. Conf. on Systems, Man and Cybernetics (SMC'95), Vancouver, BC, pp.2142–2147, 1995.

[85] Zuberek, W.M., "Composite schedules of manufacturing cells and their timed Petri net models"; Proc. IEEE Int. Conf. on Systems, Man and Cybernetics (SMC'96), Beijing, China, pp.2990–2995, 1996.

[86] 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`).

[87] Zuberek, W.M., "Modeling using timed Petri nets – event-driven 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`).

[88] Zuberek, W.M., "Hierarchical derivation of schedules for manufacturing cells"; Proc. 9-th Symp. on Information Control in Manufacturing (INCOM-98), Nancy-Metz, France, pp.423–428, 1998.

[89] Zuberek, W.M., "Stepwise refinements of net models and their place invariants"; Proc. 8-th Int. Workshop on Petri Nets and Performance Models (PNPM'99), Zaragoza, Spain, pp.92–101, 1999.

[90] Zuberek, W.M., "Hierarchical derivation of Petri net models of composite schedules for manufacturing cells"; Proc. IEEE Int. Conf. on Systems, Man and Cybernetics (SMC'99), Tokyo, Japan, vol.3, pp.775–780, 1999.

[91] Zuberek, W.M., "Petri net models of process synchronization mechanisms"; Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'99), Tokyo, Japan, vol.1, pp.841-847, 1999.

[92] Zuberek, W.M., "Approximate performance evaluation of multithreaded distributed memory architectures"; Proc. 15-th Performance Engineering Workshop, Bristol, UK, pp.81–92, 1999.

[93] Zuberek, W.M., "Performance modeling of multithreaded distributed memory architectures"; in: **Hardware Design and Petri Nets**, A. Yakovlev, K. Gomes (eds.), pp.311–331, Kluwer Academic Publishers 2000.

[94] Zuberek, W.M., "Discrete–event simulation of timed Petri net models"; Proc. 33-rd Annual Simulation Symposium; Washington, D.C., pp.91–98, 2000.

[95] Zuberek, W.M., "Performance comparison of fine-grain and block multithreaded architectures"; Proc. High Performance Computing Symposium (HPC'2000), Washington, DC, pp.383–388, 2000.

[96] Zuberek, W.M., "Petri nets in hierarchical modeling of manufacturing systems"; Proc. IFAC Conf. on Control System Design (CSD'2000), Bratislava, Slovakia, pp.287-292, 2000.

[97] Zuberek, W.M., "Analysis of pipeline stall effects in block multithreaded multiprocessors"; Proc. 16-th Performance Engineering Workshop, Durham, UK, pp.187-198, 2000.

[98] Zuberek, W.M., "Timed Petri net models of cluster tools"; Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'2000), Nashville, TN, vol.4, pp.3063-3068, 2000.

[99] Zuberek, W.M., "Hierarchical analysis of manufacturing systems using Petri nets"; Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'2000), Nashville, TN, vol.4, pp.3021-3026, 2000.

[100] Zuberek, W.M., "Timed Petri nets in modeling and analysis of cluster tools"; IEEE Trans. on Robotics and Automation, vol.17, no.5, pp.562–575, 2001.

[101] Zuberek, W.M., "Petri net modeling and performance analysis of cluster tools with chamber revisiting"; Proc. IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA'01), Juan-les-Pins, France, pp.105–112, 2001.

[102] Zuberek, W.M., "Performance analysis of enhanced fine-grain multithreaded distributed-memory systems"; Proc. IEEE Conf. on Systems, Man, and Cybernetics (SMC'01), Tucson, AZ, pp.1101–1106, 2001.

[103] Zuberek, W.M., "Timed Petri net models of multi-robot cluster tools"; Proc. IEEE Conf. on Systems, Man, and Cybernetics (SMC'01), Tucson, AZ, pp.2729–2734, 2001.

[104] Zuberek, W.M., "Analysis of performance limitations in multithreaded multiprocessor architectures"; 2-nd Int. Conf. on Application of Concurrency to System Design, Newcastle, UK, pp.43-52, 2001.

[105] Zuberek, W.M., "Approximate simulation of distributed–memory multithreaded multiprocessors", Proc. 35-th Annual Simulation Symposium, San Diego, CA, 107-114, 2002.

[106] Zuberek, W.M., "Analysis of performance bottlenecks in multithreaded multiprocessor systems"; Fundamenta Informaticae, vol.50, no.2, pp.223-241, 2002.

[107] Zuberek, W.M., "Performance equivalence in the simulation of multiprocessor systems"; International Journal of Simulation, vol.3, no.1-2, pp.80-88, 2002.

[108] Zuberek, W.M., "Approximate simulation of distributed-memory multithreaded multiprocessors"; Proc. 35-th Annual Simulation Symposium, San Diego, CA, pp.107–114, 2002.

[109] Zuberek, W.M., "Performance balancing of fine-grain multithreaded distributed-memory multiprocessors"; Proc. High Performance Computing Symposium (HPC'02), San Diego, CA, pp.129-134, 2002.

[110] Zuberek, W.M., "Interconnecting networks and the performance of multithreaded multiprocessors"; Proc. SSGRR Summer Conf. on Infrastructure for e-Business, e-Education, e-Science and e-Medicine; L'Aquila, Italy, 2002.

[111] Zuberek, W.M., "Performance–equivalent multiprocessor systems", Proc. ICALP'2003 Stochastic Petri Net Workshop, Eindhoven, The Netherlands, pp.123–136, 2003.

[112] Zuberek, W.M., "Structural methods in performance analysis of discrete-event systems"; Proc. 9-th IEEE Int. Conf. on Methods and Models in Automation and Robotics, Miedzyzdroje, Poland, 2003.

[113] Zuberek, W.M., "Performance analysis of fine–grain multiprocessors"; systems"; International Journal of Simulation, vol.4, no.3-4, pp.12–20, 2003.

[114] Zuberek, W.M., Bluemke, I., "Hierarchies of place/transition refinements in Petri nets"; Proc. 5-th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA'96), Kauai, Hawaii, pp.355–360, 1996.

[115] Zuberek, W.M., Govindarajan, R., "Performance balancing in multithreaded multiprocessor architectures"; Proc. 4-th Australasian Conf. on Parallel and Real–Time Systems (PART'97), Newcastle, Australia, pp.15–26, 1997.

[116] 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 (CPN'98), Aarhus, Denmark, pp.253–270, 1998.

[117] Zuberek, W.M., Kubiak, W., "Timed Petri nets in modeling and analysis of simple schedules for manufacturing cells"; Computers and Mathematics with Applications, vol.37, no.12-12, pp.191–206, 1999.

[118] Zuberek, W.M., Rada, I., "Space partitioning and speedup in distributed generation of state spaces"; 15-th European Simulation Multiconference, Prague, Czech Republic, pp.554–559, 2001.

[119] Zuberek, W.M., Rada, I., "Modeling and analysis of distributed state space generation for timed Petri nets"; 34-th Annual Simulation Symposium (SS-2001), Seattle, WA, pp.93–98, 2001.

[120] Zuberek, W.M., Zuberek, M.S., "Transformations of timed Petri nets and performance analysis"; Proc. 33-rd Midwest Symp. on Circuit and Systems (Special Session on Petri Net Models), Calgary, Alberta, pp.774–778, 1990.