The Timed (Coloured) Petri Net Formalism: position paper

QMIPS groups in UTO+ZAR

Abstract

This paper presents the point of view of the groups of the universities of Torino and Zaragoza on the timed Petri net formalism. Coloured and non coloured models are covered at the same time, the latter making sense in case of systems without symmetries. We argue that Petri net formalisms allow more freedom than other established formalisms in the choices regarding the trade off between structural model complexity and interpretation of the graph. Some considerations are made on coloured Petri nets that could suggest them as a superior modelling formalism. The main comparison term that we selected are BCMP type queueing networks.

1 Basic ideas and concepts

1.1 The P/T level

A Petri net model [1, 2, 3] of a dynamic system consists of two parts: (1) a net structure, a weighted-bipartite directed graph that represents the static part of the system, and (2) a marking, representing a distributed overall state on the structure. The above separation allows reasoning on net based models at two different levels: structural and behavioural. Reasoning at the structural level, we can derive some "fast" conclusions on the behaviour of the modelled system relating (whenever possible) structural and behavioural properties. Purely behavioural reasonings based on state space analysis are usually computationally very heavy.

To model a discrete event dynamic system we need to take into account its possible states and the events leading to the state evolutions. In net systems the state is described by means of a set of state variables representing local conditions. Moreover, net models explicitly highlight the existence of state-transitions, therefore net structures are built on two disjoint sets of objects: places (represented as circles), and transitions (represented as bars or boxes). Places are the support of the state variables. Places and transitions are related through a weighted flow relation described by a (unweighted) flow relation F and a weighting function W. Formally, a Petri net is defined in the following manner:

Definition 1 *A Petri net is a four-tuple $\mathcal{N} = \langle P, T, F, W \rangle$ where P is a finite non-empty set of $n = |P|$ places, T is a finite non-empty set of $m = |T|$ transitions, $P \cap T = \emptyset$, i.e., places and transitions are disjoint sets, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation (set of directed arcs, $\mathrm{dom}(F) \cup \mathrm{range}(F) = P \cup T$), and $W : F \to \mathbb{N}^+$ assigns a weight to each arc.*

Non unitary arc weights allow to model for example bulk arrivals and bulk services. In many practical cases there exists neither bulk arrivals nor bulk services, so that usually all the arc weights are 1. In this case the net is said to be *ordinary*.

An alternative way to see Petri nets is to define the weighted flow relation through two incidence functions, Pre and $Post$, where $Pre : P \times T \rightarrow \mathbb{N}$ is the pre-incidence or input function and $Post : T \times P \rightarrow \mathbb{N}$ is the post-incidence or output function. There is an arc going from the place p_i to the transition t_j iff $Pre(p_i, t_j) > 0$. Similarly, there is an arc going from transition t_k to place p_i iff $Post(t_k, p_i) > 0$. The arc weight $Pre(p_i, t_j) = W(p_i, t_j)$ or $Post(t_k, p_i) = W(t_k, p_i)$ labels the corresponding arc. The pre- and post-sets of transition $t \in T$ are defined respectively as ${}^\bullet t = \{p | Pre(p, t) > 0\}$ and $t^\bullet = \{p | Post(t, p) > 0\}$. The pre- and post-set of a place $p \in P$ are defined respectively as ${}^\bullet p = \{t | Post(t, p) > 0\}$ and $p^\bullet = \{t | Pre(p, t) > 0\}$. A practical way of representing the net structure is to use incidence matrices. The pre- and post-incidence functions can be represented by means of pre- and post-incidence matrices, Pre and $Post$, both having $n = |P|$ rows and $m = |T|$ columns.

A pair comprised of a place p and a transition t is called a self-loop if p is both an input and output place of t. A Petri net is said to be *pure* if it contains no self-loop. A self-loop can be easily eliminated (e.g., by expanding the transition into a sequence: initial transition-intermediate place-final transition). Pure nets are completely characterized by the (single) incidence matrix $C = Post - Pre$. Positive (negative) entries in C represent the post- (pre-) incidence function. If the net is not pure, the incidence matrix "does not see" self-loops.

The structure of a net is static. Assuming that the behaviour of the system can be described in terms of the current system state and its possible changes, the dynamics of a net structure is specified by defining its marking and marking evolution rule.

Definition 2 *A marking M of a net $\mathcal{N}$ is an application of P on $\mathbb{N}$, i.e., an assignment of a non-negative integer (number of tokens) to each place.*

Definition 3 *A marked Petri net or net system is the pair $\langle \mathcal{N}, M_0 \rangle$, where $\mathcal{N}$ is a Petri net and M_0 is the initial marking.*

The number of tokens at a place represents the local state of the place (i.e., the value of the corresponding state variable). The state of the overall net system is defined by the collection of local states of places. A marking M is denoted as an n-vector whose p-th component $M(p)$ represents the number of tokens in place p. The vector M is the state-vector of the discrete event dynamic system described by the net system. Pictorially, we place $M(p)$ black dots (tokens) in the circle representing place p.

Once the distributed state is defined, the question is: How does a net system work? The evolution is defined through a firing or occurrence rule, informally called as the "token game". This is because net structures can be seen as "special checkers", the tokens as "markers" and the firing rule as the "game rule". Transitions represent potential moves in the "token game".

Definition 4 *A marking in a net system evolves according to the following firing (or occurrence) rule:*

1. *A transition is said to be enabled at a given marking if each input place has at least as many tokens as the weight of the arc joining them.*

2. *The firing or occurrence of an enabled transition is an instantaneous operation that removes from (adds to) each input (output) place a number of tokens equal to the weight of the arc joining the place (transition) to the transition (place).*

The pre-condition of a transition can be seen as the resources required for the transition to fire. The post-condition represent the resources produced by the firing of the transition.

An important remark concerning the firing rule of our abstract model is that enabled transitions are never forced to fire. This is a form of nondeterminism. In practical modelling the interpretation partially governs the firing of enabled transitions (e.g., depending on whether or not an external event associated to an enabled transition occurs).

The enabling and firing of a transition can be represented in a very convenient way using incidence matrices and marking vectors. Denoting the columns associated with t in the different incidence matrices as $Pre(t)$, $Post(t)$, and $C(t)$. Then, transition t is enabled at M iff $M \geq Pre(t)$. Representing as $M_1[t\rangle M_2$ the fact that M_2 is reached by firing t at M_1 (M_1 enables t), we have: $M_2 = M_1 + Post(t) - Pre(t) = M_1 + C(t)$. Finally assuming $\mathcal{N}$ to be pure (otherwise it can be easily transformed), it is not difficult to derive the following:

$$M_1[t\rangle M_2 \Leftrightarrow M_2 = M_1 + C \cdot e_t \geq 0 \tag{1}$$

where e_t is the characteristic vector of t: $e_t(x) :=$ if $(x = t)$ then 1 else 0. The right hand side of (1) is clearly a (linear) state equation: M_1 is the present state, M_2 the next state, and e_t the input vector. Unfortunately classical control theory is not of great help to study the dynamic behaviour of net systems: the state (marking) and input vectors must take their values on non-negative integers.

Integrating the state equation from M_0 along a firing sequence $\sigma = t_i t_j \ldots$ leading to M_k (M_k is said to be reached from M_0 by means of σ) we can write:

$$M_0[\sigma\rangle M_k \Rightarrow M_k = M_0 + C \cdot \vec{\sigma} \geq 0, \vec{\sigma} \geq 0 \tag{2}$$

where $\vec{\sigma}$ is the firing count vector of σ: $\vec{\sigma}(t)$ is the number of times t has been fired in σ.

Equation (2) is called the fundamental equation or, more frequently, the state equation of the net system[1]. The main important remark now is that only the right implication exists in (2). Otherwise stated, unfortunately a non-negative integer solution $\vec{\sigma} \geq 0$ of $M_k = M_0 + C \cdot \vec{\sigma} \geq 0$ does not imply that there exists a σ such that M_k is reachable from M_0 (i.e., does not imply $M_0[\sigma\rangle M_k$).

Petri nets, as introduced till now, are a mathematical formalism. Now, we present a number of features which—in our opinion—make nets an interesting modelling formalism, especially suited for discrete event dynamic systems with concurrent or parallel events and activities. These considerations are general, i.e., still on the abstract formalism, and valid independently of any particular interpretation. The reader can easily check the simplicity of representing with nets three basic modelling notions: causal dependence (e.g., sequence), conflict (decision, choice), and concurrency.

[1]Remark: properly speaking the state equation is (1) while (2) is the transition equation in control theory terminology.

As already mentioned, a major feature of nets is that they do not define in any way how and when (i.e., time independence) a given conflict should be solved, leading to nondeterminism on its behaviour. Sequence and conflict are classical notions in sequential systems (e.g., in finite automata); concurrency is a third concept that net systems represent in a extremely natural way (informally speaking two transitions are concurrent at a given marking if they can be fired "at the same time", simultaneously).

Synchronization is an additional concept very important in modelling distributed and concurrent systems. How are synchronizations modelled with nets? Basically with transitions with more than one input place. A less obvious way of synchronization may appear, even if a transition has only one input place, when the arc is weighted.

Possible extensions of the above presented place/transition nets are obtained with the addition of priorities to transitions and with the introduction of inhibitor arcs [4]. The priority function $\pi : T \rightarrow I\!N$ maps transitions into non-negative natural numbers, representing their priority levels. Then, in order for a transition to be enabled at a marking it is necessary that no higher priority transitions are enabled in the same marking. The inhibition function is a function $H : P \times T \rightarrow I\!N$. In the graphical representation of Petri nets, inhibition functions are represented by circle-headed arcs connecting the place p_j to the transition t_k if $H(p_j, t_k) > 0$. When greater than one, the value $H(p_j, t_k)$ weights the arc. Then, in order for a transition t_k to be enabled at a marking M it is necessary that $M(p_j) < H(p_j, t_k)$ for all $p_j \in P$ such that $H(p_j, t_k) > 0$.

Nets extended with inhibitor arcs or transition priorities in general allow to increase the theoretical modelling power of place/transition nets (they have the same descriptive power than Turing Machines in the unbounded case!). Therefore they are theoretically difficult to analyze in general. In bounded system however, inhibitor arcs or priority at transitions do not enlarge the modelling power too much [5] and make nets practical modelling facilities since they allow a substantial level of parametrization with respect to variations of the initial marking.

Historically, Petri nets have been used in many different ways and with different purposes. Among the most important ones are the study of net languages and the modelling of dynamic systems. In the former case, terminal elements are associated with transition firing and equivalencies are studied from the point of view of languages generated by firing transition sequences. In the latter case an interpretation is established among place markings and system states on one hand, and among transition firings and system state changes on the other. These two ways of using the formalism are semantically quite different, so that for instance equivalence results derived from the first approach hardly produce meaningful results when applied to models developed using the second approach. As an example of such a difference consider the classical proof of equivalence of the modelling power of ordinary versus non ordinary nets. Using the language generation approach one may transform the net shown in Figure 1.a into the one shown in Figure 1.b with the introduction of the two "invisible" transitions $t4$ and $t5$ (that are not associated with any terminal of the language). Of course these additional transitions have no physical meaning and in particular cannot be interpreted as immediate transitions (with higher priority) in order not to perturb the behaviour of the rest of the system[2]. Here we are obviously interested only in the use of net

[2]In order to obtain the correct physical behaviour, $t4$ and $t5$ should be defined as immediate transition with priority level *lower* than that of the three timed transitions, which is inconsistent with the timed model that we are going to introduce.

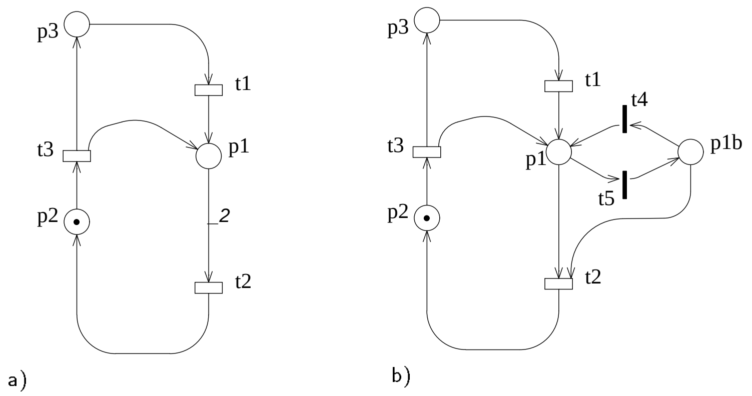

Figure 1: Language equivalence between net with multiple arcs and ordinary net

formalisms for the modelling of dynamic systems.

A Petri net can be used to model a discrete event dynamic systems assigning a meaning to its associated elements (places, transitions, and tokens) and relating explicitly the modelled system and its environment: In general, the behaviour of a system is influenced by the environment (through events in our case), while the actions generated by the system influence the behaviour of its environment. Therefore, interpreting a net system corresponds to establishing a conventions for: (1) the meaning of places, transitions and tokens; (2) a meaning for the conditions which govern the transition firing; the marking evolution rule is slightly modified by the interpretation which becomes also a function of the behaviour of the modelled system's environment; and (3) the actions generated by the model.

If the behaviour of a net system is not influenced by the environment, such a system is said to be autonomous. Non-autonomous net systems have more constrained behaviour than the underlying autonomous net system. Many possible interpretations exist even for a given class of problems or application domain.

Uninterpreted Petri nets do not include any notion of time and are aimed to model only the logical behaviour of systems by describing the causal relations existing among events. The introduction of a timing specification is essential if we want to use this class of models to consider performance, scheduling, or real-time control problems.

1.2 High level nets

High Level Petri Net (HLPN) [6, 7] can be used to build compact and parametric models that require cumbersome structures if specified using classic PN. Often PN models are composed of several similar subnets: a more compact representation can be obtained by "folding" similar subnets one on the other. For example let's consider the PN model of the dining philosophers problem depicted in figure 2.(a). All the philosophers behave in the same way: indeed the same subnet is repeated for each philosopher. We can obtain a more compact representation by drawing only one subnet and adding some annotation to denote the fact that it actually represents several similar subnets. The problem when folding places one onto the other is that we lose information if tokens remain anonymous: indeed tokens that were originally contained in different places are now mixed together; for example when place *think* in the unfolded net contains two tokens, we know that two philosophers are thinking but we don't know the identity of

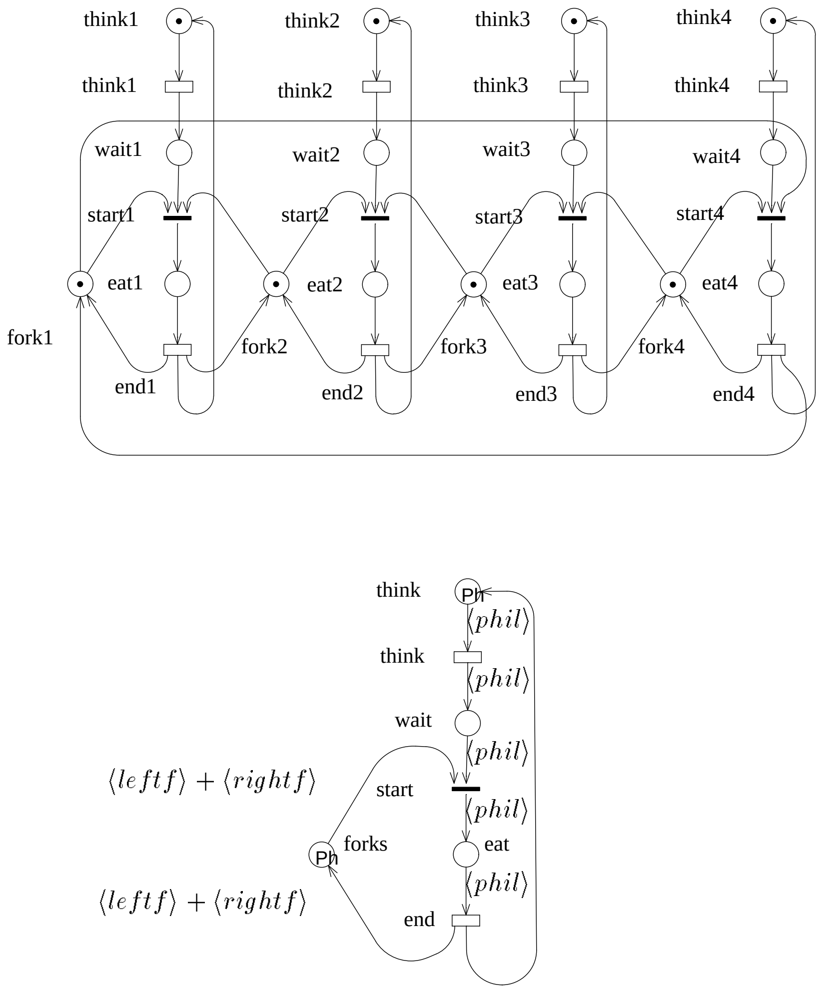

Figure 2: PN and HLPN representation of the "dining philosophers" problem.

these philosophers. Similarly when firing a folded transition we cannot say which of the original transitions represented by the folded one is actually firing. To overcome this problem we can use "colors" to distinguish (a) token originally belonging to different places and (b) different firing instances among those represented by a folded transition firing. In our example we could use a set Ph of colors of cardinality equal to the number of philosophers in the system, and attach a philosopher identity to each token and transition firing. In this way tokens become distinguishable, and it is necessary to define the color of the tokens to be removed from/added into places when a given transition instance fires. This is achieved by labelling the arcs with functions.

Observe that the new representation is more concise and more parametric: systems with a different number of philosophers share the same net structure, the only difference being the *cardinality* of the philosophers color set. On the contrary, a different structure is needed to represent these systems using the non coloured formalism.

1.2.1 WN Definition

A particular class of HLPN that will be used in the sequel of this paper is that of Well Formed Petri nets (WN). In this formalism, tokens may carry information, i.e., they are no longer undistinguishable and transition firings are parameterized: a transition represents a *class* of events rather than a single event. When a transition fires, the actual value of its parameters must be specified to define which specific event in the class occurs. The set of all the possible values of a transition parameter constitutes its *color domain*. Usually, different instances of a given transition cause different state changes: this must be specified in the model description by means of arc labeling functions from the set of transition color instances to the set of input/output place colors.

As in PNs, places of WNs together with their markings, play the role of describing the system state while transitions represent events that cause state changes. Indeed a token can be regarded as an instance of a given data type. Often it may be convenient to have "structured data types", i.e., data types that are obtained as the composition of some elementary data types (called *basic color classes*) by means of suitable operators. A typical composition operator is the Cartesian product: in this case the information associated with a token is a data structure with a certain number of typed fields which are in turn basic color classes. The definition of the "data type" associated with a given place p is called *place color domain* (denoted $C(p)$). The notation $M(p)$ denotes the marking of place p, i.e. the multiset of $C(p)$ contained in p according to marking M.

Transitions in WNs can be considered as procedures with formal parameters. A transition whose formal parameters have been instantiated to actual values is called *transition instance*. We use the notation $[t, c]$ for an instance of transition t, where c represents the *assignment* of actual values to the transition parameters. Observe that an assignment c is actually an element of the set $C(t)$ and for this reason it is often referred to as a "color instance" of t. In order to fire a transition, it is necessary to specify actual values for its formal parameters i.e., we can only fire *transition instances*. The enabling check of a transition instance, and the state change caused by its firing, depend on the *function* labeling the arcs that connect the transition to its input, inhibition and output arcs. Optionally, a predicate can be associated to a transition: it is a function $\phi_t : C(t) \rightarrow \{true, false\}$. In this case, a transition instance $[t, c]$ is enabled only if $\phi_t(c)$ returns "true".

The declaration of the basic color classes, used to define place and transition *color domains*, is part of the model definition. The declaration of a basic color class C_i includes (1) the list of objects it contains, (2) a set of functions $\mathcal{F}_i = \{f : C_i \rightarrow C_i\}$, and (3) a set of predicates $\mathcal{P}_i = \{p : C_i \times C_i \rightarrow \{\text{true}, \text{false}\}\}$.

The functions labeling the arcs, also called *arc expressions*, are structured according to the corresponding place color domain. If the place color domain contains k basic color classes (i.e., k "fields"), then the corresponding arc expression is a weighted sum of k-tuples. The tuples coefficients comprise two parts: an integer and an optional predicate on the transition formal parameters. The optional predicates are used to define "variant" arc expressions: given a transition color instance, only the expression terms associated with a *true* predicate have to be considered as part of the function. The whole expression denotes a multiset in the corresponding place color domain.
The j^{th} element in each k-tuple is an expression denoting a multiset of C_j, where C_j is the "type" of the j^{th} "field" in the place color domain. This element is a weighted sum of functions from $\mathcal{F}_j$ applied to either objects of C_j or to variables of type C_j, representing the transition formal parameters. An arc expression that contains variables can be

interpreted as a *pattern* standing for any multiset that can be obtained binding the variables to actual elements in the proper basic color class. We call *assignment* a collection of variable bindings.

The set of variables appearing in all arc expressions related to a single transition are its formal parameters. Observe that when the same variable appears in many arc expressions related to the same transition, the different occurrences actually denote the same object, on the other hand when the same variable is used within several arc expressions related to different transitions, there is no relation between the objects represented by the different occurrences of the variable.

1.2.2 Coloured token game

The enabling of a transition instance $[t, c]$ is determined by evaluating the transition predicate and the arc expressions of all its input and inhibitor places with respect to the assignment c. Notice that in this case an arc expression can be seen as a *function*, whose arguments are the variables appearing in the expression itself. A transition instance $[t, c]$ is enabled iff the predicate evaluates to true, iff each input place contains the multiset resulting from the evaluation of the corresponding arc expression and iff for each inhibitor place, each tuple contained in it has a smaller multiplicity than the same tuple in the multiset resulting from the evaluation of the corresponding arc expression.

An enabled transition instance $[t, c]$ can fire. The state change caused by the firing amounts to subtracting/adding from/to each input/output place p the multiset resulting by evaluating the corresponding arc expression through the assignment c. The notation used to indicate the arc expression labeling an input, inhibitor or output arc connecting $t \in T$ and $p \in P$ is $W^+(p, t)$, $W^h(p, t)$, and $W^-(p, t)$ respectively.

Observe that many instances of the same transition can be concurrently enabled. They are considered as independent, concurrently occurring events (unless they are in conflict).

Notice that even if the initial idea behind HLPN was to fold similar subnets, if complex arc functions are allowed, places and transitions could be folded even if they do not have any similarity. As an extreme, we could fold a whole PN model reducing it to a HLPN with only one place and one transition, but with very complex functions on the arcs connecting them. The disadvantage of hiding too much of the net description in the arc functions is that the model becomes very difficult to understand and to analyze. As we shall discuss in the following, the main interest of the introduction of coloured nets in our framework is the efficient analysis of symmetric systems. This is obtained by relating the syntax of the colour function definitions to the symmetries that we want to exploit in the analysis. Thus in our approach folding non symmetric systems with the only purpose of reducing the graphic complexity of the model is not particularly interesting.

1.3 Timed nets

The concept of time was intentionally avoided in the original work by C.A.Petri [8], because of the effect that timing may have, in general, on the behavior of nets. Indeed, the association of timing constraints with the activities represented in a PN model may prevent certain transitions from firing, thus destroying the important property of being able to capture the essence of the behavior of a system with the structure of the net only. The concept of time is however essential when the efficiency of an application is the subject of a study, and this is the reason why timed Petri nets are extensively used in areas like computer architecture, communication protocol, and software system analysis.

More generally, we can recognize that Petri nets are inherently non deterministic in many aspects (e.g. the conflict resolution, the actual firing in case of enabling, etc.). The nondeterminism must be reduced in order to obtain a model that is completely specified from a behavioural point of view, thus allowing for instance a reproducible simulation experiment. The introduction of the concept of firing time after the enabling represents part of the behavioural specification of a model that reduces the non determinism related to the actual firing of transitions. A probabilistic routing policy may be another way of reducing the basic non determinism in case of conflicting transitions. In case of coloured net models the specification of a (queueing) policy for the extraction of different tokens from places in relation to transition firings may also be needed.

The pioneering works in the area of timed Petri nets were performed by Merlin and Faber [9], and by Noe and Nutt [10]. In both cases, Petri nets were not viewed as a formalism to statically model the logical relationships among the various entities composing real system, but rather for the description of the global behavior of complex structures. The nets were used to *tell* all the possible stories that the system could experience, and the temporal specifications were an essential part of the picture.

Different ways of incorporating timing informations into PN models have appeared in the literature. In principle time can be either associated with places or with transitions. It is possible to show that these two ways of specifying time are (almost) equivalent and we will thus consider in the sequel of this paper only the second alternative that is the most commonly used. A discussion of the implication of associating time with places can be found in [11].

1.3.1 Transition Timing

The firing of a transition in a PN model corresponds to an event that changes the system state. This change of state can be due to one of two reasons: it may either be induced by the completion of some activity in the system, or it may results from the verification of some logical condition. Considering the former case we note that transitions can be used to model activities so that transition enabling periods correspond to activity executions and transition firings correspond to the instants of activity completions. Hence, time can be naturally associated with transitions which will then be called *timed*.

As we noted before, not all the events that occur in a real system model correspond to the end of time-consuming activities (or of activities that are considered time-consuming at the level of detail at which the model is developed). Indeed, some may correspond to logical conditions that change without actually consuming any time. In these cases it may be convenient to introduce a second type of transitions called *immediate*. Immediate transitions fire as soon as they become enabled (with a null delay). Hence it is possible to give them *priority* over timed transitions in a consistent way. In net representation thick bars and boxes are used for timed transitions, while thin bars are used for the immediate ones that have always higher priority.

1.3.2 Parallelism and Conflict

The introduction of temporal specifications in PN models must not change the modeling capabilities with respect to the untimed case. Let us verify this condition as far as parallelism and conflict resolution are concerned.

Consider, as the simplest possible example, the timed transition $t1$ in Figure 3. Transition $t1$ can be associated with a local clock or timer. When a token is generated in place p_1, $t1$ becomes enabled, and the associated timer is set to an initial value. The timer is then decremented at constant speed, and the transition fires when the timer reaches the zero value. The timer associated with the transition can thus be used to model the duration of an activity whose completion induces the state change that is represented by the change of marking produced by the firing of $t1$. [3]

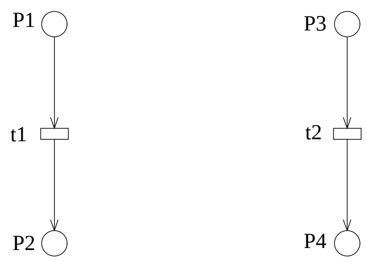

Figure 3: Non-conflicting concurrent transitions

Pure parallelism can be modeled by two transitions that are enabled in the same marking. The evolution of the two activities is measured by the decrement of the clocks associated with the two transitions. Consider the net in Figure 3: transitions t_1 and t_2 are not related, and model two completely independent concurrent activities. When one of the counters (for example that associated with t_1) reaches zero, the transition fires and a new marking is produced. In the new marking, transition t_2 is still enabled, and its counter goes on unaffected; the possible ways of managing timers is discussed in the following in case for the case of conflicting transitions.

It is important to note that the activity is assumed to be in progress while the transition is enabled. This means that in the evolution of more complex nets, an interruption of the activity may take place in the real system if the transition loses its enabling condition before it can actually fire. The activity may be resumed later on, during the evolution of the system when a new enabling of the associated transition takes place. This may happen several times, until the timer goes down to zero and the transition finally fires. As an example of this type of behavior, consider the timed PN system depicted in Figure 4, where transitions t_3 and t_4 behave exactly as transition $t1$ in Figure 3. Transitions t_1 and t_2 have however a more complex dynamic behavior since they belong to a free-choice conflict, and the firing of either one of them disables

[3] A three phase firing rule is also proposed in the literature where an enabled transition removes first the token(s) from its input place(s) (first phase), absorbs the token for a given amount of time (second phase); and then deposits the token(s) in the output place(s) (third phase) [12]. Again, nets executing according to this firing rule can be proved to be equivalent to nets with atomic firing rule. We will thus avoid considering the three phase firing rule due to its inconsistency with the untimed PN model.

the other. The choice of the transition to fire first may be performed according to two possible policies: *race* and *preselection.* In the race policy case, the choice is accomplished using the temporal specification of the net. Referring again to the net of Figure 4, if the initial marking is the one shown in the figure, the timers of t_1 and t_2 are set to their initial values, say θ_1 and θ_2, with $\theta_1 < \theta_2$, after a time θ_1, transition t_1 fires, and the timer of t_2 is stopped. Now the timer of t_3 is started and decremented at constant speed until it reaches the value zero. After t_3 fires, the conflict comprising t_1 and t_2 is enabled again. The timer of t_1 must be set again to an initial value (possibly, again θ_1), whereas the timer of t_2 can either resume from the point at which it was previously interrupted, i.e., from $\theta_2 - \theta_1$, or be reset to an initial value. The use of timers to solve conflicts represents a way of reducing the non determinism of the net to a *race* among concurrent activities. Of course the result of the race is determined only if $\theta_1 \neq \theta_2$, otherwise a time-independent specification (i.e. an independent selection) must be adopted. In the preselection policy case instead, a random choice is performed among the conflicting transitions according to an "external" specification and the timer of the chosen transition is the only one to be set. A particularly simple case of free-choice conflict resolution with preselection policy occurs when $\theta_1 = \theta_2 = 0$ (i.e. the transitions are immediate). In this case only the independent conflict resolution policy is needed to reduce the non determinism of the underlying net (assuming the earliest firing mechanism).

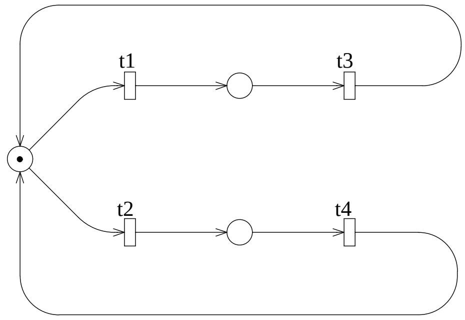

Figure 4: A timed PN system

When two or more immediate transitions are enabled in the same marking, an analysis must be performed to distinguish whether their firings produce (mutual) interference or they represent truly concurrent events. In the first case, some rule must be specified to solve the conflict and to decide which transition to fire first. Two types of rules may be used to solve conflicts. The first is based on a deterministic choice of the transition to fire using the mechanism of *priority.* The second mechanism consists of the association of a discrete probability distribution function with the set of conflicting transitions and of the corresponding random choice.

Notice that the priority that immediate transitions have over timed ones allows to easily implement a preselection policy among timed transitions, separating the conflict resolution from transition timing specifications (see Figure 5). The presence of immediate transitions induces a distinction among markings. Markings in which no immediate transition is enabled are called *tangible*, whereas markings enabling at least one immediate transition are said to be *vanishing.* The timed PN system spends a

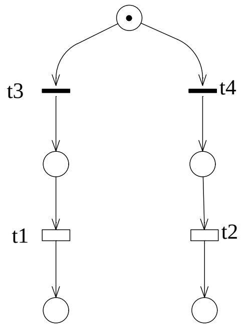

Figure 5: Using immediate transitions to separate conflict resolution from transition timing specification

positive amount of time in tangible markings, and a null time in vanishing markings. This characteristics can be exploited to simplify the performance evaluation of a timed PN system since vanishing markings do not contribute to average performance figures.

1.3.3 Memory

An important issue that arises when conflicting timed transitions are disabled before the corresponding timers go to zero is how to reset their timers upon their re-enabling. Different possible alternatives exist of setting the timer, or, equivalently, of taking into account the time already spent by a transition "executing" its corresponding activity. The issue is more generally a question concerning the *memory policy* of the transition. Many different ways of keeping track of the past behavior are possible and adequate to modelling different situations of real systems. We describe here the following two *memory policies*:

- **Enabling memory**. The memory of the past is recorded with an *enabling memory variable*, associated with each transition. The enabling memory variable accounts for the work performed by the activity associated with the transition since the last instant of time it became enabled. The timer associated with the transition in this case is discarded whenever the transition loses its enabling condition. This policy can be used to represent activities that take place while the rest of the system evolves, but that are lost if they are interrupted before their completions. In these cases when such activities resume, they must start from scratch and no memory is kept of the work performed before the interruption. This policy is useful to represent faults.

- **Age memory**. The memory of the past is recorded with an *age memory variable*, associated with each transition. The age memory variable accounts for the work performed by the activity associated with the transition since the time of its last firing. The timer associated with the transition in this case is reset whenever the transition becomes first enabled after firing. This policy is the most common in parallel and distributed systems where activities may be interrupted, but they resume later on when proper conditions are satisfied without loosing any of the work performed before the interruption.

1.3.4 Multiple Enabling

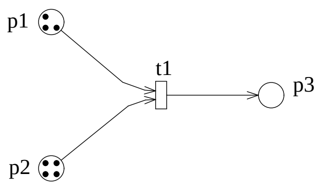

Figure 6: Timed transition with enabling degree 3

Special attention must be paid to the timing semantics in the case of timed transitions enabled more than once. The concept of multiple enabling of a transition in a marking can be formalized using the notion of *enabling degree.* The enabling degree of a transition in a given marking is the maximum number of enablings that could be served independently if the transition would act as an "infinite server".

Consider transition t_1 in Figure 6. It is enabled whenever one or more tokens are in place p_1. The number of independent enablings of $t1$ in the depicted marking is 3, since each enabling of $t1$ gets simultaneously 1 token from $p1$ and 1 from $p2$ (so there is one token in place $p2$ that does not belong to any enabling).

The enabling degree of a transition is a function of time, since the marking of its input places vary with time. If $M[p](\tau)$ is the marking of place p at time τ, then the enabling degree $e_i(\tau)$ of transition t_i at time τ is given by:

$$e_i(\tau) = \min_{p_k \in {}^\bullet t_i} \left\lceil \frac{M[p_k](\tau)}{W(p_k, t_i)} \right\rceil$$

(where $\forall a \in I\!\!R$, $\lceil a \rceil$ denotes the largest integer not greater than a). Different semantics are possible when several tokens are present in p_1. Borrowing from queueing network terminology, we can consider the following different situations.

Single server semantics : the temporal specification associated with the transition is independent of the enabling degree. In other words we can say that each transition firing is processed serially by the transition.

Infinite server semantics : the temporal specification associated with the transition depends on the enabling degree. Each transition firing is processed independently of possible other firings of the same transition with no queueing delay.

Multiple server semantics : the temporal specification associated with the transition depends on the enabling degree up to a predefined threshold. The first K instances of transition firings are processed in parallel, while additional firing instances must wait for "a server to become free."

The infinite-server semantics exactly corresponds to the concept of internal transition concurrency for untimed PNs. Multiple or single server transitions may be easily obtained by limiting the enabling degree to a maximum value K (possibly $K = 1$) by means of the addition of a self loop on a place containing exactly K tokens in the initial marking of the system. In the particular case of exponentially distributed timed transitions the multiple server semantics might also be viewed as a form of "marking dependency" of the firing rate of a single server transition. Actually the latter is the way in which multiple (infinite) server transitions have been historically introduced in the framework of stochastic Petri net models. We think however that this approach is now obsolete, since it fails in establishing a strong relation between timed and untimed Petri net semantics. More complex forms of marking dependencies may also be envisioned for SPN models, but an extensive exploitation of marking dependencies may introduce problems in the model specification and in its dynamic behavior [13].

1.3.5 Firing and Marking Processes

The evolution of a Petri net depends on the successive firing of its transitions. A characterization of this evolution may be obtained identifying for each transition of the net a sequence of non-decreasing real values that represent their firing *epochs*. Such firing sequences can be denoted with

$$[\tau_i(1), \tau_i(2), \cdots \tau_i(n), \cdots]$$

where $\tau_i(n)$ represents the time of the $n-th$ firing of transition t_i.

Given the firing sequences of each transition of the net, it is possible to define a *timed execution* of a timed PN as a transition sequence augmented with a set of nondecreasing real values describing the epochs of firing of each transition. Such a timed execution, that is also called *firing process*, derives from the merging of the individual transition firing sequences and is denoted as follows:

$$[(\tau_{(1)}, t_{(1)}); \cdots ; (\tau_{(j)}, t_{(j)}); \cdots]$$

The time intervals $[\tau_{(i)}, \tau_{(i+1)})$ between consecutive epochs represent the periods during which the timed PN system sojourns in marking $M_{(i)}$. This sojourn time corresponds to a period in which the execution of one or more activities is in progress and the state of the system does not change. Notice that in case of null sojourn time such a firing process is not completely defined by the transition firing sequences. The structure of the net may in some case impose a total ordering among events occurring at the same time, but in general concurrency is allowed thus defining only a partial ordering among simultaneous events. This is not a problem from the point of view of measuring performance indexes from the observation of the firing process (at least in case of confusion free PNs [13]) since admissible interleavings may result in differences only for vanishing states.

Focusing the attention on the states that are reached by the net during its execution, we can observe the existence of a *marking process* $\{M(\tau), \tau \geq 0\}$ whose state space is the set of possible markings of the net. The same remarks on the indetermination of the process in case of concurrent simultaneous events applies as in the case of the firing process.

1.3.6 Modelling features

The generality of the Petri net formalism allows the natural representation of customers, resources, and processes as tokens placed at the initial marking of the net in properly defined places. This generality has the drawback that no special symbols are used to represent these entities and the interpretation of the meaning of the net may in some cases derive only from the proper assignment of names to places and transitions of the net. Conflicts among immediate transitions may be used to represent the routing of customers within the net. Conflicts comprising timed and immediate transitions allow to model the interruption (or preemption) of ongoing activities when some special situation is verified. This last feature is easily represented in the model thanks to the priority that immediate transitions have over timed ones. Figure 7 illustrates the case of three customers waiting for service while one other customer is already in service.

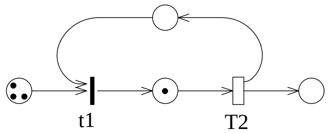

Figure 7: Customers accessing a single server system

Similarly, the presence of customer classes that group entities with similar, but not identical behaviour, must be explicitly represented in Petri net models with the corresponding need of duplicating certain structures as it is depicted in Figure 8 where two classes of customers compete for the use of the same resource; timed transitions t_2 and t_3 may easily account for different average service time for the customers of the different classes.

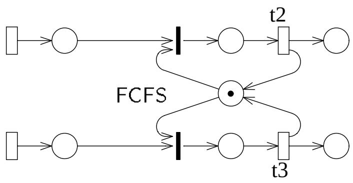

Figure 8: Multiple classes of customers accessing a single server system

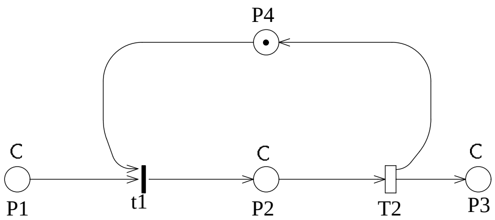

Figure 9: Colored representation of multiple class customers accessing a single server system

When enough symmetry exists in the model, the introduction of colours may help in simplifying the representation of multiple classes of entities yielding models that can be considerably compact (see Figure 9). A final remark must be made concerning the possibility of the definition of queueing policies. The enabling degree of a transition is determined by the minimum number of tokens satisfying the input functions among the input places. In one input place there might be a higher number of tokens than required for the repeated number of firings as many times as its enabling degree. In this case a choice must be made for the tokens to remove or to leave in the input places, and this choice may be important for the determination of the behavior of the model in case of tokens with different colours. Either random choice (in the absence of further specification) or usual queueing policies (such as FCFS, etc.) may be used to reduce this form of non determinism. In a coloured net queueing policies may be easily represented in an explicit manner using colour functions for the selection of the tokens to be withdrawn from the input places (for instance adding a counter "field" in the token colour that is set accordingly to the order of "arrival"). Alternatively, an external interpretation such as the one used in BCMP type queues may be added together with the non deterministic net specification.

1.4 Underlying mathematical and logical concepts

We discuss the underlying mathematics separately for the three classes of nets that have been introduced: untimed P/T; coloured; timed (coloured).

1.4.1 Untimed P/T nets

The decomposition of a net system model into a static part (the structure and the initial marking) plus a dynamic part (the firing rule) is essential to understand the model and to derive specific analysis and synthesis techniques. Since the net structure can be represented as a graph or as a matrix, mathematical concepts from both graph theory and linear algebra underlay Petri net models. Petri nets can be considered as a graph theoretic tool specially suited to model and analyze discrete event dynamic systems which exhibit parallel evolutions and whose behaviours are characterized by synchronization and sharing phenomena. Graph based objects are circuits, P- and T-components, siphons (structural deadlocks), traps, handles, and bridges. Structural boundedness, structural repetitiveness, conservative (P-semiflows) and consistent (T-semiflows) components, and structurally implicit places are linear algebraic concepts or objects.

To be able to use a Petri net in modelling a given type of application, we must enrich it with an adequate interpretation. The interpretation of graph theoretic tools is nothing new. A graph (in its theoretical sense) is a set of objects (nodes) with relations. With a graph, the connectivity between sites (towns, points in a circuit...) can be represented using obvious interpretations. Another kind of interpretation on graphs allows to model discrete and finite dynamic systems: the nodes represent the states of the system, the arcs represent transitions between states. Particularizing a little more the state-based interpretation, State Diagrams (SD), and State Transition Diagrams (STD) are widely used as interpreted graphs: SD's allow the modelling of finite state sequential switching systems, while STD's allow the modelling of homogeneous finite Markov chains. For both formalisms (SD's and STD's) the evolution of the system can be defined in continuous time (asynchronous state diagrams; state transition rate diagrams) or in discrete time (synchronous state graphs; discrete time state transition diagrams).

If the behaviour description of a net model is linearly relaxed, linear programming and convex geometry techniques can be applied to the analysis problem. Linear descriptions such as the net state equation or a basis of marking linear invariants are convex closures of the marking and firing count vectors belonging to the behaviour of the net. Therefore, solutions of the linear description may result that do not correspond to the net behaviour ("spurious" solution). In general, the net state equation is a closer approximation to the net behaviour than a basis of linear invariants and the latter is closer than that provided by the minimal non-negative linear invariants. For some particular net subclasses, two or more different linear descriptions coincide and, in some cases, completely characterize the net behaviour.

Petri nets, as a formal model for concurrent systems, admit the definition of equivalence notions similar to those proposed for Milner's Calculus of Communicating Systems [14]. Equivalence notions can be used to compare systems on the basis of the behaviour they show to their environment. Each transition of a net model can be labelled as observable or unobservable and the language of firing sequences of observable transitions can be used to compare a net system with another one.

The number of tokens at a place represents a local state (of that place) and the state of the overall net system is defined by the collection of local states of the places. Transitions firings produce state changes. From this point of view, a Petri net can be considered as an automaton. The enumeration of all reachable markings and transition firings (generating the state changes) leads to the construction of the reachability graph (finite or infinite, if the net system is unbounded). In the case of unbounded models, coverability graphs can be constructed that constitute a partial, but finite representation of the reachability graph.

1.4.2 High-level nets

Concerning high-level nets, all above considerations are still valid since the unfolding of the high-level net model gives a place/transition net model. Additionally, the presence of some symmetries in the behaviour can be exploited by means of the automatic construction of the symbolic reachability graph [15], where a symbolic marking represents an equivalence class on the state space of the model, and the equivalence is in terms of the possible basic colour permutations that yield the same behaviour.

1.4.3 Timed nets

As a consequence timed Petri nets, as non-autonomous net models, are derived by adding a timing interpretation to underlying Petri nets. All fundamental mathematical and logical concepts supporting Petri net models are inherited by timed net models (graph theory-based, linear algebra and convex geometry-based...).

However, for performing timing analysis of the model, graph theory or linear algebra are not sufficient and other concepts and techniques must be considered. Stochastic process and operational analysis are classical research fields that provide adequate tools for the quantitative analysis of performance models. Operational analysis is a conceptually very simple way of deriving mathematical equations relating measurable quantities in queueing systems. Operational analysis techniques can be also applied to derive linear equations and inequalities relating interesting performance measures in timed Petri net models [16]. The main conceptual difference between queueing and Petri net models is the presence of a synchronization primitive in the latter. Therefore, operational inequalities for synchronizations that have no counterpart in operational laws for queueing networks must be derived.

The operational approach allows to derive properties of a given behavioural realization of the timed model. If statistical statements about all possible behaviour sequences are to be obtained, then the theory of stochastic processes must be introduced. In particular, Markov processes [17] and stochastic recurrent equations [18] can be used to understand and quantify the statistical behaviour in all possible realizations. Lumping of the underlying Markov chain can be applied in some particular cases to exploit the symmetry properties of models with respect to different colours belonging to the same basic colour classes in well-formed coloured nets.

1.5 Results and performance measures

An important characteristics of TPN models is the possibility of defining and computing not only performance results in strict sense, but also other figures that characterize the behaviour of the model from a "qualitative" point of view. The latter results strongly rely on our a-priori assumption of coherence between the underlying net semantics and the timing specification. We discuss separately these two classes of results.

1.5.1 Qualitative and logical properties

We can identify two classes of qualitative properties of a Petri net model: global properties characterizing the complete model and local properties referring to some of its components. Among global properties we may identify reachability, boundedness, liveness, and home space as the principal ones. Local properties may be divided in place and transition properties. Among place properties we may identify: generalized siphons, traps, flows and semi-flows, marking bounds, and marking mutual exclusions. Relevant transition properties include enabling and liveness bounds, conflicts, causal connections, enabling and firing mutual exclusions, concurrency, and synchronic distance.

Reachability: can be used whenever we want to answer questions of the type: "is it possible for the modelled system to be in state x?". A marking M' is said to be *immediately reachable* from M if and only if there exists a transition t enabled in M whose firing change the state from M to M' (denoted $M[t\rangle M'$). M' is said to be *reachable* from M iff there exists a sequence of transitions σ_M such that $M[\sigma_M\rangle M'$. We denote $[M\rangle$ the set of all markings reachable from M.

Boundedness: a place $p \in P$ is said to be *k-bounded* iff $k \in I\!N$ is the minimum value such that $\forall M \in [M_0\rangle$, $M[p] \leq k$. A place p is said to be bounded iff $\exists k < \infty$ such that p is k-bounded. A P/T system is said to be bounded iff all its places are bounded.

Liveness: a transition $t \in T$ is said to be *k-live* iff $k \in I\!N$ is the maximum value such that $\forall M' \in [M_0\rangle$, $\exists M'' \in [M'\rangle\ : E(t, M'') \geq k$. A transition t is said to be live iff $\exists k > 0$ such that t is k-live. A P/T system is said to be live iff all its transitions are live.

Reversibility: represents the possibility for a P/T system to come back to its initial state infinitely often. A system is said to be *reversible* iff $\forall M \in [M_0\rangle$, $M_0 \in [M\rangle$. A generalization of this concept is the idea of home space. A marking $M' \in [M_0\rangle$ is said to be a *home state* iff $\forall M \in [M_0\rangle$, $M' \in [M\rangle$. The set of all home states is called the *home space*. The initial marking is a home state iff the system is reversible.

Marking characterization: are linear inequalities that are proven to be invariant assertions for all reachable markings $M' \in [M\rangle$. In general they are computed by linear algebraic techniques applied to the incidence matrix of the net. In [19] a detailed discussion of these properties can be found.

Transition properties: besides liveness, other transition properties may be defined to characterize the service, concurrency, and choice features of a model. These qualitative properties have strict counterparts in the timing/stochastic interpretations of a performance model.
Effective conflict: $\forall t_i, t_j \in T$, $\forall M \in [M_0\rangle$, $t_i EC(M) t_j$ iff $M[t_i\rangle M'$ and $ED(t_j, M) > ED(t_j, M')$.
Structural conflict: $\forall t_i, t_j \in T$, $t_i SC t_j$ iff $\exists p \in (\,^\bullet t_i \cap {}^\bullet t_j)$ such that $W(p, t_i) > W(t_i, p)$.
Relation between structural and effective conflict: $\forall t_i, t_j \in T$, if $\exists M \in [M_0\rangle$: $t_i EC(M) t_j$ then $t_i SC t_j$.
Effective causal connection: $\forall t_i, t_j \in T$, $\forall M \in [M_0\rangle$, $t_i ECC(M) t_j$ iff $M[t_i\rangle M'$ and $ED(t_j, M) < ED(t_j, M')$.
Structural causal connection: $\forall t_i, t_j \in T$, $t_i SCC t_j$ iff $\exists p \in (t_i^\bullet \cap {}^\bullet t_j)$ such that $W(p, t_i) < W(t_i, p)$.
Relation between structural and effective causal connection: $\forall t_i, t_j \in T$, if $\exists M \in [M_0\rangle$: $t_i ECC(M) t_j$ then $t_i SCC t_j$.
Other properties related to the mutual exclusion and concurrency are defined in [19].

1.5.2 Quantitative or performance properties

Basic measurable quantities may be defined using an operational approach ([20]) and considering the timed Petri net model as the specification of a dynamic system. Average performance indexes defined in terms of these basic operational quantities may be introduced without any particular stochastic assumption on probability distributions. More sophisticate performance indexes may be defined instead in terms of reward stochastic processes based on the marking process.

Basic operational quantities

Assume the timed Petri net system is observed during an experiment starting at time $\tau = 0$ and ending at time $\tau = \theta$. Assuming the infinite server semantics, one may collect a set of measures during such an hypothetical experiment.

The **instantaneous marking**, i.e. the number of tokens observed in place p_k at time τ, is denoted with $M[p_k](\tau)$ ($\forall p_k \in P$, $\forall \tau\ :\ 0 \le \tau \le \theta$). The **average marking** of place p_k during the experiment interval is defined as

$$\forall p_k \in P, \bar{M}[p_k] = \frac{1}{\theta} \int_0^\theta M[p_k](\tau) d\tau$$

The **instantaneous enabling degree** of transition t_i (i.e. the internal concurrency of transition t_i at time τ), is defined as:

$$\forall t_i \in T, \forall \tau \ : \ 0 \leq \tau \leq \theta \ \ e_i(\tau) = \max\{k \in I\!N : \forall p \in \ {}^\bullet t_i \ \ M[p](\tau) \leq kW(p, t_i)\}$$

and it can be interpreted as the number of active servers associated with the transition firing.

The **total enabling work** Θ_t, i.e. the total active time for all server associated with the transition firing during the experiment, is defined as

$$\Theta_t \ \triangleq \ \int_0^\Theta e_t(\tau) \ d\tau$$

$\forall t \in T$ such that ${}^\bullet t \neq \emptyset$. From the definitions of total enabling work and instantaneous enabling degree, the **average enabling degree** can be defined as:

$$\forall t_i \in T, \ \ \bar{e}_i(\theta) \ \triangleq \ \frac{\Theta_t}{\theta}$$

and it represents the average number of active servers associated with t_i during the experiment interval.

In the case of transitions the observable quantity is the number of firings, so it is possible to define the **total firing count** $F_t(\tau)$ for each transition t as the total number of firings observed from 0 to τ.

The throughput of transitions, i.e. the actual firing frequencies of transitions observed during the experiment, can be thus defined as

$$\forall t \in T \ \ x_t(\theta) \ \triangleq \ \frac{F_t(\theta)}{\theta}$$

if $\theta > 0$.

The average service time of transitions can also be defined as a basic operational quantity. In case of persistent transitions t (i.e. never disabled before firing) or transitions with age memory policy, the average service time is a function of the total enabling work, i.e. $\forall t \in T$, t persistent or with age memory,

$$\bar{S}_t(\theta) \ \triangleq \ \frac{\Theta_t}{F_t(\theta)}$$

The above operational quantities have been defined for nets without conflicts. In the general case in which transitions may be enabled in conflict, the definitions of service time and average enabling degree must be modified in order to take the possibility of preemption into account. In the case of race policy the work of an enabled transition can be wasted due to preemption, so only the “useful” work done should be taken into consideration.

If the preselection policy is used, only the selected transitions put their servers to work and fire for sure after the elapsing of their firing time, but the same results of the race policy case can be derived.

The **useful total enabling work** Θ'_t is the work really completed by transition t, i.e. its number of enablings not preempted by any other conflicting transition. The formal definition of Θ'_t requires the identification of the servers associated with transitions. However for the sake of clarity we do not present here the formal definition (a complete discussion of this point can be found in [39]).

The average service time for transitions that may be preempted can be thus defined as

$$\bar{S}'_t(\theta) \triangleq \frac{\Theta'_t}{F_t(\theta)}$$

For the race policy the useful average service time might be strictly less than the nominal transition firing times due to the effect of preemption from conflicting transitions ($\Theta'_t < \Theta_t$). In the case of preselection policy instead, the useful service time of a transition is exactly the transition firing time.

Among the basic operational quantities above defined the following operational law holds for any measurement experiment:

Enabling operational law $\forall t_i \in T$, $\bar{e}_i(\theta) = x_i(\theta)\bar{S}_i(\theta)$. The enabling operational law can be considered as a generalization of the well known "Utilization law" derived in the framework of single server queues. From the enabling law follows that if the average firing time of a transition is known, then its throughput is proportional to its average enabling degree. In the case of conflicting transitions the enabling operational law is easily extended substituting $\bar{e}_i(\theta)$ and $\bar{S}_i(\theta)$ with respectively $\bar{e}'_i(\theta)$ and $\bar{S}'_i(\theta)$.

An interesting application of the operational analysis technique for timed Petri net is the rederivation of Little's law which in this framework holds under much weaker conditions.

In particular it may be proven that $\forall p \in P$,

$$R_p(\theta)\left(\frac{M_0[p]}{\theta} + \sum_{t_i \in {}^\bullet p} W(t_i,p)x_i(\theta)\right) = \bar{M}[p](\theta)$$

where $R_p(\theta)$ is the average residence time for tokens in place p. Under the hypothesis of ergodicity that limits exist for $\tau \to \infty$, the "steady state" version of Little's law apply

$$\forall p \in P \qquad \bar{R}_p = \lim_{\theta \to \infty} R_p(\theta) = \frac{\bar{M}[p](\infty)}{\sum_{t_i \in {}^\bullet p} W(t_i,p)x_i(\infty)}$$

In [39] has been shown that the throughput and the average service time of a transition is linearly related with the average marking of each of its input places by the relation:

$$\forall t_i \in T, x_i(\theta) \cdot \bar{S}_i(\theta) \le \min_{p_k \in {}^\bullet t_i} \left\{ \frac{\bar{M}[p_k](\theta)}{W(p_k,t_i)} \right\} \tag{3}$$

The above equation establishes an upper bound for the average enabling in the case of transitions with more than input place (modelling thus a synchronization). If the service time is defined, inequality (3) establishes an upper bound for the transition throughput. The above inequality derives from the consideration that at any point τ in time $\forall p_k \in {}^\bullet t_i$, $e_i(\tau) \le \frac{\bar{M}[p_k](\tau)}{W(p_k,t_i)}$.

Operational analysis for timed Petri nets allows one to derive also lower bounds for the average enabling of transitions. An example of such relation is the following inequality that holds for single input, persistent or immediate transitions: $\forall t_i \in T$: (t_i is persistent $\vee$ $\bar{S}_i(\theta) = 0$) and ${}^\bullet t_i = \{p\}$, $x_i(\theta) \cdot \bar{S}_i(\theta) \ge \frac{\bar{M}[p](\theta) - W(p,t_i)+1}{W(p,t_i)}$.
The above lower bound on the average enabling of transitions can be intuitively explained in the following way: if the marking of place p is less than $W(p,t_i)$, then

transition t_i is never enabled. However $M[p](\theta) \geq W(p, t_i) + \epsilon$ implies that transition t_i is enabled at least for the fraction of time in which the inequality holds, and consequently that the average enabling degree cannot be less than the above quantity (since it is an integer quantity).

Similar inequalities for lower bounds on throughputs have been derived in [39], in which has been also shown how to use the above relations to compute performance bounds.

A little remarks on the average service time of transitions is now needed. From the above discussion one may think that the average service time is a variable in the model, since it has been defined as an operational quantity using basic operational quantities. This derivation has been stated only to allow one to relate average service time, throughput and average enabling degree of transitions. Nevertheless it should be noted that during the development of the model, and during its performance analysis, the average service time of transitions is one of the model parameters.

Model specific results: definition in terms of reward on marking and firing processes. For a more detailed discussion see [21].

1.6 Specific features

Petri nets constitute a unique framework for:

- A graphical and precise formalism which allows easy and deep dialogues about the expected behaviour of the system among the different teams that participate in the design process (designers, owners, users...).
- A well founded theory for qualitative verification of net model properties (liveness, fairness, boundedness...).
- A reasonable offer for quantitative analysis (performance evaluation), presently undergoing important developments.
- A technology independent implementation method providing, in particular, some well understood code generation techniques for the real-time control software from the net model.

One aspect of the usefulness of Petri net models is their possibility of expressing all basic semantics of concurrency: interleaving, step, and partial order semantics. All these semantics can be compared within the Petri net formalism. In this sense, Petri nets are capable of modelling "true concurrency". The importance of true concurrency in a performance oriented concurrent model can be explained from the "temporal realism" that provides step and partial order semantics of concurrent events.

Identifying within a net in a bipartite structure and a marking as separate concepts makes the net based approach very powerful for modelling purposes. In particular, the dichotomy places/transitions leads to an even treatment of states and actions. This makes—in our opinion—nets superior to either purely state- or purely transition-oriented formalisms where one of the notions is explicit and the other has to be deduced.

The existence of a locality principle on states and actions (transitions) in net models is a direct consequence of its bipartite structure and of its marking definition. The importance of the locality principle resides in the fact that net models can be locally modified, refined or coarsed, without altering the rest of the model. This means, in particular, that nets can be synthesized using top-down and bottom-up approaches. Top-down synthesis is any procedure that starting with an initial (very abstract) model, leads to the final model through stepwise refinements. In a bottom-up approach modules are produced, possibly in parallel by different groups of designers, and later composed. Restricting the many possible refinements and compositions strategies, we just mention here place and transition refinements and compositions through merging of transitions (i.e., synchronization of modules) and merging of places (i.e., fusion of modules).

The possibility of progressive modelling is absolutely necessary for computer systems, communication networks, and flexible manufacturing systems because they are usually large/complex systems. The refinement mechanism allows the building of hierarchically structured net models. A trade-off between structure and interpretation complexity is possible. If the refinement-transformation rule catalog is well selected and the model well designed, the design can be easily or trivially proven correct with respect to the specifications.

Therefore, compositions and refinements help to deal with model complexity. Obviously this work can be done with place/transition nets as well as with more abstract formalisms, like coloured nets.

Summarizing, still at an abstract level, net systems have the following practical features for modelling:

- Graphical and equational representations. Therefore, net systems enjoy some comparative advantages for documentation and analytical studies.

- Natural expression of causal dependences, conflicts, and concurrency.

- Simple, appealing and powerful synchronization mechanism making natural the construction of mutual exclusion constraints.

- Locality of states and actions which allows the hierarchical and the modular construction of large net models.

2 Technical aspects

2.1 Untimed P/T nets

Distributed and concurrent systems are complex and difficult to master for designers by nature. Therefore, desirable "good properties" must be formally defined and the model must be validated for these properties. Techniques for analyzing net systems can be divided into the following groups:

- Analysis by simulation.

- Analysis by enumeration.

- Analysis by transformation.

- Structural analysis.

Simulation methods are called dynamic and proceed exercising the net system model under certain strategies. In this case some bugs can be detected (e.g., some deadlocks), providing "some confidence on the model", if problems do not arise during the simulation process. Anyhow, in general, simulation methods do not allow to prove properties, even if they might be of great help for understanding the modelled system. In particular, simulation methods are extremely useful when time is associated with the net evolution (timed systems), or when we wish to know the response of the system described with a net in an environment which is also defined by simulation.

The other three groups of techniques are called static methods, and their application to net systems as abstract models leads to exact results. Enumeration methods are based on the construction of a reachability graph (RG) [3] which represents the net markings and transition firings. If the net system is bounded, the reachability graph is finite and the different qualitative properties can be verified easily. If the net system is unbounded, the above graph is infinite and it is thus impossible to construct. In this case, finite graphs known as coverability graphs can be constructed. In spite of its power, enumeration is often difficult to apply, even in nets with few places, because of its computational complexity (it is strongly combinatory).

Analysis by transformation [22, 2] is based on the following idea: given a net system $\langle \mathcal{N}, M_0 \rangle$ in which we wish to verify the set of properties Π, we transform it into the net system $\langle \mathcal{N}', M_0' \rangle$ such that:

1. $\langle \mathcal{N}', M_0' \rangle$ satisfies the properties Π iff $\langle \mathcal{N}, M_0 \rangle$ satisfies them (i.e., the transformation preserves the properties Π).

2. It is easier to verify the properties Π in $\langle \mathcal{N}', M_0' \rangle$ than in $\langle \mathcal{N}, M_0 \rangle$.

Reduction methods are a special class of transformation methods in which a sequence of net systems preserving the properties to be studied is constructed. The construction is done in such a way that the net system $\langle \mathcal{N}^{i+1}, M_0^{i+1} \rangle$ is "smaller" than the previous in the sequence, $\langle \mathcal{N}^i, M_0^i \rangle$.

The applicability of reduction methods is limited by the existence of irreducible net systems. Practically speaking, the reductions obtained are normally considerable, and can allow the desired properties to be verified directly. Because of the existence of irreducible systems, this method must be complemented by some other methods.

Finally, structural analysis techniques [23, 3, 24] carefully consider the net structure (hence their name), while the initial marking acts, basically, as a parameter. Structural analysis techniques investigate the relationships between the behaviour of a net system and the structure of the net.

In this last class of analysis techniques, we can distinguish two subgroups:

- Linear algebra / linear programming based techniques, which are based on the net state equation. In certain analysis they permit a fast diagnosis without the necessity of enumeration.

- Graph based techniques, in which the net is seen as a bipartite graph and some "ad hoc" reasonings are applied. These methods are especially effective in analyzing restricted subclasses of ordinary nets.

The three groups of analysis techniques outlined above are by no means exclusive; they are instead complementary. Normally the designer can use them according to the needs of the ongoing analysis process. Obviously, although we have distinguished between reduction and structural analysis methods, it must be pointed out that most popular reduction techniques act basically on the net structure level and thus can be considered also as structural techniques.

Net subclasses can be defined exclusively by introducing constraints on the structure of ordinary nets [3]. Therefore it is very easy to recognize if a net model belongs to a subclass (i.e., the membership problem). By restricting the generality of the model, it will be easier to study its behaviour. In particular, powerful structural results allow to fully characterize some (otherwise hard to study) properties as liveness and reversibility.

Definition 5 *Let $\mathcal{N}$ be an ordinary net. Then,*

1. *$\mathcal{N}$ is a state machine (SM) if: $\forall t \in T, |{}^\bullet t| = 1$ and $|t^\bullet| = 1$. That is, any transition has one input and one output place.*

2. *$\mathcal{N}$ is a marked graph (MG) if: $\forall p \in P, |{}^\bullet p| = 1$ and $|p^\bullet| = 1$. That is, any place has one input and one output transition.*

3. *$\mathcal{N}$ is a free choice net (FCN) if: $\forall p \in P, |p^\bullet| > 1 \Rightarrow |{}^\bullet t_k| = 1, \forall t_k \in p^\bullet$. That is, if two transitions, t_i and t_j, have a common input place p, it is the only input place of t_i and t_j.*

4. *$\mathcal{N}$ is a simple net (SN) if: $\forall t \in T, |\{p \in {}^\bullet t \mid |p^\bullet| > 1\}| \leq 1$. That is, each transition has at most one input place shared with other transitions.*

SM's allow the modelling of decisions (conflicts) and reentrance. It is important to note that the concept of state machine, considered as a subclass of nets, is more general than the classical state diagram or state graph, since it can have more than one token. In any case, state machines do not "create" tokens; thus can model only finite state systems.

MG's is a subclass of structurally decision-free nets. They can model systems for ordering activities as PERT's (Program Evaluation and Review Technique) do. They are more general than PERT's in the sense that recycling is allowed and places can contain several tokens. Moreover, provided with an adequate stochastic interpretation, strongly connected MG's are equivalent (i.e., have the same descriptive power) to Fork/Join Queueing Networks with Blockings (FJQN/B).

Here it is worth noting that the modelling capacities of MG's and SM's are dual in the sense that SM's can model choices, but not synchronizations, and, on the other hand, MG's can model synchronizations, but cannot model choices.

FCN's include SM's and MG's; therefore, they can model some restricted interleaving among choices and synchronizations, although not with the generalitity provided by ordinary nets. FCN's can be considered as an extension of:

- SM's by allowing MG's-type synchronization (i.e., if two places share a common output transition then this is their unique output transition), or

- MG's by allowing SM's-type conflicts (i.e., if two transitions share a common input place, then this is their unique input place).

It is not difficult to realize that both statements represent identical restrictions. FCN's are a common generalization in which choices and conflicts do not directly interfere with each other.

From a behavioural point of view, FCN's are models in which either all or none of the output transitions of each place are enabled. In an FCN, if a place is marked and has more than one output transition, the transition to be fired can be freely chosen (i.e., independently of the rest of the marking). Hence the name. The behavioural and structural analysis of FCN's is particularly elegant and well understood.

FCN's do not allow the modelling of sequential subsystems synchronized through mutual exclusion semaphores (i.e., shared resources). The subclass of SN's allows it in some simple case: when no more than one exclusion semaphore is considered in any synchronization. In SN's, choices are not-free in general but they can be solved locally, because each choice is centered around a unique shared place. In spite of its relative generality, this subclass has some interesting properties. Nevertheless, its behaviour is by far not so well understood as that of FCN's.

The structure of the above presented net subclasses is rich enough to give plenty of information on the net systems we can define by putting an initial marking. This is particularly true for FCN systems and their subclasses, MG and SM systems. For SN systems there exist also some interesting results, but the strongest properties of FC systems cannot be extended (e.g., the Rank Theorem, or the liveness monotonicity with respect to the initial marking).

2.2 High-level nets

Analysis techniques that are peculiar of coloured net models are the construction and analysis of the symbolic reachability graph (SRG), the symbolic simulation, the structural decolorization, and the parametric computation of invariant relations. All these techniques aim at exploiting symmetries in the coloured models to reduce the complexity of the analysis, while obtaining the same results as in the case of uncoloured net models without structural symmetries.

A simple way of obtaining results for symmetric coloured nets that can be applied as a general principle is to "lift" results derived for P/T nets. A semiautomatic way of doing this is to perform a study of the unfolded net for once, and then try to write the results in parametric way using a proper formalism such as the one of Well-Formed Nets. This general principle has been already successfully followed for net reductions [25, 26] and for the derivation of operational bounds [27].

2.2.1 The Symbolic Reachability Graph

Structural symmetries in HLPN models, often lead to behavioral symmetries that can be exploited to reduce the computational cost of analysis methods based on the reachability graph construction [28, 25].

The Regular Net (RN) and Well-Formed Colored Petri Net (WN) formalisms [25, 29] have been introduced to systematize the symmetry exploitation technique: behavioral symmetries of models described with this formalisms can be automatically discovered and exploited. This goal is achieved by defining equivalence classes of markings called *symbolic markings*. A *Symbolic Reachability Graph* (SRG) can be directly generated without previously generating the ordinary RG. The SRG retains enough information to study the qualitative properties of the model.

The core of the SRG generation method relies on the definition of object types (the basic color classes), and, for each type, of disjoint object subsets that exhibit homogeneous behavior[4]. The set of functions defined on the basic color classes are restricted to a small set: the identity function (x), the successor function ($\oplus x$, applicable only to *ordered* classes), and the diffusion function: a function that returns all the objects of a basic color class (S_{class}) or all the objects of a static subclass ($S_{subclass}$). In the particular case of subclasses comprising only one colour, the diffusion function becomes a constant colour. Restricting the set of functions to a subset with particular symmetry properties, allows us to define equivalence classes of markings as sets of markings that are equal up to a permutation of homogeneously behaving objects. Observe that this definition implies that the potential aggregation decreases as the partitioning of classes into subclasses increases.

Symmetries in WN are implicitly defined at the color class level, by means of *symmetry functions*. A symmetry function s_i on an ordered/unordered color class C_i is any rotation/permutation on the elements of C_i that satisfies the following constraint[5]: $\forall c \in C_i, d(s_i(c)) = d(c)$, i.e. permutations are allowed only among objects in the same static subclass. A symmetry functions $s = \{s_1, s_2, \dots, s_n\}$ applicable to place markings and transition color instances is defined as follows:
$\forall s \in \xi, \forall c_i^j \in C_i,$

$$s(\langle c_1^1, c_1^2, \dots, c_2^1, c_2^2, \dots, c_n^1, c_n^2, \dots\rangle) = \langle s_1(c_1^1), s_1(c_1^2), \dots, s_2(c_2^1), s_2(c_2^2), \dots, s_n(c_n^1), s_n(c_n^2), \dots\rangle.$$

We denote wit with ξ the set of all such functions. The following proposition allows to define the marking equivalence classes, called *symbolic markings*.

Proposition 1 *The firing property is preserved by applying a permutation both on the markings and the transition instantiation. $\forall M$ ordinary marking, $\forall t \in T$, $\forall c \in C(t)$, $\forall s \in \xi$,*

$$M[t,c\rangle M' \iff s.M[t,s(c)\rangle s.M'$$

Definition 6 (Symbolic marking) *Let Eq be the equivalence relation defined by:*

$$M \ Eq \ M' \iff \exists s \in \xi, \ M' = s.M$$

An equivalence class of Eq is called a symbolic marking, denoted with $\mathcal{M}$.

Proposition 1 states that the possible future qualitative evolution of the model is the same for all the markings in an equivalence class; hence the aggregation of markings into equivalence classes (inducing a "folding" of subgraphs of the RG) doesn't cause any loss of information.

In Figure 10 a portion of RG is depicted: it schematizes the meaning of Proposition 1. Given two markings M and M' in the same equivalence class, there exists a symmetry function $s \in \xi$ such that $M' = s.M$. Any transition instance $[t, c]$ fireable from M (represented by an arc exiting from the box labelled M) can be associated with a corresponding instance $[t, s(c)]$ fireable from M'. In Figure 10 this is represented by means of similar dashed arcs exiting from equivalent markings. The corresponding arcs in the RG lead to two new markings that belong to the same equivalence class (e.g., $M2$ and $M2'$).

[4]In the WN terminology these subsets are called *static subclasses*.

[5]Function $d(c)$ returns the static subclass object c belongs to.

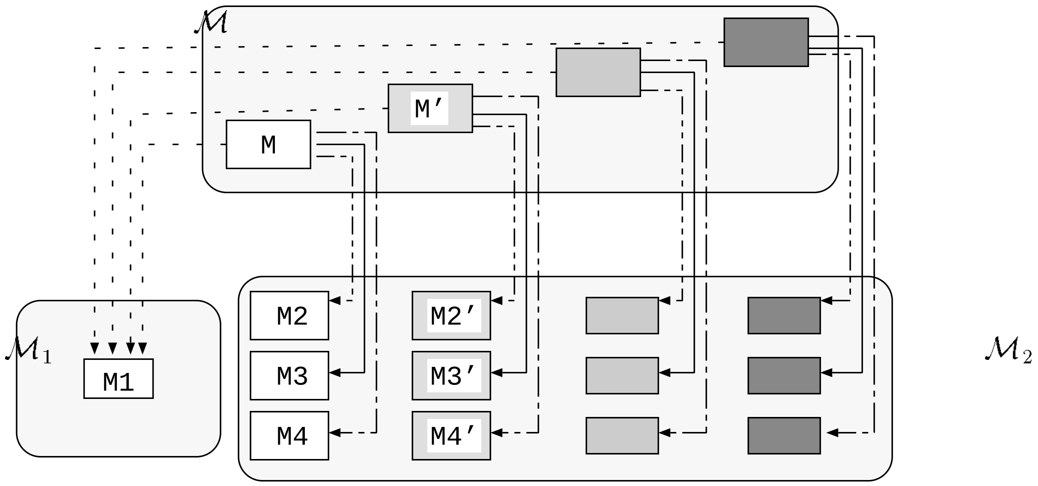

Figure 10: An example of marking and firing equivalence classes.

Consider the system in Figure 11: it is a closed system composed of two service centers in tandem. Let's assume that there are 5 customers in the system. Customers cycle between two service centers. The first one is a single server machine, while the second is a multiple server machine with four servers. A customer chooses randomly one of the four servers on each visit to the multiserver node. In Figure 11.(b) a WN representation of the system is depicted.

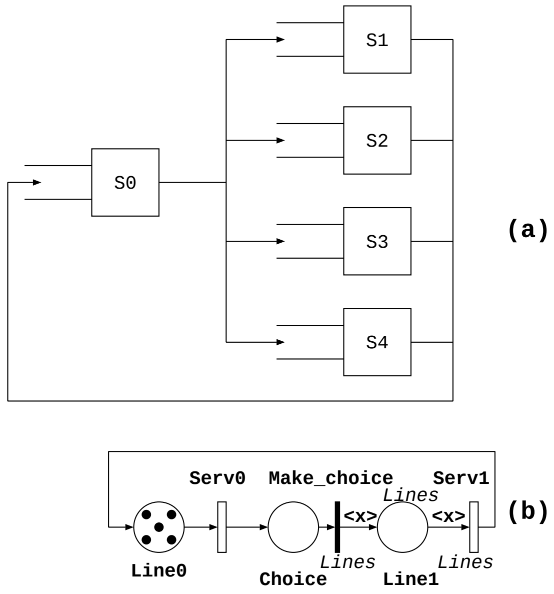

Figure 11: An example queueing system and its WN model

Only one color class is used in this model, namely $Lines = \{l_1, \ldots, l_4\}$, representing the four servers in the multiserver queue. The presence of n tokens with associated color l_i in place $Line1$ means that n customers are queued up to get service from server S_i. A possible marking of this model is $M = Line0()Choice()Line1(\langle l_1\rangle, \langle l_2\rangle, \langle l_3\rangle, 2\langle l_4\rangle)$.

There are three markings equivalent to M, that can be obtained by applying all the possible $Lines$ objects permutations to M. Thus M belongs to a symbolic marking $\mathcal{M}$ of cardinality four. The symbolic marking $\mathcal{M}$ could be represented as follows: $\mathcal{M} = Line0()Choice()Line1(\langle z_1\rangle, \langle z_2\rangle, \langle z_3\rangle, 2\langle z_4\rangle)$ where z_i are variables representing objects in $Lines$ and all the ordinary markings belonging to $\mathcal{M}$ can be obtained by assigning actual objects to variables (different objects must be assigned to different variables). Four transition instances are enabled in all the markings belonging to $\mathcal{M}$: $[Serv1, l_i], i = 1, \ldots, 4$. The situation is schematized in Figure 10: from the symbolic marking $\mathcal{M}$, two symbolic markings can be reached: $\mathcal{M}_1$ (containing only one ordinary marking) and $\mathcal{M}_2$ (containing twelve ordinary markings). Observe that three out of four firing instances exiting from M end up in the same symbolic marking $\mathcal{M}_2$. Actually it is possible to know in advance which firing instances lead to markings in the same equivalence class: indeed all the objects that have the same *distribution of tokens* in places can be interchanged in a firing instance without changing the symbolic marking reached after the firing. For example, in marking M objects l_1, l_2 and l_3 have the same distribution of tokens in places (all of them appear only in place $Line1$ with multiplicity one). Firing $Serv1$ instanced to any object in the set $\{l_1, l_2, l_3\}$, a new marking is reached that belongs to $\mathcal{M}_2$. In order to exploit this property, it is convenient to use a representation for symbolic markings that keeps objects with the same distribution of tokens grouped[6] into sets. We call these sets *dynamic subclasses*, they are denoted Z^j_{class} and are characterized by their cardinality. The new representation of $\mathcal{M}$ using dynamic subclasses is: $\mathcal{M} = Line0()Choice()Line1(\langle Z^1_{Lines}\rangle, 2\langle Z^2_{Lines}\rangle)$, where $|Z^1_{Lines}| = 3$ and $|Z^2_{Lines}| = 1$. The *symbolic firing* is performed for only one representative for each dynamic subclass, so that the enabled symbolic firing instances in $\mathcal{M}$ are $[Serv1, Z^1_{Lines}]$ and $[Serv1, Z^2_{Lines}]$ where the former actually represents three ordinary firing instances, all going to the same symbolic marking $\mathcal{M}_2$.

2.2.2 Symbolic simulation

The notion of symbolic marking can be used also to speed up the simulation of WNs. The advantage of using such representation in the implementation of a simulator is due to the *saving in space* obtained by grouping firing instances into equivalence classes thus reducing the event list length and the number of enabling tests to be performed. The technique and the experimental evaluation of its efficiency are described in [30, 31].

2.2.3 Decolorization

In some cases the presence of symmetry in the state space can be due to redundancy in the color definition of the model. By redundancy we mean the unnecessary introduction of color specification that causes some states to be considered as distinct even if such distinction is not needed for the description of the system behavior. In [32, 33] a structural analysis algorithm for the discovery and elimination of redundancy in WN models has been proposed. The simplification of the color structure consists of "shrinking" the color classes so that some objects that are distinguishable in the original model become undistinguishable in the "decolorized" one. The RG of the decolorized model is smaller than that of the original model, but contains all the information needed to study the system properties. The SRG size for the decolorized model can be smaller

[6]Only objects in the same *static subclass* can be grouped.

than (or equal to) the SRG size of the original model, however the SRG generation for the decolorized model is in any case computationally less expensive (due to the simplification of the colour domains). Another possible simplification of the color structure specification consists of merging a set of static subclasses into a single static subclass: in this case the RG size is the same for the two models, while the SRG size is smaller for the decolorized model.

2.2.4 P and T invariants

Structural analysis techniques are very useful to study properties of models without building the whole state space. P and T invariant analysis techniques require the solution of a system of linear equations associated with the incidence matrix of the net. When HLPN are considered, some difficulty arises because the coefficients of the matrix are linear functions instead of integers. One possible solution is to *unfold* the net into the corresponding PN and to apply the classic invariant computation algorithm to the unfolded net. This approach poses two problems: (1) in order to build the unfolded net the basic color classes must be instanced, so that a parametric analysis cannot be performed, (2) the large number of invariants and their description in terms of the unfolded net can cause many difficulties for their interpretation. It is thus desirable to devise an invariant computation method specific to HLPNs: the ultimate goal is to develop an algorithm for the computation of a generative family of *symbolic invariants* (Place/Transition weights in symbolic invariants are linear functions instead of natural numbers; a symbolic invariant describes many invariants of the unfolded net with similar structure). By now this problem has been solved only for some subclasses of HLPNs, namely: Regular Nets [25]; Unary Nets [34]; Commutative Nets [35]. Only partial solutions exist for general HLPNs [6, 36, 37, 38].

2.3 Timed nets

Discrete event simulation is a general technique that applies to all proposed versions of TPN formalism. Operational bounds were initially proposed in relation to structural subclasses of TPN models without specific assumptions on the timing distributions. Now such results have been extended to general P/T net models [39] as well as to some restricted cases of Well-formed, symmetric coloured nets [27], thus promoting the computation of operational bounds as a general technique for the analysis of TPN models under very weak assumptions.

Additional analysis techniques (mostly numerical ones) can be applied to particular types of marking and firing processes. Several subclasses of timed Petri nets have been proposed in the literature, each one related to a specific numerical analysis technique. Here we consider: GSPNs, DSPNs, product form SPNs, and SWNs.

2.3.1 GSPNs

Stochastic Petri Nets (SPN) are a particular class of Timed Petri Nets in which the firing delay associated with each transition is a random variable with negative exponential distribution. The memoryless property of this distribution allows to show in a relatively simple manner that SPN are isomorphic to continuous time Markov Chains (CTMC). In particular a k-bounded SPN can be shown to be isomorphic to a finite MC that can be obtained following these simple rules: (1) The MC state space $S = \{s_i\}$ corresponds to the reachability set $RS(M_0)$ of the PN associated with the SPN ($M_i \leftrightarrow s_i$); (2) The transition rate from state s_i (corresponding to marking M_i) to state s_j (M_j) is obtained as:

$$q_{ij} = \sum_{k \in E_j(M_i)} w_k \tag{4}$$

where $E_j(M_i) = \{h : t_h \in E(M_i) \wedge M_i[t_h\rangle M_j\}$ is the set of transitions enabled by marking M_i whose firing generates marking M_j.

As discussed in the section on timed Petri nets, it is often convenient to allow certain transitions of a net to be *immediate* in order to represent logical actions that take place in no time. SPN models of the type described above, in which immediate transitions coexist with timed transitions, are known with the name *generalized SPN* (GSPN) [40, 13]. In the graphical representation of GSPNs, immediate transitions are drawn as bars or segments, and timed transitions as (white) rectangular boxes.

Immediate transitions fire with priority over timed transitions. Thus, if timing is disregarded, the resulting PN model comprises transitions at different priority levels. Different types of transitions induce different types of markings. The reachability set of a *GSPN* comprises *tangible* and *vanishing* markings. A tangible marking is a marking in which (only) timed transitions are enabled. A vanishing marking is a marking in which (only) immediate transitions are enabled (the "only" is in brackets since the different priority level makes it impossible for timed and immediate transitions to be enabled in the same marking). The time spent in any vanishing marking is deterministically equal to zero. On the contrary, the time spent in tangible markings is positive with probability one.

In order to describe the GSPN dynamics, we separately observe the timed and the immediate behavior, hence referring to tangible and vanishing markings, respectively. Let us start with the timed dynamics; this is identical to the dynamics in SPN and TPN as well. We can assume that each timed transition possesses a timer. The timer is set to a value that is sampled from the negative exponential pdf associated with the transition when the transition becomes enabled for the first time after firing. During all time intervals in which the transition is enabled, the timer is decremented. Transitions fire when their timer reading goes down to zero. No special mechanism is necessary for the resolution of timed conflicts: the temporal information provides a metrics that allows the conflict resolution (the probability of two timed transitions firing at the same time being zero).

In the case of vanishing markings, the GSPN dynamics consumes no time. If several immediate transitions are enabled, a metrics is necessary to identify which transition will produce the marking modification. Actually, the selection of the transition to be fired is relevant only in those cases in which a conflict must be resolved: if the enabled transitions are concurrent, they can be fired in any order. For this reason, *GSPN*s associate *weights* with immediate transitions belonging to the same conflict. The transition weights are used to compute the firing probabilities of the simultaneously enabled transitions comprised within the conflict. We can thus observe a difference between the specification of the temporal information for timed transitions and the specification of weights for immediate transitions. The temporal information associated with a timed transition only depends on the characteristics of the activity modeled by the transition. Thus, the temporal specification of a timed transition requires no information on the other (possibly conflicting) timed transitions, or on their temporal characteristics. On the contrary, for immediate transitions, the specification of weights must be performed considering at one time all transitions belonging to the same conflict. Indeed, weights are normalized to produce probabilities by considering all enabled transitions within a conflict, so that the specification of a weight, independent of those of the other transitions in the same conflict, is not possible. When several transitions are enabled in the same marking, the probabilistic choice of the transition to fire next depends on parameters that are local to these transitions and that are not function of time. The general expression for the probability that a given transition t_k, enabled in marking M_i fire is:

$$P\{t_k|M_i\} \ = \ \frac{w_k}{q_i} \tag{5}$$

where $q_i \ = \ \sum_{j \in E(M_i)} \ w_j$.

When the marking is vanishing, the weights w_k of the immediate transitions enabled in that marking represent the selection policy that is used to make the choice. When the marking is tangible, the weights w_k of the timed transitions enabled in that marking are the rates of their associated negative exponential distributions and represent the race policy used to select the transition to fire next. The average sojourn time in vanishing markings is zero, while that of tangible markings is given by the following expression:

$$SJ_i \ = \ \left[\sum_{j \in E(M_i)} \ w_j \right]^{-1} \tag{6}$$

Numerical solution of GSPN models The application purposes for which *GSPN* have been proposed allow the following assumptions to be made: (1) The reachability graph is finite; (2) Firing rates do not depend on time parameters; (3) The initial marking is reachable with a non-zero probability from any marking in the reachability graph (i.e., it is a home state). No marking (or group of markings) exists that "absorbs" the process. These assumptions further specify the nature of the stochastic process associated with the *GSPN* so that an embedded Markov chain (EMC) can still be recognized within the process.

The probability of each vanishing marking is known a priori to be zero. This suggests that a technique can be devised to reduce the computation to tangible states only [40]. The solution of the chain allows the direct computation of the mean number of visits performed by the net to tangible states only between two subsequent visits to a reference state. The stationary probability distribution associated with the set of tangible states is thus readily obtained by means of their average sojourn times using standard semi-Markov process techniques.

2.3.2 DSPNs

Deterministic and stochastic Petri nets originated from an "exercise" in derivation of an embedded Markov chain technique for nets containing both deterministic and exponentially distributed timed transitions. The condition under which the derivation of the embedded Markov chain was found to be possible is:

$\forall M \in [M_0\rangle$, at most one deterministic timed transition is enabled in M.

This condition can be checked automatically by computing the mutual exclusion relation among deterministic timed transitions. Under this assumption it is possible to prove that the reachability graph is not affected by the presence of deterministic timing with respect to the one computed for the underlying untimed net. The technique for the numerical evaluation of these nets is described in detail in [41], and is more expensive than the usual Markovian analysis of GSPN models. An efficient implementation has been proposed by Lindeman for the steady state analysis [42]. Sensitivity analysis techniques have been recently proposed by Trivedi et al. [43].

2.3.3 Product form SPNs

The combinatorial explosion of the state space of Stochastic Petri Nets (SPN) is a well known problem that inhibits the exact solution of large SPNs and thus a broad use of this kind of formalism as a modelling tool. Certain SPN systems have been proven to be [44, 45, 46] characterized by a steady state probability distribution of their markings that can be factorized, yielding a so called *Product Form Solution* (PFS). The proposal due to Lazar and Robertazzi [46, 47] is based on the check for the presence of a special structure in the SPN reachability graph. The other PFS criterion proposed by Coleman, Henderson, Lucic and Taylor [44, 45, 48] is based on a structural analysis at the net level, and gives some conditions for recognizing the class of SPNs to which this criterion is applicable.

The structural conditions that a SPN has to satisfy are given in terms of input and output bags and can be resumed as follows: (1) no two transitions have the same input bag; (2) for every transition s there must exist a transition t such that the input bag of s corresponds to the output bag of t. For the class of SPNs identified by this criterion the factorization of the solution contains as many terms as there are places in the net.

Starting from this factorization two efficient algorithms have been derived [49, 50] for the computation of the PFS. Both have polynomial time and space complexities. They recall, respectively, the convolution algorithm derived by Buzen [51] for PFS Queueing Networks and the Mean Value Analysis algorithm of Reiser and Lavenberg [52]. Basic to the derivation of the convolution algorithm is a recursive expression of the normalization constant that is a generalization of that derived by Buzen for multiple class product form queueing networks with load independent service centers. The complexity (time and space) of this algorithm depends on the initial marking through a vector $\mathbf{K}$ (load vector) that contains as many terms as there are P-invariants in the net. Each component of $\mathbf{K}$ represents the number of tokens in the corresponding P-invariant. As in the case of multiclass queueing networks, using the convolution algorithm it is possible to compute several performance indices, such as transition throughputs and average number of tokens in places.

The Mean Value Analysis (MVA) algorithm is based on a recursive expression of certain performance indices (place throughput, average sojourn time of a token in a place, visit ratios and probability distributions of tokens in places). With respect to the analogous algorithm for multiclass queueing networks the MVA for SPNs requires several new concepts, one of the more interesting is a new kind of visit ratios. These visit ratios differ from the ones commonly used for QNs since they are not independent of the load of the net. In [50] a discussion may be found on the role played by these visit ratios in the MVA algorithm.

2.3.4 SWNs

When stochastic (exponential) timing is associated to transitions, the modeled system performance can be computed by analyzing the continuous time Markov chain (CTMC) isomorphic to the model RG. In case behavioral symmetries allow to aggregate the RG, if the aggregation of markings into equivalence classes satisfies the *lumpability conditions* [53] for Markov chains, then we can exploit the symmetries to lower the cost of performance analysis.

In order to get an SWN definition, two more functions have to be added to the WN definition, namely the transition priority and weight functions (denoted π_t and θ_t respectively). Transition instances may have different priority levels (identified by natural numbers). In particular a transition instance $[t, c]$ is *immediate* if $\pi_t(c) > 0$ while it is *timed* if $\pi_t(c) = 0$. The enabling rule must be modified to take priority into account: a transition instance $[t, c]$ cannot fire if there exists another enabled transition instance $[t', c']$ such that $\pi_t(c) < \pi_{t'}(c')$.

The transition weight function θ_t is used to define the firing rates of timed transition instances and the firing probabilities of immediate transition instances. Both the transition probability and the transition weights may depend on the particular transition instance, however they cannot depend on the actual identity of the color instance components, but only on the static subclass they belong to. This constraint is necessary to guarantee that all the objects belonging to the same static subclass behave homogeneously *by construction*.

The following proposition ensures that the marking aggregation induced by the *symbolic marking* notion satisfies the lumpability conditions:

Proposition 2 *Let $\{M_{i,k}\}$ be the set of ordinary markings belonging to symbolic marking $\mathcal{M}_i$, and let $M_{i,s.k} = s.M_{i,k}$ where $s \in \xi$. The probability of going from $M_{i,k}$ to $M_{j,q}$ is the same as the probability of going from $M_{i,s.k}$ to $M_{j,s.q}$, i.e.:*

$$q_{(i,k)(j,q)} = q_{(i,s.k)(j,s.q)}$$

Proposition 3 *The embedded Markov chain verifies the lumping condition [53], i.e.:*

$$\forall \mathcal{M}_j, \quad \forall M_{i,k}, \quad \forall s \in \xi, \qquad \sum_{M_{j,q} \in \mathcal{M}_j} q_{(i,k)(j,q)} = \sum_{M_{j,q} \in \mathcal{M}_j} q_{(i,s.k)(j,q)}$$

With respect to the aggregation scheme depicted in Figure 10, Proposition 2 states that similar dashed arcs exiting from equivalent markings, have the same associated rate.

The information contained in the SRG allows to automatically compute the transition rates between symbolic marking, as a consequence the lumped CTMC can be directly derived from the SRG without first computing the complete one.

When the lumped CTMC is ergodic, the steady state probability of symbolic markings can be computed. Abstract performance indices can thus be computed from this probability distribution. It is possible to prove that all the ordinary markings belonging to a given symbolic marking are equiprobable. Hence, from the probability distribution of symbolic markings it is possible to compute the probability distribution of any ordinary marking (because the number of ordinary markings in each symbolic marking can be easily computed).

3 Methodological aspects and modelling of parallel systems

We attempt an assessment of the timed Petri net modelling procedure by recalling examples of application in different domains. The analysis of the results obtained and the difficulty encountered in this cases highlight several characteristics of the considered modelling formalism and gives some suggestion regarding its effective use.

3.1 Modelling parallel architectures

In this section we shall describe by means of two examples the design process of some parallel architectures. Informally, the approach we are going to describe comprises the following steps:

1. definition of the abstraction level of the model,
2. decomposition of the system into its basic components,
3. construction of each component model,
4. composition of the submodels.

An alternative approach consists of building a very abstract model of the complete system, and then refining subnets in order to obtain a more detailed model.

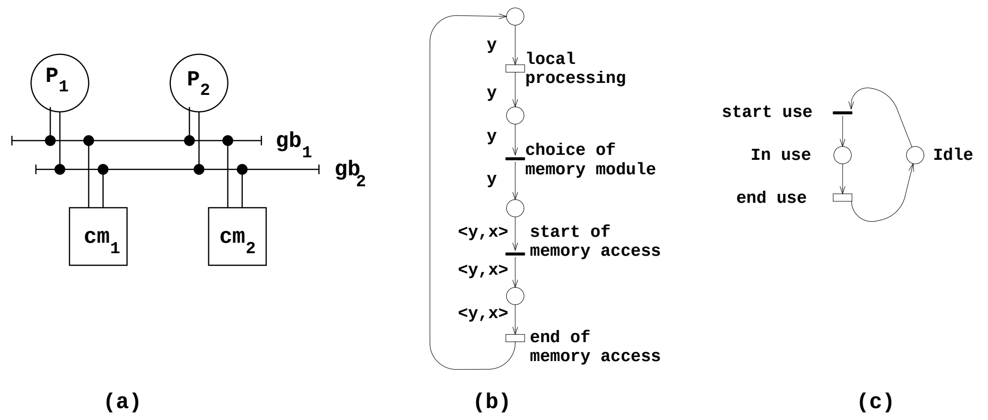

Figure 12: An example of shared memory architecture model composition

A shared memory architecture The first example model describes the shared memory parallel system composed of n processing units, m common memory modules and k global busses, depicted in figure 12.(a), for $n = m = k = 2$. The behavior of the generic processing unit (at a very abstract level) is the following: it performs some local processing, then it issues a request to access a given common memory module cm_i (we model the memory module choice probabilistically). When both a global bus and the required common memory module are available, the memory access starts; finally, when the memory access ends, the processing unit starts the cycle all over again. The model of the processing units behaviour is depicted in figure 12.(b). The memory

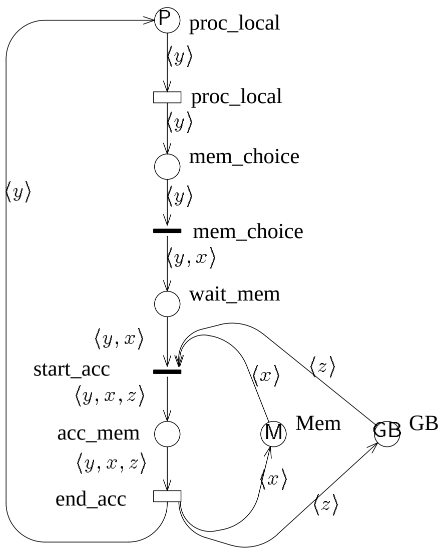

Figure 13: A SWN model of a shared memory architecture.

modules and global busses are represented by the subnet in figure 12.(c): they can be either in state *Idle* or *In use*.

The composition of the various system parts results in the net of figure 13: the composition operation amounts to *merging* the nodes in the subnets that represent the same event/state. The basic color classes needed are the class P of processing units, the class M of memories and the class GB of global busses.

The color structure of the net can be simplified using the decolorization rules explained in [32]: indeed, both the processors and global busses color classes are redundant[7] and the corresponding color components can be *decolorized*. The memories color component instead, cannot be decolorized because of the synchronization on transition *start_access*.

The model can be further refined to take into account the possible failure of the system components as shown in figure 14. Any component can fail in any state. After being repaired, a processing unit restarts in the "local processing" state while a common memory module/ global bus restarts in the "Idle" state. When a processor (or memory, or global bus) fails during a memory access, the associated memory and global bus (or processor and global bus, or memory and processor) involved in the same access are released. A possibly interesting measure of the system performance is for example the steady state average number of processors that are active, computed as

$$\sum_{j=1}^{n} j \ Prob(\#\text{of processing units working locally } = j)$$

A distributed memory architecture Let us work out another example model, representing a distributed memory system, i.e. a system in which processing units synchronize by means of *message passing*. We consider only machines with a regular structure. The abstraction level is chosen so as to allow the evaluation of the latencies

[7]Observe that the transitions *proc_local* and *end_access* must be defined as *infinite server* transitions after the color simplification

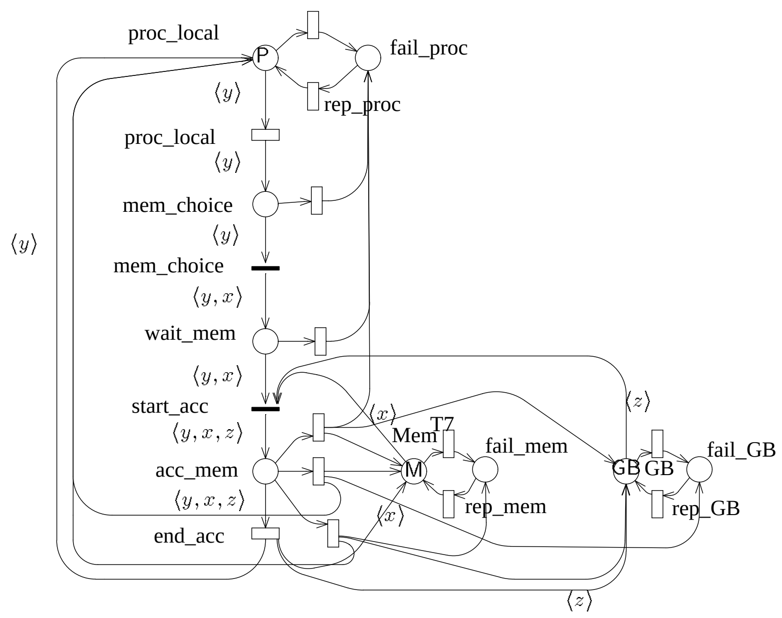

Figure 14: A model of shared memory architecture with failures

of packets traveling along the interconnection network. Latency depends on the various choices for the topology, the routing mechanism, and other factors. In [54] a model has been presented focusing on the features of the message passing distributed memory that are relevant for the performance of packet latency, and in particular: the topology, the routing and the flow control strategies. The basic idea is to *model separately each of these features and to compose them in a structured way in order to obtain a global model of the architecture*. For the sake of simplicity, in this section only the topology and routing aspects of the system will be modeled.

Modularity is achieved using the subnet refinement approach, but it may also require changes in the basic color classes, in the predicates and in the function definition.

Message passing parallel architectures comprise a set of processors (nodes) connected by an interconnection network and communicating by exchanging messages among them. Each processor has its own local memory and the only way in which it can exchange data with other processors is through the message passing mechanism.

In this example we focus on message passing machines using direct networks, i.e. machines in which the switching is performed by the processing nodes themselves rather than by dedicated switching units. For this family of architecures we can recognize the following key features:

1. **Processing features**: peculiar characteristics are the speed of the nodes and the availability or not of specialized hardware dedicated to the management of communications (*communication processor*). The latter feature is very important since the availability of a communication processor in each node frees the CPUs from the burden of the communication management.

2. **Network features**: **Topology**: the topology is the geometric organization of the interconnection network; **Routing**: routing provides the mechanism to allow the communication among non directly connected nodes; **Flow Control**: is the method used to regulate the traffic in the network.

Our idea is to model each of these features and then to compose them in a hierarchical way to obtain a global model of a parallel architecture. In the sequel we describe how to model nodes, topologies and routing (see [54] for a model comprising also flow control). We explain the methodology taking as an example k-ary n-cube topologies, in particular the so-called 2D toroidal mesh.

The node model We consider the model for a node as composed by a *computation unit* submodel and a *communication unit* submodel.

The computation unit is a device running the processes allocated to it. Processes may generate messages for the communication unit to transmit. A process can be viewed (in a very abstract way) as an entity generating both computation and communication requests. We model the part of the processing unit related to the generation and processing of messages, therefore the whole computation unit is modelled by a single timed transition (*Comp_proc*) with no input places and a single output place (*Comm_proc*). A firing of *Comp_proc* models the request for a message to be sent. The time between two successive firings, is the time between two successive requests of communication from the running processes. Observe that in a later refinement step, transition *Comp_proc* could be substituted by a complex subnet representing a more detailed model of the processes running on the node.

The communication unit is that part of a node that manages both the locally-generated messages and the messages coming from other nodes (which may need re-routing or not). It consists of buffers (in which messages or part of them are stored) and of communication channels.

Figure 15 depicts the model for a set of unconnected independent nodes having 4 communication links. Place *Comm_proc* represents the buffers of the communication unit. Timed transitions *Transmit_N, Transmit_S, Transmit_W, Transmit_E* represent the channels of each node. The time associated to these transitions is the time needed by a channel to transmit a data unit, independently of any delay due to resource contention. For the rest of this section we shall indicate with *Transmit_i* a generic transmit transition with $i \in \{E, W, N, S\}$.

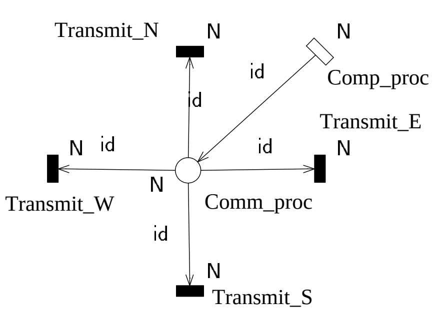

Figure 15: The colored node model

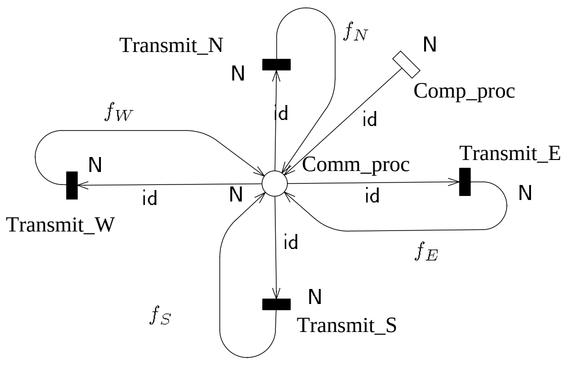

Figure 16: The model of a generic topology for a node with four links

The net in figure 15 represents a model that is parametric in the number of nodes but it is not in the number of channels associated with each node: the latter is fixed by the graphical representation of the net, and in this case it is equal to four. This choice has been made observing that the number of links in each node is usually an hardware characteristic (at least in the Transputer) and it is therefore fixed even for reconfigurable architectures. A model parametric in the number of links can of course be obtained by substituting in the models of Figure 15 the four *Transmit_i* transitions with a properly colored single transition. This is the most frequent choice a modeler has to face while using colored nets: trading graphical evidence for parameterization.

Modeling the topology The model of Figure 15 represents a set of unconnected nodes; to model a given topology we need to connect the channels in such a way that the resulting global model has the desired topology. A generic topology may simply be implemented by adding an arc from each of the four Transmit_i transitions to the node Comm_proc, as shown in Figure 16. Each arc i is labeled with the function f_i that computes for each node n the node connected to n through channel i. Observe how the meaning of place Comm_proc has now changed: a token of color n in this place may either be a request generated by transition Comp_proc of node n or a message coming from one of the nodes adjacent to n.

In order to connect each node to its adjacents we need a way to compute the set of adjacents of a node. This operation can be rather simple if the basic color class of the nodes has a well defined structure. As said above, we focus on k-ary, n-cube topologies. To model a n dimensional architecture with k objects per dimension, we define a basic ordered color class C containing k objects (the elements of a single dimension) and two associated functions: $P(a)$, the predecessor of color a in the class C, and $!a$, the successor of color a in C. The address of a node is an element in the Cartesian product $C^n = C \times \cdots \times C$. A generic node is then identified by an address of the type $< a_1, \ldots, a_n >$; its neighbours in the SWN formalism are the $2n$ nodes whose addresses are $< P(a_1), a_2, \ldots, a_n >, < a_1, P(a_2), \ldots, a_n >, \ldots, < a_1, a_2, \ldots, P(a_n) >$, $<!a_1, a_2, \ldots, a_n >, < a_1, !a_2, \ldots, a_n >, \ldots, < a_1, a_2, \ldots, !a_n >$ (with $!a_n = a_1$ and $P(a_1) = a_n$, by definition of ordered color class in SWN).

We can generalize this model allowing different numbers of nodes for each dimension. If k_i is the number of nodes in the i^{th} dimension, we define n basic ordered color classes $C_1, \ldots, C_n$ where C_i has k_i objects. The address of a node is now expressed using the Cartesian product $C_1 \times \cdots \times C_n$ and the neighbours of a node are identified as above. For example to model a 2D toroidal mesh with k_1 processors on the x-dimension and k_2 processors on the y-dimension, we define two basic colour classes P_x, P_y (containing respectively k_1 and k_2 objects) and in Figure 16 we define the color domain N as $N = P_x \times P_y$.

Figure 17 shows an example of topology consisting of a k-ary 2-cube parallel architecture (also known as 2D toroidal mesh). The color domain of place Comm_proc is consequently $P_x \times P_y$ and the functions labelling the arcs are used to express the connections in the hypothesis that a processor is identified by the token $< x, y >$.

From a modelling point of view the model of the topology is abstracting away two important aspects: indeed there is no explicit representation in the model of the contention that may arise if two messages require transmission on the same channel, and of the choice of the channel on which a message should be transmitted. If the goal of the model is to provide performance estimates of the behaviour of real architectures

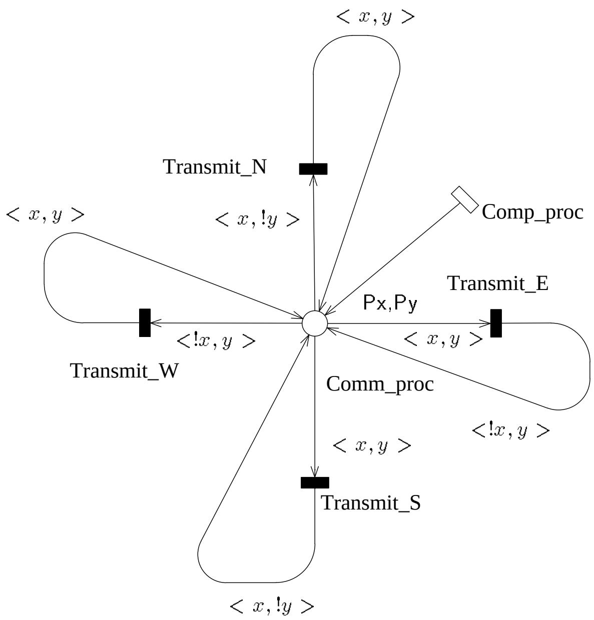

Figure 17: 2D toroidal mesh

these two aspects need to be taken into account. In the following section we explain how to model the choice of the channel to be used for a transmission (routing strategy).

Modelling routing strategies The models depicted in Figure 16 and 17 do not specify the way in which a message can choose a channel among the four available; more generally the way in which a path in the network is chosen to reach a destination node (i.e. the routing strategy) is not represented.

Routing strategies can be classified into *deterministic, oblivious*, or *adaptive*. With deterministic routing the path a message follows depends only on its source and destination nodes, while with the other two strategies the path may vary from time to time.

In order to model the routing strategy we need to include additional information in our model. We concentrate on deterministic routing and therefore we need to change the identification of the message to include information not only on its current position, but also on its destination. We change the color classes associated with transitions Comp_proc and Transmit_i and place Comm_proc from N to $N \times N$, where N is the basic color class representing nodes.

The routing function is modelled as a choice out of place Comm_proc, as shown in Figure 18, where each subnet Comm_proc - Transmit_i (place - timed transition) has been substituted by the subnet Comm_proc - direction_i - Chan_handler_i - Transmit_i (place - immediate transition - place - timed transition). Observe that this implies a major change in the role of place Comm_proc, from "buffer" for the messages of the node to decision place (indeed the mean number of tokens in the place is always zero). The set of messages of each node is now split in four buffers, the Chan_handler_i places.

When a message reaches its destination, the corresponding token should be removed

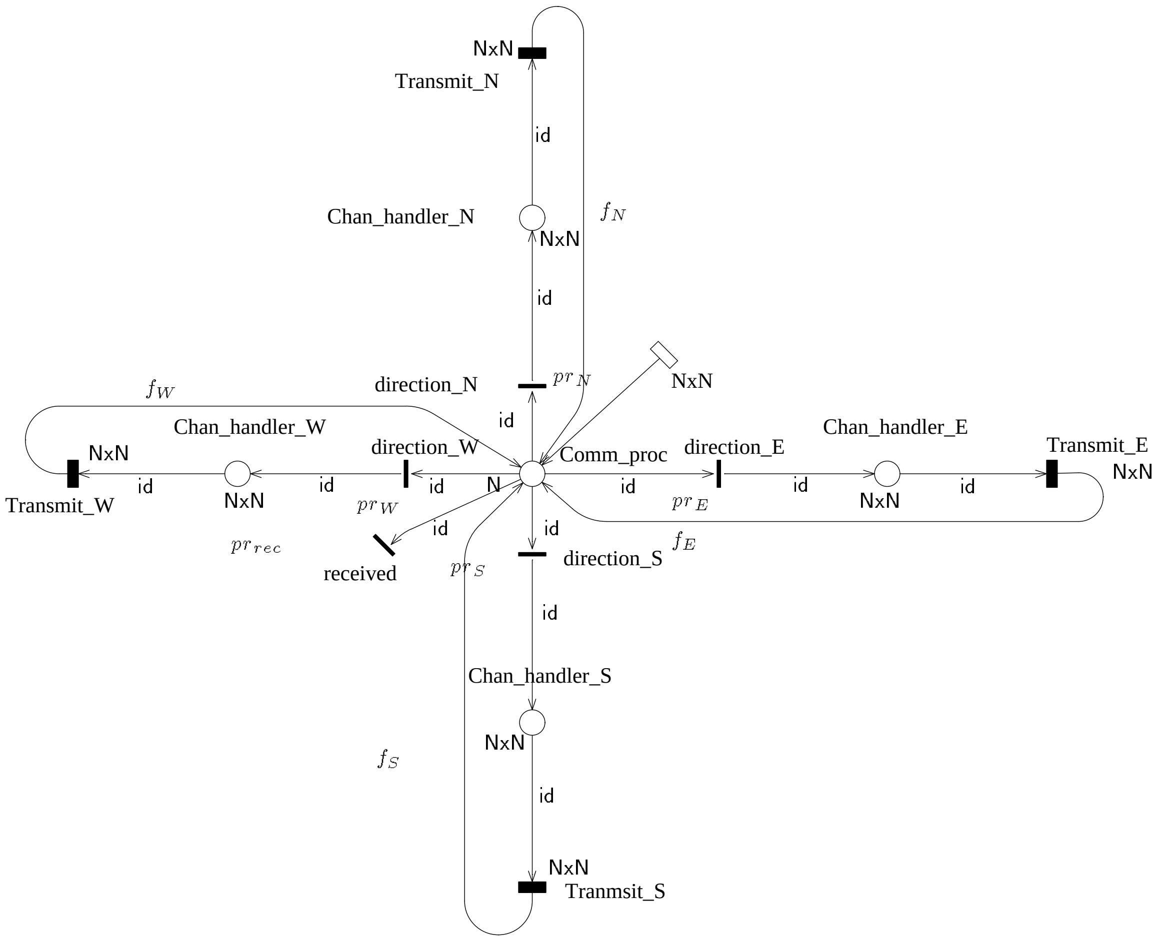

Figure 18: Generic routing

from place Comm_proc (consumed by the destination). This can be modelled by adding an output transition to place Comm_proc, called received in the net of Figure 18, with no output nodes (absorbing transition). The routing algorithm can now be modelled simply using the predicates pr_i and pr_{rec} associated with immediate transitions.

In case of a deterministic routing one and only one of the five transitions may be enabled for a given message since by definition of deterministic routing only one channel can be used at any step. That is to say for the same color instance (pair <cur, dest>) the five transitions are mutually exclusive. From a Petri net point of view the choice of a routing algorithm is basically a transformation of what appears graphically as a free choice conflict into a persistent subnet using predicates.

As an example we describe how to model a particular routing strategy for k-ary, n-cube architectures known as *e-cube routing* starting from the generic routing model of Figure 18. The e-cube algorithm is a deterministic routing such that at each step the destination address of the message is examined. For the i^{th} digit of the destination address a route in the i^{th} dimension to the proper coordinate is chosen. For example if a message with destination (2,3) is currently at node (0,1), it will first be routed to

(2,1) (thus following the first dimension), and then to (2,3) (in the second dimension). For 2D toroidal mesh, given the present position of a packet, the routing strategy will choose the x-direction first, until the destination column is reached, and subsequently the y-direction, until the destination node is reached. The algorithm is encoded in the predicates of transitions direction_N, direction_S, direction_E and direction_W.

3.2 Modelling distributed software

Petri net based models are very well suited to represent concurrent programs; it is quite natural to represent concurrent programs basic statements at various level of abstraction and compose them into program models by means of Petri nets. At higher abstraction levels long sections of code without communication statements can be represented by a simple transition, some choice that in the program are deterministically performed on the actual variable values can be converted in non-deterministic choices with a probability distribution assigned to a set of possible alternatives. The maximum level of detail should include the representation of all the program variable values but in general this level of abstraction leads to very complex models without giving substantially useful information. An intermediate abstraction level could include the detailed representation of only some variables crucial to the program behavior and whose values depend only on the particular execution chosen and not on the actual value of the input variables.

A methodology that automatically builds a GSPN model starting from a detailed representation of a CSP-like program structure has been proposed in [55]. The procedure consists of three steps: the derivation of a "process scheme" directly from the program, the translation of each process scheme into a GSPN, and, finally, the composition of the individual GSPN nets into the GSPN model of the whole program. If the program contains several instances of the same process the resulting GSPN model has several identical subnets that can be folded into a single one leading to a more compact HLPN model. At the end of the next section we'll briefly describe a scheme for the compact representation of a concurrent program mapped on a parallel architecture using a HLPN model.

As an example consider the CSP-like program depicted in Figure 19. The program represents a monitor system that controls three different devices *S1*, *S2* and *S3*. An anomaly in the i-th device corresponds to a message being sent from Si to process *Acquire*, through channel Sig_i. Process *Acquire* listens to the three channels Sig_i ($i = 1, 2, 3$) and forwards any received message to process *Elaborate*. Upon reception of a message both processes *Acquire* and *Elaborate* activate two children (*ReadSi* and *AskSi* respectively) with the special task of getting the current values of the other signals. After a given number of signals is received, some statistics are computed and the monitor system starts again. The translation of each process scheme in isolation in shown in Figure 20 where process *Init* is omitted and only generic Si, *AskSi* and *ReadSi* are represented. Timed transitions represent parallel activations (transitions PAR) or activities that requires the use of CPU resources (transitions check_Si, AcqSi, comp). Input and output statements are represented by immediate transitions whose names end with "?" (input statement) and "!" (output statement). Notice that at this stage of the translation, communications between processes are represented by means of immediate transitions.

```
Init :
PAR
    P0(Sig1, Sig2, Sig3)
    S1(Sig1)
    S2(Sig2)
    S3(Sig3)
EndInit
```

```
Si :
while true
    check_anomaly(val_Si)
    Sig_i!val_Si
EndSi
```

```
P0 :
while true
    PAR
        Acquire(Sig1, Sig2, Sig3, AcEl1, AcEl2, AcEl3, RdAsk1, RdAsk2, RdAsk3)
        Elaborate(AcEl1, AcEl2, AcEl3, RdAsk1, RdAsk2, RdAsk3)
EndP0
```

```
Acquire : <acquiring process>
i:= 20
while (i> 0)
    seq
    ALT
        (Sig1 ? val_S1)
            AcEl1!val_S1
            PAR_11
                Read_S2(RdAs1)
                Read_S3(RdAs3)
        (Sig2 ? val_S2)
            AcEl2!val_S2
            PAR_12
                Read_S1(RdAs1)
                Read_S3(RdAs3)
        (Sig3 ? val_S3)
            AcEl3!val_S3
            PAR_13
                Read_S1(RdAs1)
                Read_S2(RdAs2)
    i:=i−1
EndAcquire
```

```
Elaborate : < signal processing >
j:= 20
while (j > 0)
    seq
    ALT
        (AcEl1 ? val_S1)
            PAR_21
                Ask_S2(RdAs2)
                Ask_S3(RdAs3)
        (AcEl2 ? val_S2)
            PAR_22
                Ask_S1(RdAs1)
                Ask_S3(RdAs3)
        (AcEl3 ? val_S3)
            PAR_23
                Ask_S1(RdAs1)
                Ask_S2(RdAs2)
    datum := process(val_S1, val_S2, val_S3)
    j := j−1
EndElaborate
```

```
Read_Si :
    < acquire val_Si >
    (RdAs_i ! val_Si)
EndRead_Si
```

```
Ask_Si :
    (RdAs_i ? val_Si)
EndAsk_Si
```

Figure 19: Code of the monitoring system.

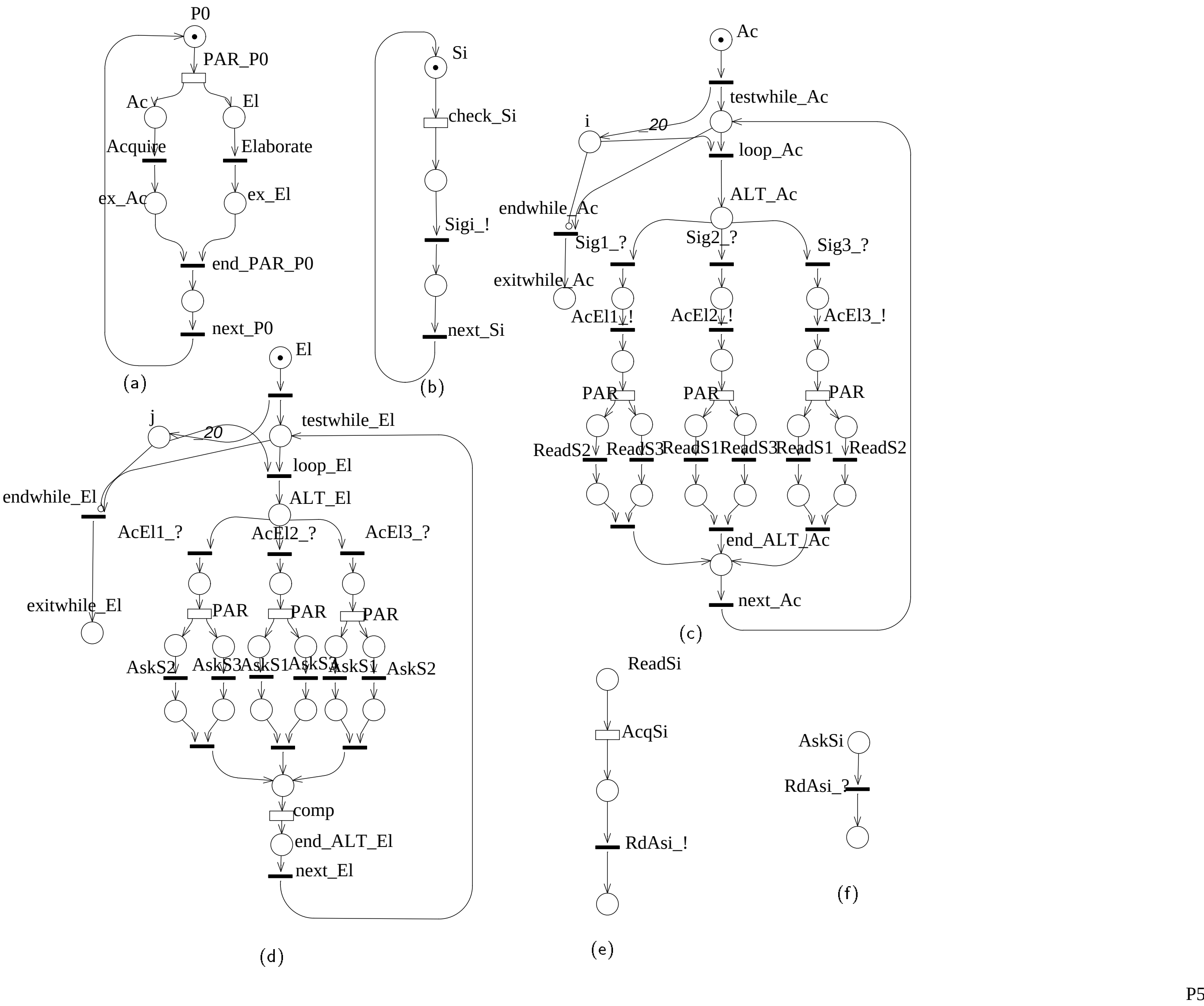

Figure 20: Translation of each process.

Process activations are modelled using immediate transitions labelled with the corresponding process names (see for example transitions Acquire and Elaborate in Figure 20.(a)).

The GSPN model of the whole program is obtained composing all the process schemes and it is shown in Figure 21. Pairs of corresponding send/receive transitions of different processes are folded together into a single immediate transition which is then refined into a sequence of one immediate transition and one timed transition: the first one represents the synchronization between the two processes, the second one represents the time needed for moving the message from one process to the other. Immediate transitions that represent processes activations are substituted by the corresponding processes schemes: this substitution is obtained by merging the input place of the immediate transition with the first place of the corresponding process scheme.

The graphical visualization of the concurrent program may turn out to be extremely useful since its animation can be used for debugging purposes and for identifying peculiar behavior patterns that may suggest restructurings of the program.

The structural analysis techniques available for GSPN models allow the automatic detection of some basic properties of the concurrent program. An example of information that can be derived using the net P-invariants together with the initial marking is about mutually exclusive code segments. Mutually exclusive processes can be put on the same processors without loss of performance. In our program for example process *Elaborate* and processes *ReadSi* are never active together. In this case the mutual exclusion relation could be easily derived by observing the parallel activation of processes, but in general it could be difficult to discover this relation using only intuition.

The use of a formalism that allows to incorporate time specifications in the model allows quantitative studies on the behavior of the program without the need of the development of new models. Timing information is assigned to transitions representing actions that take time (computation, communication). The throughputs of transitions representing communication statements are used to compute the amount of communication messages exchanged by two processes. For example let us consider communication between process *S1* and process *Acquire*. Transition com_Sig1 models this communication, and the communication cost is simply obtained considering the throughput of this transition (communication costs between all pairs of communicating processes can be computed in the same way).

3.3 Matching hardware and software models

Given a parallel program, various mapping strategies can be applied. At this point if one wants to compare the efficiency of the different mappings by using models, two further PN layers should be generated, one for the architecture and one for the mapping; these layers are superimposed to the GSPN model of the program to obtain a complete model on which some performance evaluation of the mapped program can be performed.

Modelling the architecture Let us show how the concurrent program previously described can be mapped on a 2×2 mesh topology. The hardware layer is quite simple and consists in generating a resource place for each processor and a resource place for each physical communication link connecting the processors.

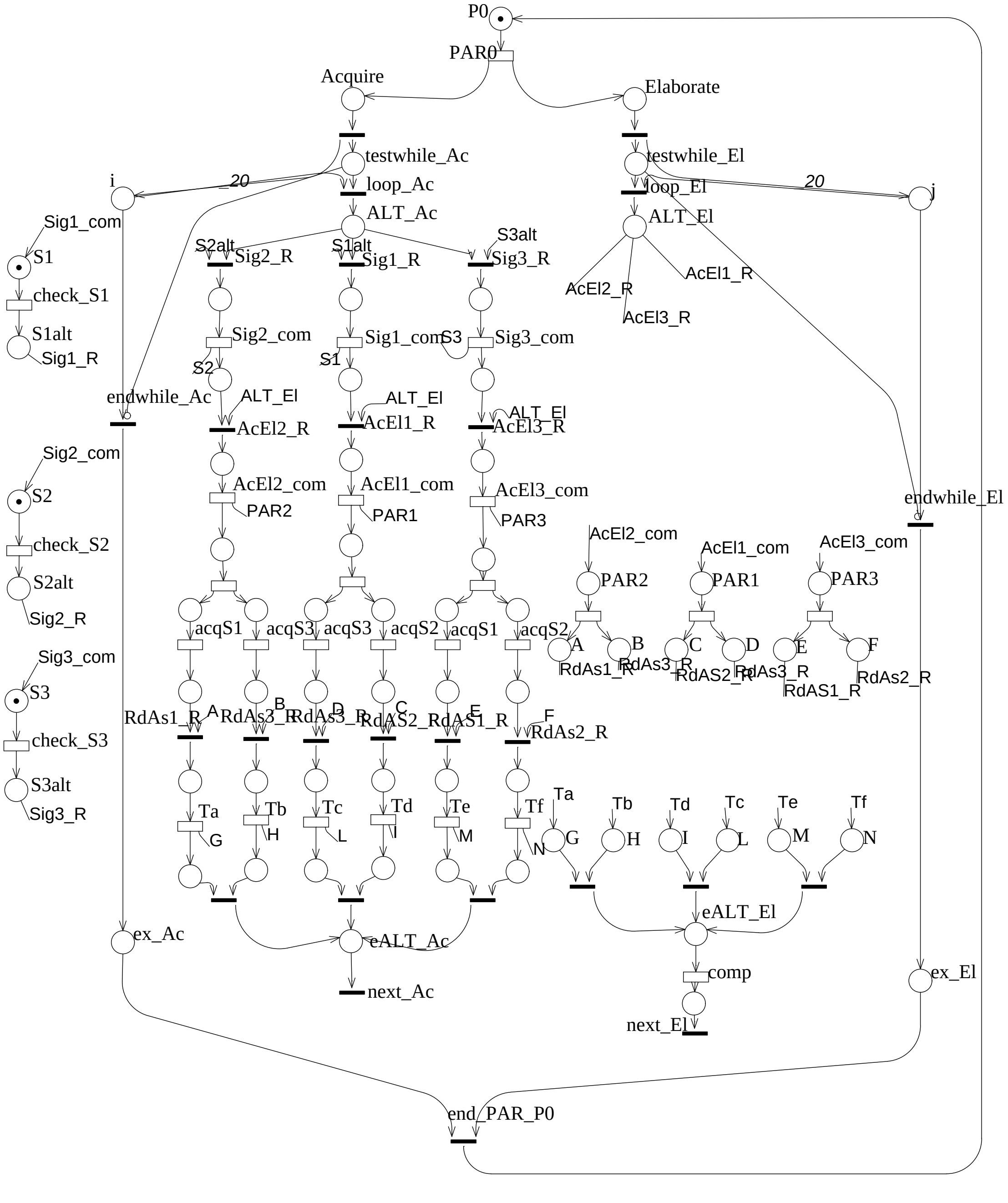

Figure 21: The complete GSPN representation of the software system of Fig. 19.

As long as each pair of communicating processes are placed onto directly connected processors this is enough, but since in general this is not true, a further layer representing the routing system should be modelled. The 2×2 topology is shown in Figure 22.(a) and the corresponding model is shown in Figure 22.(b) where places Pi represent the four processors and places Lij represent the links that connect the i-th and the j-th processors.

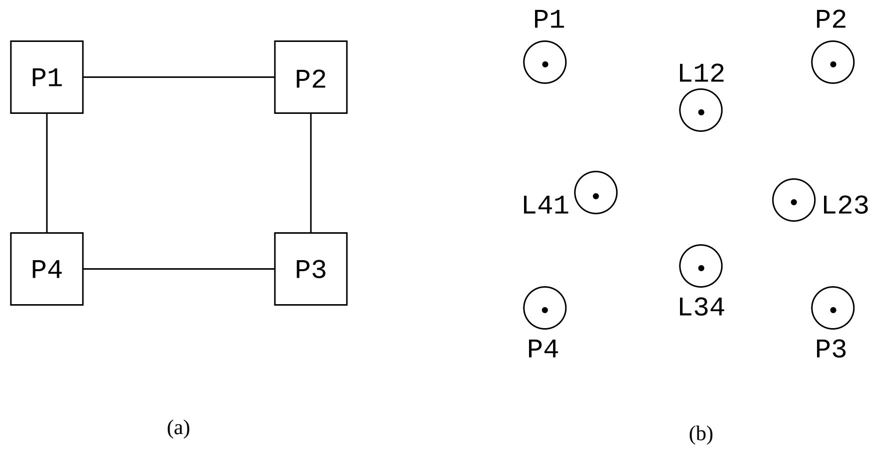

Figure 22: The hardware level.

Mapping the program model onto the architecture model The mapping layer is quite simple and consists of arcs going from the hardware resource places to the transitions representing activities that need those resources and reverse arcs representing the release of the resource at the end of the activity. Transitions representing computation will require the CPU on which the corresponding process has been mapped. Communications among processes mapped on different CPUs need the link that connects the two processors while communications occurring between processes placed on the same processor need the CPU rather than a communication link. Basic schemes for processor and link acquisitions are shown in Figure 23. Figure 23(a) represents

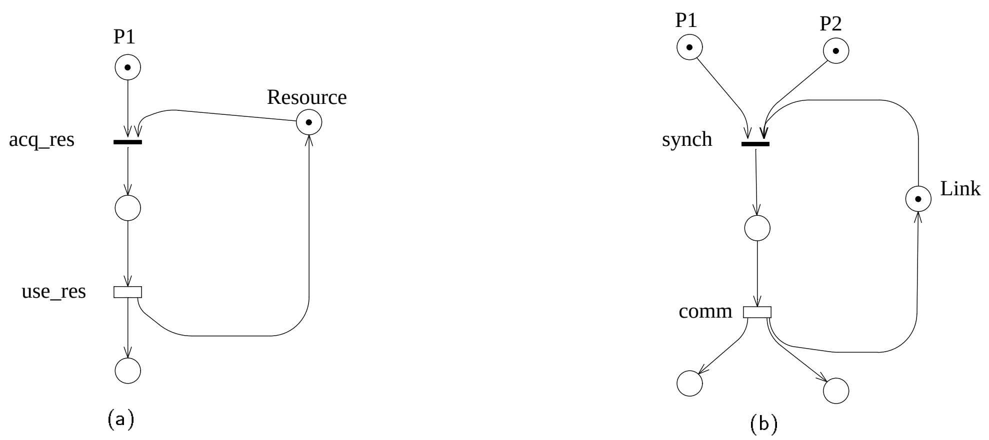

Figure 23: Mapping schemes.

the acquisition of a processor: notice that the resource consumption is modelled by a sequence of one immediate transition that represent the resource acquisition and one timed transition that models its effective use. Without the immediate transition acq_res we would model an erroneous situation in which the resource is always available (place Resource is always marked). Figure 23(b) is related to the acquisition of a communication link. A communication between two processes can be performed when both processes are ready to communicate and the link that connect the processors on which they are mapped is available. This is represented by the immediate transition synch that can fire only when its three input places (that correspond to all the mentioned preconditions) are marked. Observe that we are assuming that communication links are capable to perform communications without the use of CPU resources. If this assumption is not valid the mapping of communication statements is more complex because we need to acquire both the processors for all the duration of the communication. If the link is capable to perform communications we need to acquire the processors only for the duration of the initial start up phase and then the link will perform the effective communication but, for simplicity, we have not modelled the start up phase explicitly.

Modifying all the transitions that model computations and communications in the GSPN model of the software according to this scheme we can build the complete GSPN model comprising the program, the architecture and the mapping information. Examples of performance indices that can now be computed with respect to a given mapping are the CPU utilization or the probability of waiting for synchronization on a communication statement.

Mapping HLPN models Let us consider a parallel program composed of several instances of the same process. In Figure 24.(a) three subnets corresponding to three different instances of the same process P are shown. The folding of these subnets leads

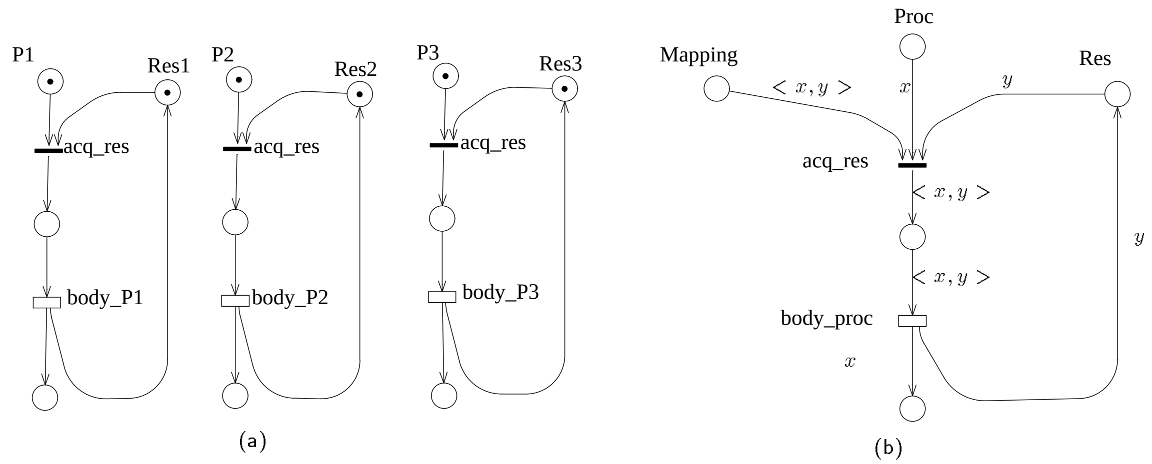

Figure 24: Mapping scheme for a colored net.

to the single net depicted in Figure 24.(b). Two basic colour classes $Proc = \{p_1, p_1, p_3\}$ and $Res = \{r_1, r_2, r_3\}$ are necessary in order to distinguish among different process instances and different resources. Notice that in this case we have the same number of processes instances and the same number of resources but in general these two numbers a can be different. Place Mapping models the mapping of the instances onto the resources: its initial marking specify where each process instance is mapped. The i-th instance of process P cau use the j-th resource only if place Mapping contains the token $< p_i, r_j >$. In this case the initial marking of place Mapping is $< p_1, r_1 > + < p_2, r_2 > + < p_3, r_3 >$.

3.4 Dealing with large models

Largeness is a bad characteristics of TPN models that may hamper their practical utilization. We may identify at least three levels of degree at which largeness may appear as a practical problem (in increasing order of difficulty):

1. simple TPN representation with huge state space
2. large TPN representation with exploitable symmetries
3. large, irregular, unstructured TPN representations

Case number 1 creates problem for Markovian exact numerical analysis. It creates no problem for operational bound computation and for discrete event simulation. The formalism is still useful to understand the problem even if the numerical computation of performance measures by Markovian analysis becomes expensive.

Case number 2 usually implies also the problems of case number 1. Coloured nets may be the correct answer to the problem of handling the complexity of the graphic representation. Operational bound computation and discrete event simulation may still apply without major problems. Behavioral properties derived from parametric structural analysis of the underlying WN may be valuable for the understanding and validation of the model.

Case number 3 represents the worst case, where even coloured nets do not help. The graphic representation of a complex system may be counter-productive in case no regularity in the structure may be identified to present the pictures in an orderly fashion. These are the models that one should refuse to draw. The first objective in a practical case of this type should be to introduce some reasonable simplifying hypothesis to reduce the model complexity. Hierarchical decomposition techniques derived from the application domain may be helpful in such a simplification of the model.

Different decomposition techniques have been proposed for Petri net models including subnet substitution, views sharing places and/or transitions, and layering. Using such techniques one can usually control the graphic complexity of real models. A problem that is still open to research is the association of such graphic partitioning techniques with analysis techniques (both qualitative and quantitative) based on the same graphic decomposition. If the analysis or simulation of a hierarchically decomposed model requires its "projection" on a complete, detailed, flat model, the search for efficient implementation of analysis algorithms is hopeless.

3.5 Tool support

Timed Petri nets must be supported by a proper modelling and analysis environment in order to be usable in practical situations. The manual derivation of results from the graphic specification of a model, although sometimes feasable for small nets, is absolutely out of question since it is too error prone. This characteristics emphasizes the importance of the implementation of a package supporting the theoretical results.

The availability of *GreatSPN*, a package developed at the University of Torino since 1986, is one of the main reasons for the popularity of the GSPN modelling formalisms more than the actual virtues of the formalism itself.

The underlying idea for the organization of *GreatSPN* is to provide a unified modelling framework in which the modeller can create and edit graphically his TPN models, apply several structural analysis techniques on them to perform a behavioural study/validation, and then compute performance using numerical or simulation techniques. The interaction between the modeller and the tool is completely based on the graphic interface. Behavioural and performance results are displayed in graphic form on the model representation.

At present the package (version 1.6) is functionally complete for the analysis of (non coloured) GSPN models. Prototype implementations of (symbolic) reachability graph and simulation for SWNs are under development. Nets with non exponential timed transitions are treated only by simulation.

Modules that are planned to be introduced in future releases of *GreatSPN* include: computation of bounds by linear programming techniques for non coloured as well as well-formed timed nets; efficient DSPN solution numerical solution techniques; structural reduction for P/T and well-formed nets.

4 Relationships to other modelling methods

In this section we attempt a comparison with the queueing networks (QNs) formalism. In particular BCMP type QNs will be mainly considered. Other formalisms will not be considered due to our limited experience with them. We believe that more general comparisons should be developed as a result of the discussion in this workshop.

4.1 Analogies and differences with queueing networks

The availability of operational analysis results for TPNs may suggest analogies with QNs. In a broad sense one could compare tokens in TPNs with customers in QNs, places with queues, and transitions with servers. Such a mapping is not mathematically precise, however, due to the many differences that exist between the two formalisms.

From the analysis of many TPN models proposed in the literature one may easily realize that tokens may be used not only to represent customers wondering around attempting to obtain services, but also to represent available or busy resources, different states of servers, etc. The real difference between TPNs and QNs is that, while the latter specifies the semantics of the components of a model at the level of the modelling formalism, the former does not. TPNs (like PNs) are a general purpose formalism in which the basic building blocks (places, transitions, arcs, tokens) may assume different semantics not only in different models but also within a single model. Tokens in a place represent a portion of distributed state variable that characterize the model. In the same model one place may represent all customers waiting to receive service (i.e. a queue) while another place may represent one state of a server; the tokens contained in them have thus different semantics.

Of course nothing prevents the modeller to add restrictions to the use of places to model systems belonging to a given application domain, thus introducing a precise semantics at the formalism level, just as in the case of QNs. This is however not required to define and use TPNs. This lack of predefined application semantics is an advantage in the sense that the TPN formalism is open to new application domains. One can always derive a TPN model by just finding a state representation for the system in terms of natural numbers. The fact that the number of tokens in a place represents the lenght of a queue is an important information for the modeller to interpret the analysis results but is absolutely irrelevant for their computation.

Synchronization is a basic construct of the PN formalism, obtained by connecting more than one input arc to a transition. All types of synchronization can be expressed in terms of this basic construct. Being a "lower level" formalism with respect to QNs in terms of system semantics embedded into the modelling formalism, a PN model is able to express any kind of synchronization (rendez-vous, semaphores, monitors, etc.) as more or less complex subnets containing such basic construct. On the other hand, QNs need to enumerate all variations of synchronization mechanisms as different primitives to preserve their modelling semantics (passive resources, blocking, fork-join, split-merge, etc.).

The basic PN formalism leaves several aspects of the model behaviour indeterminate. Several different interpretations may be added to complete the behavioural specification of the net that are not contradictory with the basic, untimed specification. This degree of freedom can be exploited to the benefit of the modeller in order to tune the abstraction level of the model to his needs. For instance in case of conflicting transitions several queueing disciplines may be adopted interchangeably without changing the qualitative behaviour of the net model, the main difference being the possibility of preemption of a started service or not. In some cases the specification of a policy may be irrelevant in the sense that it may not change the performance indexes; in other cases it may be relevant (in the sense of affecting the performance indexes) and may be either represented explicitly by means of a subnet, or added as a textual specification to the nondeterministic net representation. The analysis algorithm must of course take into account not only the graphic structure of the model but also its associated interpretation. The use of sophisticated interpretations for abstract TPN models in which details are left undetermined is the way in which modelling abstraction can be efficiently exploited.

Another interesting point that must be compared is the possibility of model parametrization. QN models are naturally parametric in terms of the populations of customers and the service times and routing probabilities. In the field of PNs, traditionally two different structures have been considered and studied: nets and system. The difference between the two is that the former does not include the initial marking specification, while the latter does. In [5] an "intermediate" definition called "Petri net model" was introduced, in which part of the marking may be fix while another part may be kept parametric. The interest of such a new definition is that usually PNs have non monotonic behaviour with respect to changes in the initial marking, so that the same net structure may exhibit wrong behaviours in case of improper definition of the initial marking. The characteristics of "proper" initial markings depend on the individual model, and are not easily generalized to net classes. With this idea of model class with an associated set of possible proper initial markings that preserve the characteristics of the model, the TPN formalism has the same parametrization possibilities of QNs.

4.2 Merging by flow or delay equivalent transitions

Hierarchical modelling is often the solution for the study of complex problems yielding very large models. Hierarchical modelling corresponds to the "divide and conquer" approach in which a complex model is seen as the composition of modules corresponding to physical or logical components of the real system. A hierarchical model is thus seen as a set of representations of a complex system made at different levels of detail. In hierarchical modelling the basic idea is that of identifying certain submodels that can be studied in isolation. A characterization of the overall behaviour of each submodel is then included in the higher level model by means of an equivalent server or transition.

The method has been widely discussed in the framework of queueing networks for the solution of non-product form queueing networks [56, 57] and relies on decomposition results that show that approximate solutions of large Markovian models turn out to be quite accurate when the subsystems to be replaced with equivalent servers weakly interact with the rest of the system [58].

The same idea can be conveniently used with GSPN models by isolating subnets that can be studied in isolation. In its simplest setting, subnets with a single input and a single output places are identified within the model and studied in isolation by removing the rest of the net and by superimposing the input and output places. Operating in this manner, and choosing a proper initial marking for the common input/output place, a bounded *GSPN* is obtained that is solved by computing the common input/output place throughput (i.e., the sum of the throughputs of all the transitions that have such place within their input set). These throughputs conditioned on the initial marking of the common input/output place represent the speed at which tokens deposited in such input/output place are processed by the subnet and are used as its overall characterization. This representation allows the replacement of the subnet in the higher level model by means of a delay equivalent transition with a marking dependent firing rate. Examples of the application of this technique for the analysis of complex problems represented by large *GSPN*s can be found in [59, 17, 60, 61].

The characterization of the behaviour of a subnet by means of its processing speed conditioned on the number of activities simultaneously carried on, may sometimes be too detailed and thus too expensive to implement. A different approach is that of providing a characterization that is based on the distribution of the time spent by the tokens within the subnet [62, 63] that is very often non-exponential. In order to cast this approach within the context of *GSPN* models, non-exponential distributions can be represented using their *phase type* approximation [64] and thus by substituting the subnet with a mini-net exhibiting a delay distribution whose first few moments match those of the subnet's distribution.

5 Attempt of summary evaluation

Petri net based models offer several advantages with respect to other formalisms. First, they can be understood and used as generalizations of Jackson type queueing models, with the introduction of a basic synchronization primitive. The operational analysis appraoch for the study of timed Petri net models stresses such similarity. On the other hand, Petri nets are inherently non deterministic, so that different interpretations may be added to the same untimed model without changing the logic of its qualitative behaviour and changing only its performance. The level of interpretation can then be chosen in a fairly free way in order to meet different modelling requirements.

The introduction of colours (token identities) in a restricted way, such as the one proposed for Well-Formed nets, allows an efficient exploitation of model symmetries for reducing at the same time both the graphic complexity of the model and the computational complexity of its analysis. Unlike the case of multiclass queueing networks, the introduction of colours in Petri nets restricted to symmetric case allows the application of more efficient analysis and simulation algorithms.

The graphical structure of a Petri net model can be studied as a mathematical object per-se, producing results that hold true for any possible dynamics of the model itself. Results of this structural analysis can be effectively used to improve the efficiency of the implementation Markovian analysis and discrete event (sequential as well as parallel) simulation with respect to brute force approaches. In the case of computation of performance bounds, it also produces directly performance results at low computational cost for large state spaces, the method being totally independent of the number of reachable states.

Although Petri nets have a simple graphic representation, one must notice that the definition of appropriate models that represent a system in a accurate way and that can be analyzed with reasonable computational effort usually requires a deep knowledge of the formalism by the modeller. The initial claim that the graphic representation would have facilitated the use of the formalism by non specialists of performance modelling seems now utopistic. This consideration by no means diminishes the interest in the research and development of the formalism as a tool for performance evaluation profesionals.

References

[1] G.W. Brams. *Réseaux de Petri: Théorie et Pratique. T.1. théorie et analyse.* Masson, Paris, France, 1983. in French.

[2] M. Silva. *Las Redes de Petri en la Automatica y la Informatica.* Ed. AC, Madrid, Spain, 1985. in Spanish.

[3] T. Murata. Petri nets: properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[4] G. Chiola, S. Donatelli, and G. Franceschinis. Priorities, inhibitor arcs, and concurrency in P/T nets. In *Proc.* 12^{th} *International Conference on Application and Theory of Petri Nets*, Aarhus, Denmark, June 1991.

[5] G. Chiola, S. Donatelli, and G. Franceschinis. On parametric P/T nets and their modelling power. In *Proc.* 12^{th} *International Conference on Application and Theory of Petri Nets*, Aarhus, Denmark, June 1991.

[6] K. Jensen. Coloured Petri nets and the invariant method. *Theoretical Computer Science*, 14:317–336, 1981.

[7] H. J. Genrich and K. Lautenbach. System modelling with high-level Petri nets. *Theoretical Computer Science*, 13:109–136, 1981.

[8] C.A. Petri. Communication with automata. Technical Report RADC-TR-65-377, Rome Air Dev. Center, New York, NY, 1966. Tech. Rep. RADC-TR-65-377.

[9] P. M. Merlin and D. J. Farber. Recoverability of communication protocols: Implications of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, September 1976.

[10] J. D. Noe and G. J. Nutt. Macro e-nets representation of parallel systems. *IEEE Transactions on Computers*, 31(9):718–727, August 1973.

[11] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, MIT, Cambridge, MA, 1974. Ph.D. Thesis.

[12] M.A. Holliday and M.K. Vernon. A generalized timed Petri net model for performance analysis. In *Proc. Int. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.

[13] G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, 19(2), February 1993.

[14] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer-Verlag, Berlin, Germany, 1980.

[15] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On well-formed coloured nets and their symbolic reachability graph. In *Proc.* 11^{th} *International Conference on Application and Theory of Petri Nets*, Paris, France, June 1990. Reprinted in *High-Level Petri Nets. Theory and Application*, K. Jensen and G. Rozenberg (editors), Springer Verlag, 1991.

[16] G. Chiola, J. Campos, J.M. Colom, and M. Silva. Operational analysis of timed Petri nets and application to the computation of performance bounds. submitted for publication, 1993.

[17] M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems*. MIT Press, Cambridge, USA, 1986.

[18] F. Baccelli, G. Cohen, and B. Gaujal. Recursive equations and basic properties of timed Petri nets. *Discrete Event Dynamic Systems: Theory and Applications*, 1:415–439, 1992.

[19] G. Chiola. On the structural and behavioural characterization of P/T nets. In *Proc. 5th Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.

[20] P. J. Denning and J. P. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys*, 10(3):225–261, September 1978.

[21] G. Ciardo, J. Muppala, and K.S. Trivedi. On the solution of GSPN reward models. *Performance Evaluation*.

[22] G. Berthelot. Transformations and decompositions of nets. In W. Brawer, W. Reisig, and G. Rozenberg, editors, *Advances on Petri Nets '86 - Part I*, volume 254 of *LNCS*, pages 359–376. Springer Verlag, Bad Honnef, West Germany, February 1987.

[23] E. Best. Structure theory of Petri nets: the free choice hiatus. In W. Brawer, W. Reisig, and G. Rozenberg, editors, *Advances on Petri Nets '86 - Part I*, volume 254 of *LNCS*, pages 168–205. Springer Verlag, Bad Honnef, West Germany, February 1987.

[24] J. Esparza and M. Silva. On the analysis and synthesis of free choice systems. In G. Rozenberg, editor, *Advances on Petri Nets '90*, volume 483 of *LNCS*, pages 243–286. Springer Verlag, 1991.

[25] S. Haddad. *Une Categorie Regulier de Reseau de Petri de Haut Niveau: Definition, Proprietes et Reductions.* PhD thesis, Lab. MASI, Universite P. et M. Curie (Paris 6), Paris, France, Oct 1987. These de Doctorat, RR87/197 (in French).

[26] S. Haddad. Generalization of reduction theory to coloured nets. In *Proc. 9^{th} Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

[27] G. Chiola and C. Anglano. Linear programming performance bounds for symmetric coloured nets. Technical report, Dipartimento di Informatica, University of Torino, February 1993.

[28] P. Huber, A.M. Jensen, L.O. Jepsen, and K. Jensen. Towards reachability trees for high-level Petri nets. In G. Rozenberg, editor, *Advances on Petri Nets '84*, volume 188 of *LNCS*, pages 215–233. Springer Verlag, 1984.

[29] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 1993. accepted for publication.

[30] G. Chiola, G. Franceschinis, and R. Gaeta. A symbolic simulation mechanism for well-formed coloured Petri nets. In *Proc. 25th SCS Annual Simulation Symposium*, Orlando, Florida, April 1992.

[31] G. Chiola and R. Gaeta. Efficient simulation of parallel architectures exploiting symmetric well-formed Petri net models. In C.E. Knadler and H. Vakilzadian, editors, *SCS Western Simulation Multiconference '93*, volume 25 of *Simulation Series*, pages 285–290, San Diego, California, January 1993. Society for Computer Simulation.

[32] G. Chiola and G. Franceschinis. A structural colour simplification in Well-Formed coloured nets. In *Proc. 4th Intern. Workshop on Petri Nets and Performance Models*, pages 144–153, Melbourne, Australia, December 1991. IEEE-CS Press.

[33] Giuliana Franceschinis. *Modellazione e Valutazione delle Prestazioni di Sistemi Concorrenti: le Reti di Petri Stocastiche Ben-Formate e loro Tecniche di Analisi.* PhD thesis, Università di Torino, Dipartimento di Informatica, February 1993.

[34] J.M. Couvreur, S. Haddad, and J.F. Peyre. Resolution parametree d'une famille de systemes d'equations lineaires a solutions positives. Technical Report 90–38, Université Paris 6, 4 Place Jussieu, 75252 Paris Cedex 05, France, September 1990. IBP Tech. Report (in French).

[35] J.M. Couvreur and J. Martinez. Linear invariants in commutative high-level nets. In *Proc. 10^{th} International Conference on Application and Theory of Petri Nets*, Bonn, Germany, June 1989.

[36] M. Silva, J. Martinez, P. Ladet, and H. Alla. Generalized inverses and the calculation of symbolic invariants for coloured Petri nets. *Technique et Science Informatiques*, 4:113–126, 1985.

[37] S. Haddad and J.M. Couvreur. Towards a general and powerful computation of flows for parametrized coloured nets. In *Proc. 9^{th} Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

[38] J.M. Couvreur. The general computation of flows for coloured nets. In *Proc. 11^{th} International Conference on Application and Theory of Petri Nets*, Paris, France, June 1990.

[39] G. Chiola, J. Campos, J.M. Colom, M. Silva, and C. Anglano. Operational analysis of timed Petri nets and applications to the computation of performance bounds. In *Proc. 5th Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.

[40] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(1), May 1984.

[41] M. Ajmone Marsan and G. Chiola. *On Petri Nets with Deterministic and Exponentially Distributed Firing Times*, volume 266 of *LNCS*, pages 132–145. Springer Verlag, 1987.

[42] C. Lindemann. An improved numerical algorithm for calculating steady-state solutions of Deterministic and Stochastic Petri net models. In *Proc. 4th Intern. Workshop on Petri Nets and Performance Models*, pages 176,185, Melbourne, Australia, December 1991. IEEE-CS Press.

[43] V. Mainkar, H. Chio, and K. Trivedi. Sensitivity analysis of Markov regenerative stochastic Petri nets. In *Proc. 5th Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.

[44] W. Henderson, D. Lucic, and P.G. Taylor. A net level performance analysis of stochastic Petri nets. *Journal of Australian Mathematical Soc. Ser. B*, 31:176–187, 1989.

[45] W. Henderson and P.G. Taylor. Embedded processes in stochastic Petri nets. *IEEE Transactions on Software Engineering*, 17(2), February 1991.

[46] A.A. Lazar and T.G. Robertazzi. Markovian Petri net protocols with product form solution. In *Proc. of IEEE INFOCOM'87*, San Francisco, CA, USA, March 1987. Also in *Performance Evaluation*, Vol.12, 1991, pp.67–77.

[47] T.G. Robertazzi. Why most stochastic Petri nets are non-product form networks. Technical report, University of New York, 1991.

[48] J.L. Coleman, W. Henderson, and P.G. Taylor. Product form equilibriub distributions and algorithm for classes of batch movement queueing networks and stochastic Petri nets. Technical report, University of Adelaide, 1992.

[49] M. Sereno and G. Balbo. Computational algorithms for product form solution stochastic Petri nets. In *Proc. 5th Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.

[50] G. Balbo and M. Sereno. Mean value analysis of stochastic Petri nets. Technical report, University of Torino, 1993.

[51] J. P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Communications of the ACM*, 16(9):527–531, September 1973.

[52] M. Reiser and S. S. Lavenberg. Mean value analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2):313–322, April 1980.

[53] J.G. Kemeni and J.L. Snell. *Finite Markov Chains*. Van Nostrand, Princeton, NJ, 1960.

[54] C. Anglano, S. Donatelli, and R. Gaeta. Parallel architectures with regular structure: a case study in modelling using SWN. In *Proc. 5th Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.

[55] S. Donatelli G. Balbo and G. Franceschinis. Understanding parallel programs behaviour through petri net models. *Journal of Parallel and Distributed Computing*, 15(3), July 1992.

[56] K. M. Chandy, U. Herzog, and L. S. Woo. Parametric analysis of queueing networks. *IBM Journal of R. & D.*, 19(1):36–42, January 1975.

[57] K. M. Chandy and C. H. Sauer. Approximate methods for analyzing queueing network models of computer systems. *ACM Computing Surveys*, 10(3):281–317, September 1978.

[58] P.J. Courtois. *Decomposability: Queueing and Computer Systems Applications*. Academic Press, New York, 1977.

[59] M. Ajmone Marsan, G. Balbo, G. Conte, and F. Gregoretti. Modeling bus contention and memory interference in a multiprocessor system. *IEEE Transactions on Computers*, 32(1):60–72, January 1983.

[60] G. Balbo, S. C. Bruell, and S. Ghanta. Combining queueing network and generalized stochastic Petri net models for the analysis of some software blocking phenomena. *IEEE Transactions on Software Engineering*, 12(4):561–576, April 1986.

[61] G. Balbo, S. C. Bruell, and S. Ghanta. Combining queueing network and generalized stochastic Petri nets for the solution of complex models of system behavior. *IEEE Transactions on Computers*, 37(10):1251–1268, October 1988.

[62] P. Buchholz. Numerical solution methods based on structured descriptions of Markovian models. In *Proc. 5^{th} Int. Conf. Modeling Techniques and Tools for Computer Performance Evaluation*, Torino, Italy, February 1991.

[63] G. Ciardo. *Analysis of large Petri net models.* PhD thesis, Department of Computer Science, Duke University, Durham, NC, USA, 1989.

[64] M.F. Neuts. *Matrix Geometric Solutions in Stochastic Models.* Johns Hopkins University Press, Baltimore, MD, 1981.