# Compositional Construction of SWN models

Technical Report No. CSG-21-96

Isabel Rojas *
Department of Computer Science
University of Edinburgh

**Abstract**

This report presents a method for the compositional construction of Stochastic Petri Net models. The method is defined over Stochastic Well-formed Nets in order to take advantage of the state space reduction properties of this formalism. The set of composition operations is based on the operators of Stochastic Process Algebra, augmented with operations that reflect the different types of synchronisation supported by Petri nets. Several examples are presented to illustrate the use of the method. These are developed following a set of guidelines for model construction.

*The author is supported by a grant from the Venezuelan Government Research Council(CONICIT)

# 1 Introduction

Process Algebras (PA) are abstract languages for the specification and understanding of concurrent systems. Well-known examples include the Calculus of Communicating Systems (CCS) [1] and Communicating Sequential Processes (CSP) [2]. A system is characterised by its active components and the communications between them. Actions are the building blocks of the system. They are used to describe sequential components that run concurrently and cooperate through communication. Unlike Petri nets there is no notion of entity or flow within the model. However, compositional reasoning is an integral part of the language. Complex systems are built starting from actions and applying the constructors of the algebra which are operators available for composition as well as mechanisms for abstraction, which disregard internal details. PA models are created by the composition of processes or components, which are normally associated with certain functions or subsystems of the system modelled. In contrast, most Petri net (PN) based modelling formalisms represent the system modelled as a flat net. This net may not clearly reflect the elements that participate in the system and the way they communicate or interact. It can also be difficult to determine the model's behaviour, to prove some of its properties and can it affect the understanding of the model by others. The creation of a PN model is mostly state driven. The evolution from one state to another is what defines the structure of the net. Therefore it can be the case that a function of a process in the system modelled is represented by a set of dispersed transitions, thus making it difficult to isolate the elements representing a certain process from a given structure.

Viewing the model as a set of components that interact is more appropriate, especially for models of parallel and distributed systems. As stated in [3] the way in which distributed systems are perceived is as loosely coupled components running in parallel and communicating by message passing. The description, construction and evolution of these systems is facilitated by separating the system structure into a set of components with autonomous behaviour.

Based on sub-models representing components of a system, a model can then be developed as the composition of sub-models. Composition leads to a hierarchical approach to model construction. The resulting model has a structure that reflects the structure of the system itself. Models of components can be developed by different modellers and, in principle, libraries of re-usable components can be formed.

There have been many studies on compositional construction of untimed PN models, for example [4, 5, 6]. Most have been aimed at the deduction of structural properties of the composed model and the compositional construction of its state space. In [7], for example, a client-server protocol is defined for the composition of PN models. It divides the model into client, server and client-server components, and defines a set of basic rules on which the protocol relies. Communication is asynchronous and there is no notion of synchronisation between components other than the client-server relation.

Stochastic Petri Net (SPN) formalisms do not offer explicit primitives for the compositional construction of models. To the best of the author's knowledge, the closest approach to formally defining compositional primitives in SPN are the *join* and *replicate* operators proposed in [8]. However, these operators are defined to compose systems from repeated structures (through the replicate operator) previously identified, and allow them to communicate in an asynchronous manner (through the join operator). The compositional construction allows the modeller to take advantage of the repeated structure in the model, and of the reward structure defined over the model, to reduce the state space required to obtain performance measures for the system. There is no support for compositional operations such as concurrency, choice or sequential composition.

Donatelli's recent work on Superposed Stochastic Automata (SSA) [9] and Superposed GSPN (SGSPN) [10] emphasises composition, but from the point of view of the solution of the associated Markov process. In both cases the model is defined as a set of interacting components, but the interactions are not defined as part of a basic formalism. Moreover, the way in which components can interact is limited to synchronous communication between components.

In [11], Hierarchical Generalised Coloured Stochastic Petri Nets (HGCSPNs) and Stochastic Well-Formed Nets (SWN)[12] are combined to generate a reduced Markov Chain from a hierarchical net specification. Subnets are combined in a higher level of hierarchy by using an aggregate view of the subnet behaviour which is relevant to its environment. This method only supports communication between subnets in an asynchronous manner. Buchholz also proposes an approximate aggregation method to allow the analysis of larger nets.

Stochastic Process Algebras (SPA) have been introduced for generating performance aspects in the functional analysis of complex systems. They are based on untimed Process Algebra, and extend the basic actions with exponential delays to generate a Markov process from which performance measures of the system modelled can be derived. Examples of some SPA are Performance Evaluation Process Algebra (PEPA) [13] and Timed Performance Processes (TIPP) [14].

In [15] it is suggested that perhaps the best way to incorporate compositional primitives into SPNs would be based on the operators of SPA. A straightforward translation, however, would not take into account the synchronisation properties of the Petri net formalism. In this report we define a set of operators for the composition of SWN, a class of coloured SPN. These operators are based on those of SPA. The definitions have been made over SWN in order to take advantage of the state space reduction properties defined for SWN. A component is viewed as a blackbox for which an interface is defined. This interface will correspond to the set of places and transitions by which the component can interact, communicate or synchronise with other components in the system.

The rest of this report is structured in the following way. In Section 2 we briefly introduce SWN, outlining the concepts of relevance for the definition of the composition operations for the creation of SWN models. We then proceed, in Section 3, to describe the compositional SWN model proposed. Here we define the general structure of the components and the composition operations. This section is concluded with a small example that illustrates how the operators proposed can be employed to obtain a model of a system through the composition of its parts. In Section 4 we propose a set of construction guidelines based on the information about sub-components that is required, preserved and/or lost when applying each of the composition operators. At the end of this section we present another, slightly more complex, example to which we apply the guidelines proposed. Finally in Section 5 we present the conclusions of this work and discuss topics of further research in this area.

# 2 An introduction to *Stochastic Well-Formed Nets*

In this section I will assume that the reader is familiar with the Petri net notation (GSPN, SPN, etc.), both uncoloured and coloured versions. Therefore, I will only briefly introduce their definitions as a way of refreshing the reader's knowledge. For further details the reader is referred to [16], for the uncoloured versions, and to [17], for the coloured.

Let us first explain some notation to be employed in this section. Given $\{F_1, \cdots, F_n\}$, a family of sets with a set of indexes $I = \{1, \cdots, n\}$, the Cartesian product of this family is denoted by $\bigotimes_{i \in I} F_i$. A *multiset* is, intuitively, a set that can contain several occurrences of the same element. We will denote by $Bag(A)$ the set of finite multisets over a set A.

## 2.1 Petri nets

Petri nets (PN) are a graphical and mathematical modelling tool for describing concurrent systems. Graphically the systems are modelled by means of entities called *places* and *transitions*, which can be connected by oriented *arcs*. Transitions represent events, while places represent conditions. PN can be seen as directed bipartite graphs $PN = (V, A)$, whose set of nodes $V$ can be partitioned into two disjoint sets of *places* and *transitions*. The *arcs* in $A$ can go from places to transitions (input arcs) or transitions to places (output arcs).

Formally, a PN *model* is defined as a 4-tuple:

$$PN = \langle P, T, In, O \rangle$$

where:

- $P$ a finite set of places;
- $T$ a finite set of transitions, $P \cap T = \emptyset$;

- $In \subseteq P \times T$ the set of *input* arcs; the function $W^-(p,t) : In \rightarrow \mathbb{N}^+$ gives the multiplicity of the input arc from $p$ to $t$;
- $O \subseteq T \times P$ the set of *output* arcs; the function $W^+(p,t) : O \rightarrow \mathbb{N}^+$ gives the multiplicity of the output arc from $t$ to $p$.

We allow more than one arc to connect a place $p$ to a transition $t$, or viceversa. These arcs are replaced by a single *weighted* arc, where the weight, or *multiplicity*, corresponds to the original number of arcs connecting $p$ with $t$, or vice-versa. A place $p$ is an *input* place of a transition $t$ if there is an arc from $p$ to $t$. $p$ is an *output* place of $t$ if there is an arc from $t$ to $p$. We denote by ${}^\bullet t$ and $t^\bullet$ the sets of input and output places, respectively, of a transition $t$; and by $W^-(p,\cdot)$ and $W^+(p,\cdot)$ the set of transitions for which $p$ is an input or an output place, respectively.

An extension of PN is the incorporation of a third type of arc, called *inhibiting arcs* that also connect places with transitions ($H \subseteq P \times T$). The function $W^h(p,t) : H \rightarrow \mathbb{N}^+$ gives the multiplicity of the inhibiting arc from $p$ to $t$. Their use will be made clear later.

The notion of distributed state is supported in PN by the introduction of a *marking* function $M : P \rightarrow \mathbb{N}$. $M(p)$ specifies the number of tokens contained in a place $p$. The *initial marking* of the net represents initial distribution of tokens in the places of the net. A PN *system* is given by a $PN$ structure plus an *initial marking*.

The *dynamic* behaviour of a PN *system* is specified by the *enabling* and *firing* rules. A transition $t$ is said to be *enabled* if each of its input places has at least as many tokens as the multiplicity of the input arc from the place to the transition $t$, and if each inhibiting place $p_h$ of the transition has less tokens than the multiplicity of the inhibiting arc from $p_h$ to $t$. Enabled transitions can fire, removing from each input place, as many tokens as the multiplicity of its input arc, and placing in each output place, as many tokens as the multiplicity of its output arc.

## 2.2 Coloured Petri Nets

CPNs have been introduced as a modelling tool to represent and study complex systems with symmetric characteristics. Colours are mainly used to model two aspects: different behaviour patterns of entities in the system, and different parts of the system with similar or equivalent structure [18]. With the use of CPNs, the symmetries in the system can be expressed in a more compact form than with normal (non-coloured) Petri nets. They are used to group transitions with similar behaviour, that differ according to their entries (tokens). Different entities (resources, data, etc.) are identified by different "coloured" tokens that can flow through the net. Colouring could refer to places, transitions or tokens. Not only can the colouring be used as a way to represent, in a more compact form, a symmetric Petri Net, but the colour of a token can supply information useful for the determination of parameters of the net.

There are various definitions of CPNs, which differ mainly by which elements are considered to be coloured [19], [18] and [17]. We will define a *Colour Set* of a place as the set of *colours* that tokens can take in a place. Each token can take one colour, and there could be more than one token per colour. We require colour sets to be finite in order to ensure that a coloured net may be 'unfolded' into its equivalent PN. The colour of a transition is determined by the colour of its input and output places.

Formally, a CPN is a tuple $(P, T, C, \Sigma, W^+, W^-, M_o)$ where:

- $P$ is a finite, non-empty set of places;
- $T$ is a finite, non-empty set of transitions; $P \cap T = \emptyset$;
- $\Sigma$ is a finite set of types called *Colour Sets*;
- $C$ is a function from places to colour sets, $C : P \to \Sigma$, which associates each place with a colour set;[1]we denote by $C(p)$ the colour set of a place $p$, and similarly $C(t)$ as the colour set of a transition $t$;
- $W^-, W^+$ are a set of functions $W^-(p,t), W^+(p,t) : C(t) \to Bag(C(p))$;
- $M_0$ is the initial marking.

The function, $M$, defining a marking in the CPN, $M : C(p) \to \mathbb{N}$, determines for each place how many tokens there are of each colour of its associated colour set. If a place $p$ is not input(output) to a transition $t$ the function $W^-(p,t)((W^+(p,t))$ will return 0.

The firing rule is defined by:

- A transition $t$ is enabled for a marking $M$ and a colour $c_t \in C(t)$ if and only if: $\forall p \in P : \ M(p) \geq W^-(p, t, c_t)$;
- The firing of $t$ for a marking $m$ and a colour $c_t \in C(t)$ gives a new marking $M'$ defined by: $\forall p \in P : \ M'(p) = M(p) - W^-(p, t, c_t) + W^+(p, t, c_t)$.

## 2.3 Well-Formed Nets

*Well-formed coloured nets* (WN's) are identical to *Coloured Petri nets* (CPN's) from an expressive power point of view [12]. Any CPN can be translated, into an equivalent WN model with the same underlying structure. In WN the expression

[1]The colour domain of a transition is determined by the colour set of it input and output places

of the colour functions and of composition of colour classes are rewritten in a more explicit or parametric form, in terms of a set of basic constructs provided by the formalism.

In WN's a token can be regarded as an instance of a data structure with a certain number of fields whose semantics depends on the place that the token belongs to. The definition of the "data type" associated with each place is called *place colour domain*. The colours representing elements of the same type are grouped in a class. The domain of a place is selected from a set of basic types called *colour classes*. We will denote as $C_i$ with $i$ ranging from 1 to n, the *colour classes* of the net. Elements within a colour class may be ordered, assuming that this ordering is circular, so that the *successor* function applied to the last element returns the first one. When objects of the same class have different behaviour it is important to partition the class into *(static) sub-classes*, denoted $D_{i,q}$ where $q = \{1, ..., n_i\}$ and $i$ is the identifier of the colour class they belong to.
WN's distinguish three main families of colour functions:

- The projection or identity function $X$, which allows the selection of a particular object in a class $C_i$ or in a sub-class $D_{i,q}$.
- The successor function $!X$ (or $\oplus X$) to be used on ordered sets. It represents the successor of the object selected by $X$, which means that it only makes sense when it is applied over a transition which also has a function $X$ in one of its arcs.
- The diffusion or synchronisation function $S$. When associated with an input arc it represents the synchronisation of all the elements of the colour domain of the connected input place. If it is on an output arc, it will diffuse all the object of its colour domain. This function can also be defined over static sub-classes of the colour domains of the associated input or output place, in this case denoted by $S_{i,q}$, where $i$ refers to the colour class $C_i$ and $q$ to the static sub-class within $C_i$

The transitions in a WN can be considered as procedures with formal parameters. These parameters are called *transition colour domains*; their declaration is part of the net description, and the type associated with each parameter must be a colour class. The colour domain of a transition $t$ ($C(t)$) is constrained by the colour domains of its input, inhibitor and output places. A transition, whose formal parameters have been instantiated to actual values is called a *transition instance*, denoted $[t, c]$, where $c \in C(t)$ represents the assignment of actual values to the transition parameters. The enabling of a transition instance $[t, c]$ is determined by evaluating the transition's predicates and the arc expressions of all input and inhibiting places with respect to the assignment $c$.

A major interest of WN's is that they provide a modelling framework in which symmetries appear naturally. This allows the reduction of the size and complexity of the representation, maintaining the modelling power of unconstrained coloured nets.

Let us now introduce the formal definition of WNs as given in [12].

**Definition 1** *A Well-Formed net system is a 10-tuple*

$$WN = \langle P, T, C, J, W^-, W^+, W^h, \Phi, \pi, M_0 \rangle$$

*where:*

1. *$P$ is a finite set of places;*
2. *$T$ is a finite set of transitions, $P \cap T = \emptyset$;*
3. *$C$ is the family of colour classes: $C = \{C_1, ..., C_n\}$ (we denote by $I = 1, ..., n$ the ordered set of indexes) with $C_i \cap C_j = \emptyset$ for any $C_i, C_j \in C$; Any $C_i \in C$ is possibly partitioned into static sub-classes $C_i = \bigcup_{q=1}^{n_i} D_{i,q}$;*
4. *$J : P \cup T \rightarrow Bag(I)$, where $Bag(I)$ is the multiset on $I$. $C(\tau) = C_{J(\tau)}$ denotes the colour domain of node $\tau$;*
5. *$W^-, W^+, W^h : W^-(p,t), W^+(p,t), W^h(p,t) \in [C_{J(t)} \rightarrow Bag(C_{J(p)})]$ the input, output, and inhibition functions are arc expressions;*
6. *$\Phi(t) : C_{J(t)} \rightarrow \{TRUE, FALSE\}$ is a standard predicate associated with a transition $t$. By default we will assume that $\forall t \in T$, $\Phi(t) = TRUE$;*
7. *$\pi : T \rightarrow \mathbb{N}$ the priority function. By default we will assume that $\forall t \in T\ \pi(t) = 0$;*
8. *$M_0 : M_0(p) \in Bag(C(p))$ is the initial marking.*

The syntactic definition of WN's leads to new algorithms based on the concept of *symbolic marking* [12]. A symbolic marking represents an equivalence class on the state space of the WN model. The *symbolic reachability graph* (SRG) of a WN is based on the idea of symmetry of objects of the basic colour classes. It consists of a symbolic representation of all possible states of a model and the possibility of transition from one to another. The symbolic markings together with a *symbolic firing rule* allow the construction of the SRG.

The symbolic marking introduces the concept of *dynamic sub-classes*, which represent sets of objects that are not identified individually but are known to permute with each other in any firing instance to produce markings that belong to the same equivalence class. A dynamic sub-class is characterised by its cardinality, and by the static sub-classes to which the represented objects belong. The concept of dynamic sub-class affects both the symbolic marking representation and the symbolic firing. Using dynamic sub-classes instead of variables in the marking representation allows a much more compact description of the marking itself. The set of dynamic sub-classes of a colour class $C_i$ in a marking M is denoted by $\widehat{C}_i = \{Z_i^j \mid 0 < j < m\}$, where $m$ is the number of dynamic sub-classes of $C_i$ in a marking $M$. The extended notation $\hat{C}(\tau)$, $\tau \in P \cup T$ is used to denote the set of all possible tuples of dynamic sub-classes in a place/transition colour domain.

## 2.4 Stochastic Well-Formed Nets (*SWN*s)

The step from WN's to Stochastic Well-Formed nets (*SWN*) is similar to that from an untimed PN with priorities to GSPNs (see [16] for going from an untimed PN model to a GSPN model). Here transitions can be *timed* (with an exponentially distributed delay function) or immediate (with firing time zero). In order to guarantee the presence of symmetry, not only from a logical, but also from an stochastic point of view, mean values of transition firing delays can be dependent only on static sub-classes; they cannot be a function of the objects in the class. In this way all objects of a given static sub-class give rise to the same transition firing delay. This can be formalised by the introduction of the following notation. Let

$$\tilde{C}_i = \{D_{i,1}, \cdots, D_{i,q}\}$$

be the set of static sub-classes of a basic colour class $C_i$. Analogously with the notation introduced for dynamic sub-classes, given a transition $t$ with colour domain $C(t) = C_{J(t)}$ we define

$$\tilde{C}(t) = \{\bigotimes_{i=1}^{n}\bigotimes_{j=1}^{e_i} D_{i,u(i,j)} \mid 0 < u(i,j) \leq n_i\}$$

as the colour domain of $t$ defined in terms of the static sub-classes of the basic colour classes that participate in $C(t)$, considering that elements of a static sub-class have the same effect on the firing delay of $t$. Here $e_i$ denotes the number of occurrences of $C_i$ in $C(t)$ and $u(i,j)$ determines which of the static sub-classes of $C_i$ is represented in the $j^{th}$ occurrence of $C_i$ in the colour domain of $t$.

For any $c = \bigotimes_{i=1}^{n}\bigotimes_{j=1}^{e_i} c_i^j \in C(t)$ we also define $\tilde{c} = \bigotimes_{i=1}^{n}\bigotimes_{j=1}^{e_i} \tilde{c}_i^j \in \tilde{C}(t)$ such that $\tilde{c}_i^j = D_{i,q}$ iff $c_i^j \in D_{i,q}$.

We define the *static partition of a marking $M$* denoted $\tilde{M}(p) \in Bag(\tilde{C}(p))$ as:

$$\tilde{M}(p)(\tilde{c}) = \sum_{c':\tilde{c}'=\tilde{c}} M(p)(c')$$

for which following property holds.

$$\forall M, M' \in \mathcal{M}, \forall p \in P, \tilde{M}(p) = \tilde{M}'(p)$$

where $\mathcal{M}$ is the set of all possible markings of the model.

The static partition of a marking represents, for each place and for each Cartesian product of static sub-classes, the number of tuples in the place that belong to the same Cartesian product of static sub-classes.

Hence, we can define the *static partition of a symbolic marking* as well and denote it $\tilde{\mathcal{M}}$.

Having introduced this notation we can now formally define Stochastic Well-formed nets.

**Definition 2** *A* Stochastic Well-Formed coloured net *is a pair* $SWN = \langle WN, \theta \rangle$ *such that*

- $WN$ *is a well-formed Petri net;*[2]
- $\theta$ *is a function defined on the set of transitions* $T$ *such that*

$$\theta(t) : \tilde{C}(t) \times \bigotimes_{p \in P} Bag(\tilde{C}(p)) \to \mathbb{R}$$

*For any timed transition* $t$*, the function* $\theta(t)(\tilde{c}, \tilde{M})$ *represents the average firing rate for any instance of transition* $[t, c]$ *enabled in marking* $M$*. If* $t$ *is an immediate transition, the same function is interpreted as the weight to be normalised within a conflict set in order to obtain the following probability:*

$$\frac{\theta(t)(\tilde{c}, \tilde{M})}{\sum_{M[t',c'\rangle} \theta(t')(\tilde{c}', \tilde{M})}$$

The immediate transitions have firing priority when conflicting with timed transitions. Conflicts between timed transitions are solved by applying a *race policy*, whereas conflicts between immediate transitions are solved by the use of different levels of priority or by a probabilistic function.

# 3 Defining a Compositional SWN model

In Stochastic Process Algebra (SPA) the duration of an activity is determined by a random variable with negative exponential distribution. An activity is described by its type and its rate, i.e. the parameter of the negative exponential distribution governing the duration of the activity. A system is described as an interaction of components. Each component maybe atomic or may itself be composed of components. The grammar of the language defines the ways in which the behaviour of a component may be built up from activities or an interaction

[2] Where we distinguish two types of transitions, namely timed (with priorities equal to zero) and immediate (with priorities greater than zero).

of components. The notions captured by the combinators defined in the different SPA developed differ, for example PEPA captures the notions of prefix, choice, cooperation and hiding, whereas TIPP apart from these has an explicit combinator for the notion of recursion. In general the notions captured in the combinators of SPA are a subset of those of untimed process algebras. For further details on SPA the reader is referred to [13].

The operations defined for the creation of compositional SWN models are based on the compositional combinators of SPA. We have incorporated some additional ones, based on the flexibility and versatility of stochastic Petri net models and the types of synchronisation supported by them.

WN permit the identification of model symmetries by means of the symbolic reachability graph, reducing the state space representation of the model [12], and, as previously mentioned, are identical to CP-nets from an expressive power point of view. For this reason we work with SWNs rather than CP-nets. The operators defined can be applied to uncoloured nets, considering them as neutral-coloured SWNs. The concept of inhibiting arc has been omitted except in the definition of the *polite communication* (See section 3.4.5).

## 3.1 A Compositional Structure

In order to view the system as formed by components that synchronise or communicate with each other, it is necessary to determine how this communication can be made (places, arcs and/or transitions) and how much information is required about a component in order to communicate with it. This should be established according to the way we want the components to communicate, i.e. if two components need to communicate in an asynchronous manner, then the communication should be made through places; if the components need to synchronise, then the best mechanism is through transition fusion. We will not consider arc communication in this work, as it would mean modifying the input or output set of the transitions of a component. In [7] a study of the advantages and disadvantages of each type of communication is presented, as an introduction to the *Client-Server* protocol.

It is not desirable to have to know the whole net structure of a component in order to allow it to communicate with another. Associated with the notion of abstraction in an SPA, it is desirable that certain parts of the net structure of a component are visible only to the component itself. In this work a component is defined as a blackbox with an associated interface by which it can communicate and/or synchronise with other components.

The interface of a component is defined as a set of *entry places* ($ES$), by which a component receives information from other components, a set of *final places* ($FS$), from which the component transfers information to others, and a set of *synchronising transitions ($ST$)*, by which the component can synchronise with other components.

## 3.2 Data Flow versus Control Flow

When defining a compositional formalism in Petri nets we also have to associate a meaning to the tokens or elements transferred between the components. In SPAs there is no notion of data or information flow. In Petri nets to say that a component $A$ occurs before a component $B$, can mean either that $A$ has to arrive at a complete halt before $B$ can proceed, or that $A$ processes information and transfers the output to $B$, i.e. the communication between components can be seen either as a *control flow* relation or a *data flow* relation.

In SPA, where data is not represented, the relation is implicitly control flow. Our initial work with a compositional formalism for Petri nets we tried to follow this approach. Re-installing the initial marking of each component and guaranteeing a strict order in the transition firing in different components, turned out to be difficult problems with a strict control flow relation. Subsequent work has focussed on data flow and the rest of this report will present that work.

## 3.3 The Basic Element

Based on the idea of a basic element defined in SPA, as the basic construction component and its representation in GSPN [15], a similar concept for a SWN component is introduced. Components can be created by the composition of components or can be a basic component, i.e. non-decomposable into sub-components. All components have the same blackbox structure, a blackbox with an interface, which constitutes a basic block for the construction of more complex components.

**Definition 3** *A* basic SWN (bSWN) *is a net in which there is only one transition ($T_1$) with rate $\alpha \geq 0$. $T_1$ has a set of input places ($In$) and a set of output places ($O$), either of which could be an empty set. The input places cannot intersect with the output places. The set of entry places $ES = In$, the set of final places $FS = O$ and $ES \cap FS = \emptyset$ (see figure 1).*

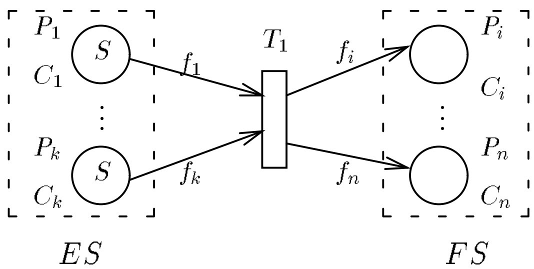



Figure 1: Basic SWN component

A bSWN will have an initial parametric marking (MP) [16], which represents a family of PNs with the same structure. The transition $T_1$ of a bSWN can be declared as synchronise-able or not. Formally this is defined in the following way:

**Definition 4** *A* **basic SWN (bSWN)** *is a* SWN

$$S = \langle P, T, C, J, W^-, W^+, W^h, \Phi, \pi, MP, \theta \rangle$$

*where:*

- $T = \{T_1\}$ *is the transition of the* bSWN*;* $ST = \{T_1\}$ *if* $T_1$ *is* synchronise-able *otherwise* $ST = \emptyset$*;*
- $P = {}^\bullet T_1 \cup T_1{}^\bullet$*; where* ${}^\bullet T_1$ *are the input places of transition* $T_1$ *and* $T_1{}^\bullet$ *its output places;* $ES = {}^\bullet T_1$ *and* $FS = T_1{}^\bullet$*;*
- $C$ *is the family of colour classes:* $C = \{C_1, ..., C_n\}$ *(we denote by* $I = 1, ..., n$ *the ordered set of indices) with* $C_i \cap C_j = \emptyset$ *for any* $C_i, C_j \in C$*; there is no* $C_i \in C$ *that is partitioned into static sub-classes* $C_i = \bigcup_{q=1}^{n_i} D_{i,q}$*;*
- $J : P \cup T \to Bag(I)$*, where* $Bag(I)$ *is the multiset on* $I.C(\tau) = C_{J(\tau)}$ *denotes the colour domain of node* $\tau$*;*
- $\forall p \in P,\ W^-(p, T_1), W^+(p, T_1) : C(T_1) \to Bag(C(p))$*, are the set of input and output arc functions of* $T_1$*, respectively; respectively;*
- $W^h = \emptyset$*;* [3]
- $\Phi(T_1) : C_{J(T_1)} \to \{TRUE, FALSE\}$ *is a standard predicate associated with the transition* $T_1$*. By default* $\Phi(T_1) = TRUE$*;*
- $\theta(T_1) \geq 0$*;* $T_1$ *can be either a timed or an immediate transition;*
- $\pi(T_1) \geq 0$*; If* $\theta(T_1) > 0$ *then* $\pi(T_1) = 0$ *else* $\pi(T_1) > 0$*;*
- $MP$ *is the initial parametric marking of the places in* $P$*;* $\forall p \in FS$ *:* $MP(p) = 0$*.*

We can define a transition predicate for $T_1$ via the function $\Phi$, which will be evaluated in every instantiation of the transition. The function $\pi$ will define the priority of the transition, if it is a timed transition ($\pi(T_1) > 0$) then it will have priority 0, and if it is a immediate transition ($\pi(T_1) = 0$) then it will have a priority greater than or equal to 0. This priority is independent of the priority of possibly conflicting transitions, and has to be determined from the moment the transition is defined.

Differing from the basic element in SPA, a basic SWN can execute the same action a finite number of times, i.e. transition $T_1$ can fire a finite number of times, depending on its initial marking and the multiplicity of its input arcs.

[3] Inhibiting arcs cannot be defined in a *bSWN*.

### 3.4 Compositional Operations

Following the definition of a component in Section 3.1 and the concept of a *bSWN*, the concept of a *compose-able SWN* is introduced, viewing a *bSWN* as its fundamental element.

**Definition 5** *A* compose-able SWN *(cSWN) is either a bSWN or a composition of cSWNs.*

$$cSWN ::= bSWN \mid cSWN * cSWN \mid \circ cSWN$$

*where $*$ represents any binary composition operator and $\circ$ an unary operator.*

In order to compose *cSWN*s it is necessary to define a set of composition operations. In the following examples given to illustrate the operations, the elements composed are *bSWN*, however, the definitions are given in terms of *cSWN*.

#### 3.4.1 Sequential Composition

A component $S$ is obtained from the sequential composition of two components $L$ and $R$ ($L\,;R$), when the final set of $L$ ($FS_L$) intersects with the entry set of $R$ ($ES_R$). This means that $L$ transfers information into $R$. In order to define which places participate in the transfer of information between the components, it is necessary to define a function $\Gamma : (FS'_L \subseteq FS_L) \rightarrow ES_R$ that associates final places of $L$ with entry places of $R$. For a place $p_l \in FS'_L$ with $\Gamma(p_l) = p_r$ where $p_r \in ES_R$, then $C(p_l) = C(p_r)$ ($C(p_i)$ is the colour domain of $p_i$). If this does not hold, then there is no sense in establishing this relation because some of the elements that are defined in $C(p_l)$ are not in $C(p_r)$ or viceversa (see figure 2). Places in the $FS'_L$ that are output places of a common transition cannot have the same image in $ES'_R$, otherwise we would be creating parallel arcs.

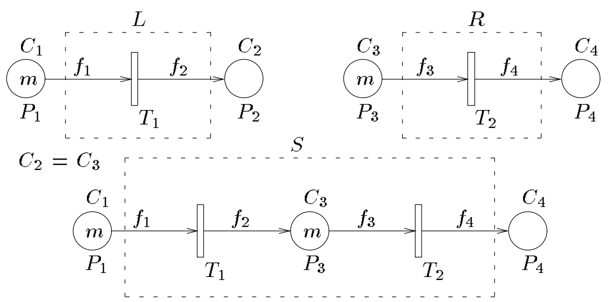



Figure 2: Sequential Composition of SWN components

Formally the $cSWN$ $S$ resulting from the sequential composition of two $cSWN$s, $L\,;R$ is a $cSWN$ defined as:

$$S = \langle P_S, T_S, C_S, J_S, W_S^-, W_S^+, W_S^h, \Phi_S, \pi_S, M_{PS}, \theta_S \rangle$$

where,

- $P_S = P_L \cup P_R - FS'_L$, where $FS'_L$ is the subset of places in the domain of the function $\Gamma$. $ES_S = ES_L \cup ES_R - ES'_R$; where $ES'_R$ is the subset of places in the range of the function $\Gamma$. $FS_S = FS_L \cup FS_R - FS'_L$ ; $\forall p_i, p_j \in FS'_L$, if $\Gamma(p_i) = \Gamma(p_j)$ then $W_L^+(p_i, \cdot) \cap W_L^+(p_j, \cdot) = \emptyset$;
- $T_S = T_L \cup T_R$; $ST_S = ST_L \cup ST_R$;
- $C_S = C_L \cup C_R$;
- $W_S^- = W_L^- \cup W_R^-$;
- $W_S^+ = W_L^+ \cup W_R^+ - \{W_L^+(p_j, t_i) \mid t_i \in T_L p_j \in FS'_L\} \cup \{W^+(p_j, t_i) \mid t_i \in T_L p_j \in ES'_R$ such that $\exists p_k \in FS'_L : \Gamma(p_k) = p_j p_k \in t_i^\bullet\}$; It holds that $W_S^+(p_j, t_i) = W_L^+(p_k, t_i)$ with $p_j$ and $p_k$ as last defined;
- $W_S^h = W_L^h \cup W_R^h$;
- $\forall t \in T_S$ :[4]

$$\Phi_S(t) = \begin{cases} \Phi_L(t) & t \in T_L \\ \Phi_R(t) & t \in T_R \end{cases}$$

- $\pi : T \to \mathbb{N}$ the priority function;

$$\pi_S(t) = \begin{cases} \pi_L(t) & t \in T_L \\ \pi_R(t) & t \in T_R \end{cases}$$

- $MP_S$ the initial parametric marking of $S$;

$$\forall p \in P_S, MP_S(p) = \begin{cases} MP_L(p) & p \in P_L - FS'_L \\ MP_R(p) & otherwise \end{cases}$$

- $\theta$, function that associates a weight to each transition.

$$\theta_S(t) = \begin{cases} \theta_L(t) & t \in T_L \\ \theta_R(t) & t \in T_R \end{cases}$$

[4]The predicates defined over the transitions of the components $L$ and $R$ are preserved in $S$

### 3.4.2 Choice (Pre-selection) Composition

The choice operator permits the probabilistic selection of the sub-component to which a given type of information should be transferred. It is necessary to define which places of the $ES$ of each component participate in the choice. Each component is assigned a weight ($w(Q)$, where $Q$ is the name of the sub-component) corresponding to the probability that it receives the information related to the choice. This weight is assigned to an immediate transition ($t_Q$) associated with the sub-component. The structure of the resulting component is obtained by augmenting the net with a place and two sets of arcs. The place (the choice place, $p_c$) acts as an entry to the choice component. The first set of arcs are from the choice place into each of the immediate transitions associated with the sub-components. The second set of arcs are from each of these immediate transitions to each place in the set of selected places in the entry set of their corresponding sub-components (see figure 3). The arc function on these arcs is the identity function, to make the choice completely probabilistic, neither influenced nor determined by the arc functions. The choice operator only makes sense if all places participating in the choice have the same colour domain, i.e. compete for the same information. The colour domain of the choice place will be that of the participating places. The choice operation can only be defined between distinct components—this is to avoid a transition in a component being immediately enabled by both paths of the choice. The function $Choice$, defines the subset of places of the $ES$ of a component involved in the choice operation.

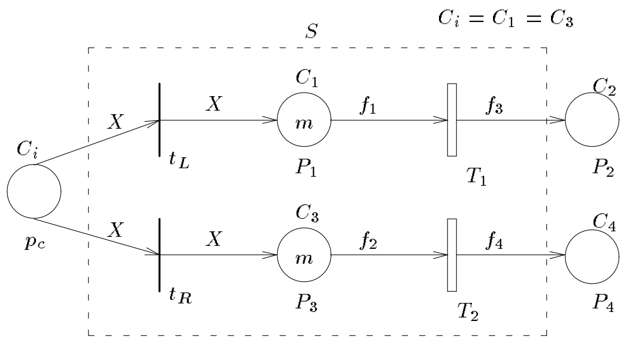



Figure 3: Choice composition of SWN components

Formally the $cSWN$ $S$ resulting from the choice composition of two $cSWN$s, $L + R\{w(L), w(R)\}$ (where $w(P)$ is the weight of a component $P$) is a $cSWN$ defined as:

$$S = \langle P_S, T_S, C_S, J_S, W_S^-, W_S^+, W_S^h, \Phi_S, \pi_S, M_{PS}, \theta_S \rangle$$

where,

- $P_S = P_L \cup P_R \cup \{p_c\}$, where $p_c$ is the entry place of the choice composition. $ES_S = ES_L \cup ES_R \cup \{p_c\} - Choice(L) - Choice(R)$; where $Choice(L)$ and $Choice(R)$ are the subset of places of $ES_L$ and $ES_R$, respectively, that participate in the choice operation; $FS_S = FS_L \cup FS_R$;

- $T_S = T_L \cup T_R \cup \{t_L, t_R\}$, where $t_L$ and $t_R$ are the immediate transitions incorporated by the choice operation, associated with the components $L$ and $R$, respectively; $ST_S = ST_L \cup ST_R$;

- $C_S = C_L \cup C_R$; $\forall p_i, p_j \in Choice(L) \cup Choice(R) : C_S(p_i) = C_S(p_j) = C_S(p_c)$;

$$\forall p \in P_S - p_c : \; C_S(p) = \begin{cases} C_L(p) & p \in P_L \\ C_R(p) & p \in P_R \end{cases}$$

- $W_S^- = W_L^- \cup W_R^- \cup \{W^-(p_c, t_L), W^-(p_c, t_R)\}$ ; where $W_S^-(p_c, t_L) = W_S^-(p_c, t_R) = X$ the identity function defined over the colour domain of $p_c$;

- $W_S^+ = W_L^+ \cup W_R^+ \cup \{W^+(p_j, t_L) \mid p_j \in Choice(L)\} \cup \{W^+(p_k, t_R) \mid p_k \in Choice(R)\}$; $\forall p_i \in Choice(L) \cup Choice(R) : \; W_S^+(p_i, t_L) = W^+(p_i, t_R) = X$ (the identity function defined over the colour domain of $p_c$);

- $W_S^h = W_L^h \cup W_R^h$;

- $\forall t \in T_S$ :

$$\Phi_S(t) = \begin{cases} \Phi_L(t) & t \in T_L \\ \Phi_R(t) & t \in T_R \end{cases}$$

- $\pi : T \to \mathbb{N}$ the priority function;

$$\pi_S(t) = \begin{cases} \pi_L(t) & t \in T_L \\ \pi_R(t) & t \in T_R \\ 0 & t = t_L \vee t = t_R \end{cases}$$

- $\forall p \in P_S$

$$MP_S(p) = \begin{cases} MP_L & p \in P_L - Choice(L) \\ MP_R & p \in P_R - Choice(R) \\ 0 & p \in Choice(L) \cup Choice(R) \\ \sum_{p_i \in Choice(L)} MP_L(p_i) + & \\ \sum_{p_j \in Choice(R)} MP_R(p_j) & p = p_c \end{cases}$$

- $\theta$, function that associates a weight to each transition.

$$\theta_S(t) = \begin{cases} \theta_L(t) & t \in T_L \\ \theta_R(t) & t \in T_R \\ w(L) & t = t_L \\ w(R) & t = t_R \end{cases}$$

### 3.4.3 Competing and Independent Parallel Composition

Parallel composition is defined over two components that can "execute" or have an active token game simultaneously, and possibly independently. Let us consider the parallel composition of the sub-components $L$ and $R$ to obtain the component $S$. Two types of parallelism can be defined: *independent* or *competing*. In independent parallelism the $ES$s of the sub-components are kept separate and the resulting $ES$ is just a union of the $ES$s of the subnets, i.e. $ES_S = ES_L \cup ES_R$. In competing parallelism a subset of the places of $ES_L$ is joined with a subset of places of $ES_R$, provided that their colour domains are equal, so that the sub-components will compete for tokens within these joined places. This differs from the choice composition in the way in which it is determined which sub-component extracts the tokens (fires). Instead of explicitly defining probabilities, as in the choice, here the competition is resolved by a race policy (if all the conflicting transitions are timed), by priorities, or at random (if they are all immediate). This implies that transitions of different sub-components can conflict, i.e. the firing of a transition in one component can affect the enabled condition of another transition in the other component. To determine the sets of conflict places, we define a function $\Lambda : (ES'_L \subseteq ES_L) \rightarrow ES_R$, which defines the ordered pairs of places which are to be merged (fused). The name of a place resulting from the merging of two places $p_l \in ES'_L$ and $p_r \in ES_R$, will be $p_l$. In order to maintain the number of input places for each transition of the participating components, the function $\Lambda$ is defined as one-to-one, i.e. $|ES'_L| = |ES'_R|$ where $ES'_R$ is the range of the function $\Lambda$. The fused place will inherit the arcs of the places associated with it. The competing parallelism can be defined over a single component provided that the range of the function $\Lambda$ does not intersect with its domain, that is a place is not fused with itself, and that places with common input and/or output transitions are not fused, to avoid parallel arcs.

As we have said we are considering that the places being fused have the same colour domain. We could however, relax this condition by saying that their colour domains have only to have the same dimension, i.e. the same number of colour sets participating. With this we could create new colour classes and redefine the existing colour classes as *static subclasses* of the newly created ones. The creation of static sub-classes would require the definition of predicates on those transitions for which the fused place is now input. These predicates would restrict the type of tokens that a transition can extract from the fused place to those with colour set equal to the original input place of the transition.

Formally the *cSWN* $S$ resulting from the independent parallel composition of two *cSWN*s, $L|R$, is defined as:

$$S = \langle P_S, T_S, C_S, J_S, W^-_S, W^+_S, W^h_S, \Phi_S, \pi_S, M_{PS}, \theta_S \rangle$$

where,

- $P_S = P_L \cup P_R$; $ES_S = ES_L \cup ES_R$; $FS_S = FS_L \cup FS_R$;

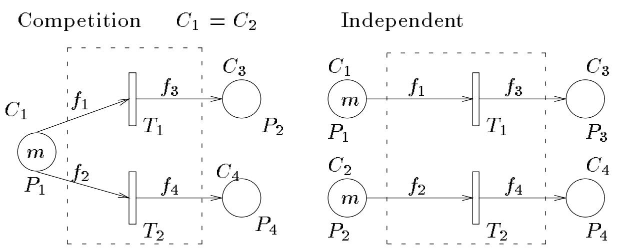



Figure 4: Parallel composition of SWN components

- $T_S = T_L \cup T_R$; $ST_S = ST_L \cup ST_R$;
- $C_S = C_L \cup C_R$;
- $W_S^- = W_L^- \cup W_R^-$;
- $W_S^+ = W_L^+ \cup W_R^+$;
- $W_S^h = W_L^h \cup W_R^h$;
- $\forall t \in T_S$ :

$$\Phi_S(t) = \begin{cases} \Phi_L(t) & t \in T_L \\ \Phi_R(t) & t \in T_R \end{cases}$$

- $\pi : T \rightarrow \mathbb{N}$ the priority function;

$$\pi_S(t) = \begin{cases} \pi_L(t) & t \in T_L \\ \pi_R(t) & t \in T_R \end{cases}$$

- $MP_S = MP_L + MP_R$;
- $\theta$, function that associates a weight to each transition.

$$\theta_S(t) = \begin{cases} \theta_L(t) & t \in T_L \\ \theta_R(t) & t \in T_R \end{cases}$$

Formally the *cSWN* $S$ resulting from the competing parallel composition of two *cSWN*s, $L|_c R$, is defined as:

$$S = \langle P_S, T_S, C_S, J_S, W_S^-, W_S^+, W_S^h, \Phi_S, \pi_S, M_{PS}, \theta_S \rangle$$

where,

- $P_S = P_L \cup P_R - ES'_R$, where $ES'_R$ is the range of $\Lambda(ES'_L)$. $FS_S = FS_L \cup FS_R$ and $ES_S = ES_L \cup ES_R - ES'_R$ ; $|ES'_L| = |ES'_R|$;

- $T_S = T_L \cup T_R$; $ST_S = ST_L \cup ST_R$;

- $C_S = C_L \cup C_R$; $\forall p_l \in ES'_L : C_S(p_l) = C_L(p_l)$;

- $W_S^- = W_L^- \cup W_R^- - \{W_R^-(p_r, t_j) \mid p_r \in ES'_R t_j \in T_R\} \cup \{W^-(p_l, t_j) \mid p_l \in ES'_L t_j \in T_R$ such that $\exists p_k \in ES'_R : \Lambda(p_l) = p_k p_k \in {}^\bullet t_j\}$;

- $W_S^+ = W_L^+ \cup W_R^+ - \{W_R^+(p_r, t_j) \mid p_r \in ES'_R t_j \in T_R\} \cup \{W^+(p_l, t_j) \mid p_l \in ES'_L t_j \in T_R$ such that $\exists p_k \in ES'_R : \Lambda(p_l) = p_k p_k \in t_j^\bullet\}$;

- $W_S^h = W_L^h \cup W_R^h$;

- $\forall t \in T_S$ :

$$\Phi_S(t) = \begin{cases} \Phi_L(t) & t \in T_L \\ \Phi_R(t) & t \in T_R \end{cases}$$

- $\pi : T \to \mathbb{N}$ the priority function;

$$\pi_S(t) = \begin{cases} \pi_L(t) & t \in T_L \\ \pi_R(t) & t \in T_R \end{cases}$$

- The initial marking $MP_S$ is defined as:

$$\forall p \in P_S, MP_S(p) = \begin{cases} MP_L(p) + MP_R(p_k) & p \in ES'_L \text{ with } \Lambda(p) = p_k \\ MP_L(p) & p \in P_L - ES'_L \\ MP_R(p) & p \in P_R - ES'_R \end{cases}$$

- $$\theta_S(t) = \begin{cases} \theta_L(t) & t \in T_L \\ \theta_R(t) & t \in T_R \end{cases}$$

In a similar manner we can formally define the resulting *cSWN* $S'$ when applying the competing parallelism composition over a single component as:

$$S' = \langle P_{S'}, T_{S'}, C_{S'}, J_{S'}, W_{S'}^-, W_{S'}^+, W_{S'}^h, \Phi_{S'}, \pi_{S'}, M_{P\,S'}, \theta_{S'} \rangle$$

where,

- $P_{S'} = P_S - P_{rng}$;where $P_{rng}$ is the set of places in the range of the function $\Lambda$;

- $T_{S'} = T_S$;

- $W^-_{S'} = W^-_S - \{W^-_S(p_k, t) \mid p_k \in P_{rng} \wedge t \in T_S\} \cup \{W^-(p_j, t) \mid t \in T_S \wedge p_j \in P_{dom}$ such that $\exists p_i \in P_{rng}\ (\Lambda(p_j) = p_i \wedge p_i \in {}^\bullet t)\}$; where $p_{dom}$is the domain of the function $\Lambda$; $\forall t \in T_{S'}$ : $W^-_{S'}(p_j, t) = W^-_S(p_i, t)$ with $p_i$ and $p_j$ as previously defined; [5]

- $W^+_{S'} = W^+_S - \{W^+(p_k, t) \mid p_k \in P_{rng} \wedge t \in T_S\} \cup \{W^+(p_j, t) \mid t \in T_S \wedge p_j \in P_{dom}$ such that $\exists p_i \in P_{rng}\ (\Lambda(p_j) = p_i \wedge p_i \in t^\bullet)\}$, $\forall t \in T_{S'}$ : $W^+_{S'}(p_j, t) = W^+_S(p_i, t)$ with $p_i$ and $p_j$ as previously defined;

- $W^h_{S'} = W^h_S$;

- $\Phi_S = \Phi_{S'}$;

- $\pi : T \to \mathbb{N}$ the priority function;

$$\forall t \in T_S : \pi_{S'}(t) = \pi_S(t)$$

- The initial marking $MP_S$ is defined as:

$$\forall p \in P_S, MP_S(p) = \begin{cases} MP_S(p) + MP_S(p_k) & p \in P_{dom} \text{ with } \Lambda(p) = p_k \\ MP_S(p) & p \in P_L - P_{dom} \end{cases}$$

  where $P_{dom}$ is the set of places in the domain of $\Lambda$;

- $\theta_{S'}(t) = \theta_S(t)$

### 3.4.4 The Closing Operator

When we consider systems with non-terminating behaviour it is necessary for a model—or sub-components within a model—to be able to feed information back from a subset of its $FS$ into a subset of its $ES$. This structure is supported by the introduction of the closing operator $CL$. This corresponds to the recursion operator in SPA. For simplicity the closing operation of a component $S'$ is defined over a pairs of places, the first place belonging to the $FS$ of the component and the second to its $ES$. To fuse a set of places we iteratively apply the closing operation over a pair of places at a time. We can fuse multiple final places with a single entry place. This is possible because the closing operation preserves the entry place condition. However, we cannot fuse multiple entry places with a single final transition. In the case that we needed this we would first have to apply the competing parallel composition to the entry places involved and then apply the closing operation.

A function $\Theta$ defined over the component determines the pair of places $(p_f, p_e)$ to be fused, where $p_f \in FS$ and $p_e \in ES$ . The operation generates a fusion place

[5] Places that are input to a common transition cannot be fused, otherwise we would obtain parallel arcs

$p_{fe}$ which will behave both as $p_f$ and $p_e$, inheriting their input and output arcs. The place $p_{fe}$ will not, however, reflect the final place condition of $p_f$, i.e. it will not be presented as a final place in the interface of the resulting $cSWN$. The place $p_e$ is completely represented by $p_{fe}$ reflecting its condition of entry place, i.e. $p_{fe}$ will be in the interface of the resulting $cSWN$ as a entry place. We must notice that $p_e$ can also be an output place of some transition(s). The colour set $(C(p_{fe}))$ of the fused place $p_{fe}$ will be that of the corresponding entry place in the relation, $C(p_{fe}) = C(p_e)$. The condition $C(p_e) = C(p_f)$ must hold in order to guarantee that there will be no tokens passed into the place $p_{fe}$ when acting as $p_e$, which are not defined in its colour set.

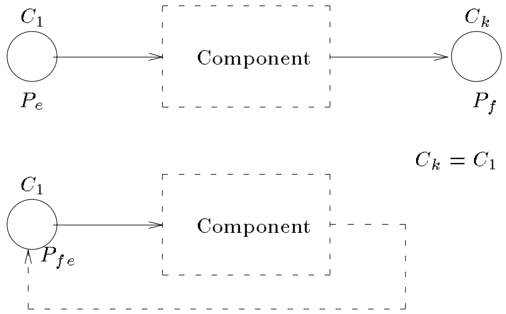



Figure 5: Closing a component

Formally the *cSWN* $S$ resulting from applyingthe closing operation over a *cSWN* $S'$ $(CL(S'))$, is defined as:

$$S = \langle P_S, T_S, C_S, J_S, W_S^-, W_S^+, W_S^h, \Phi_S, \pi_S, M_{PS}, \theta_S \rangle$$

where,

- $P_S = P_{S'} - \{p_e, p_f\} \cup \{p_{fe}\}$;$ES_S = ES_{S'} - \{p_e\} \cup \{p_{fe}\}$; $FS_S = FS_{S'} - \{p_f\}$;
- $T_S = T_{S'}$ ;
- $C_S = C_{S'}$; $C_S(p_{fe}) = C_{S'}(p_e) = C_{S'}(p_f)$;
- $W_S^- = W_{S'}^- - \{W_{S'}^-(p_e, t_i) \in W_{S'}^- \mid t_i \in W_{S'}^-(p_e, \cdot)\} \cup \{W^-(p_{fe}, t_i) \mid t_i \in W_{S'}^-(p_e, \cdot)\}$; $\forall t_i \in W_{S'}^-(p_e, \cdot) : W_S^-(p_{fe}, t_i) = W_{S'}^-(p_e, t_i)$;
- $W_S^+ = W_{S'}^+ - \{W_{S'}^+(p_f, t_i) \mid t_i \in W_{S'}^-(p_f, \cdot)\} \cup \{W^+(p_{fe}, t_i) \mid t_i \in W_{S'}^+(p_f, \cdot)\}$; $\forall t_i \in W_{S'}^+(p_f, \cdot) :\ W_S^+(p_{fe}, t_i) = W_{S'}^+(p_f, t_i)$;
- $W_S^h = W_{S'}^h$;
- $\forall t \in T_S : \Phi_S(t) = \Phi_{S'}(t)$;

- $\forall t \in T_S : \pi_S(t) = \pi_{S'}(t)$;
- The initial marking $MP_S$ is defined as:

$$\forall p \in P_S, MP_S(p) = \begin{cases} M_{P_{S'}}(p) & p \in P_{S'} \\ MP_S(p_e) & p = p_{fe} \end{cases}$$

- $\forall t \in T_S : \theta_S(t) = \theta_{S'}(t)$;

### 3.4.5 Synchronisation

One of the main advantages that Petri nets have for modelling concurrent systems is the number and variety of synchronisations that they can represent. In [20] several types of synchronisations are reviewed. Here we show how to represent those different types of synchronisation. We will define synchronisation of components through the synchronisation of their transitions. The asynchronous synchronisation, in which a sub-component can activate another component by passing information to it, and then carry on with its own activities, can be seen as a special case of sequential composition.

The transition of a *bSWN* can be defined as synchronise-able, i.e. available for synchronisation or *visible* in the interface of the component, or not. By default it is defined as synchronise-able. Extra transitions added as a consequence of a composition operation are not visible in the interface of the component. A non-synchronise-able transition can never be transformed into a synchronise-able one, but the converse is possible. This occurs when employing multiple transitions synchronisations (see *timed synchronisation and polite communication*), where the synchronisation is not represented by a single transition.

Sub-components do not detect the changes made as a consequence of a synchronisation. They still supply the same information to a transition and receive the same type of information from it, once it has fired. The functionality of the subnets involved does not change. This is a characteristic that holds for all types of synchronisation (or communication).

**Untimed Synchronisation, Patient Communication and Impolite Communication** The *untimed synchronisation*, the *impolite communication* and the *patient communication*, produce a structurally similar component when applied to two sub-components. They are all defined as binary operators. Each synchronisation operation is defined over a pair of transitions $(T_1, T_2)$ both synchronise-able. In the case that they both belong to the same component they must be different transitions $(T_1 \neq T_2)$ and their set of input and output places must not intersect. In all three cases the interaction will only be enabled in the case that both transitions would have been enabled.

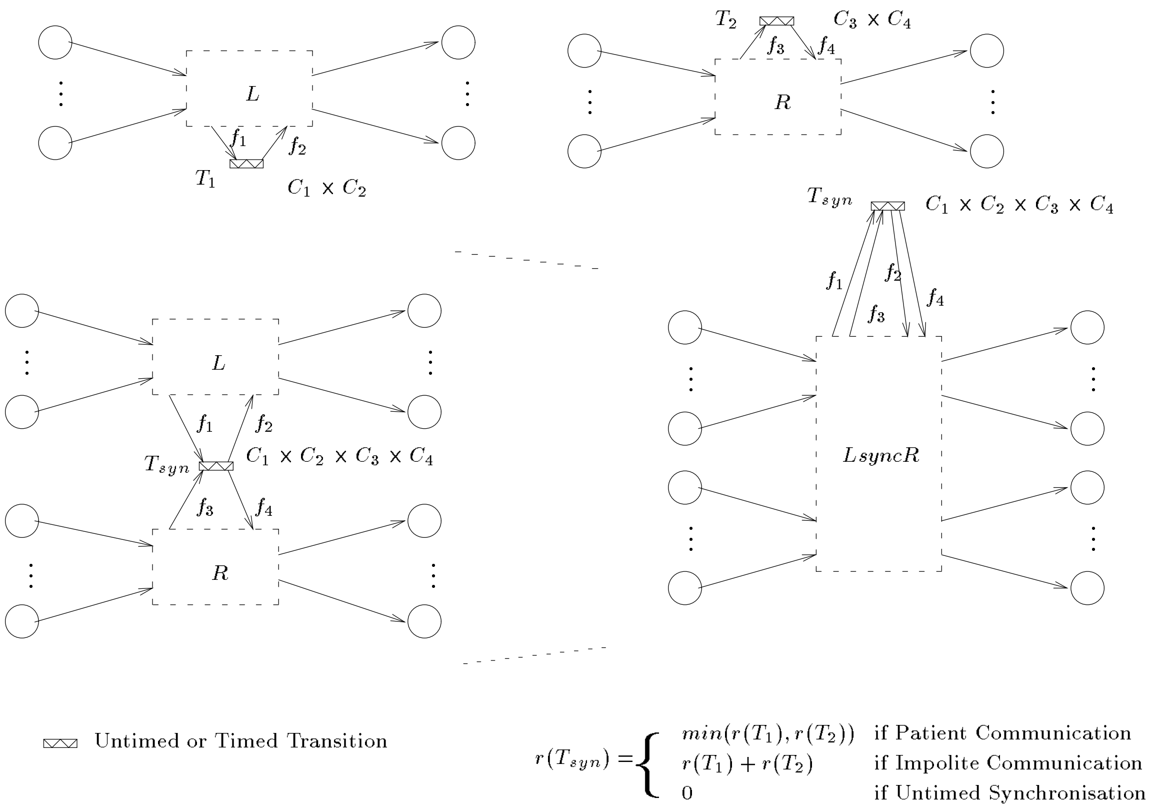



Figure 6: Representation of Untimed Synchronisation, Patient Communication or Impolite Communication of two components

The untimed synchronisation consists of an instantaneous check-pointing event, which ensures that both participants share some knowledge of their mutual situation. From an abstract perspective this can be viewed as the superposition of two immediate transitions, where the joined transition will also be an immediate transition.

In the case of patient communication the interaction is assumed to represent a communication or shared task. The rate of each individual transition represents the capacity of the component to complete its part of the shared task. The interaction is completed by both components working together at the rate of the slower one. Therefore the rate assigned to the joined transition is equal to the minimum of the rate of $T_1$ and the rate of $T_2$, $r(T_{syn}) = min\{r(T_1), r(T_2)\}$.

The impolite communication consists on the synchronisation of two transitions each representing a communication event, such that both transitions "transfer information" at the same time. The first to finish its transfer will terminate the communication. This means that the duration of the communication will be distributed as the minimum of the individual distributions. Since the individual transitions are exponentially distributed the interaction will be exponentially distributed with the sum of the rates $r(T_1) + r(T_2)$, where $T_1$ and $T_2$ are the transitions involved in the operation.

These synchronising operations differ only in the rate assigned to the joined transition ($T_{syn}$) produced when fusing the synchronising transitions.

Given the following facts:

- Both $min$ and $+$ are associative algebraic operations,
- the untimed operation can be seen as either an impolite communication or as a patient communication over transitions with rate zero, and
- all three composition operations produce a single joined transition $T_{syn}$

we can apply any of these operations over more than two components, joining pairs of components at a time. In all three cases information about the colour set of the transitions and their input and output functions is required. This information can be combined to create new predicates for the joined transition which will inherit any predicate associated with the transitions $T_1$ and $T_2$. The resulting joined transition is visible to the environment; $T_1$ and $T_2$ no longer exist. We call this type of synchronisation *single transition synchronisations* because the operations generate a single transition as a consequence of the synchronisation of two transitions.

Formally the *cSWN* $S$ resulting from applying a *single transition* synchronising operation to two *cSWN*s, $L \bowtie R\{T_1, T_2\}$ (Untimed Synchronisation), $L \diamond R\{T_1, T_2\}$ (Patient Communication) or $L \odot R\{T_1, T_2\}$ (Impolite Communication), is a *cSWN* defined as [6]:

$$S = \langle P_S, T_S, C_S, J_S, W_S^-, W_S^+, W_S^h, \Phi_S, \pi_S, M_{PS}, \theta_S \rangle$$

where,

- $P_S = P_L \cup P_R$;
- $T_S = T_L \cup T_R - \{T_1, T_2\} \cup \{T_{syn}\}$; where $T_1$ and $T_2$ are the synchronising transitions of the *cSWN*s $L$ and $R$, respectively; $T_1 \neq T_2$;
- $C_S = C_L \cup C_R$;
- $W_S^- = W_L^- \cup W_R^- - \{W_L^-(p_i, T_1) \mid p_i \in {}^\bullet T_1\} - \{W_R^-(p_i, T_2) \mid p_i \in {}^\bullet T_2\} \cup \{W^-(p_i, T_{syn}) \mid p_i \in ({}^\bullet T_1 \cup {}^\bullet T_2)\}$; $\forall p_i \in {}^\bullet T_{syn} : [W_S^-(p_i, T_{syn}) = W_L^-(p_i, T_1)$ if $p_i \in {}^\bullet T_1]$ or $[W^-(p_i, T_{syn}) = W_R^-(p_i, T_2)$ if $p_i \in {}^\bullet T_2]$;

[6] The same definition applies for the case of a single component considering $L = R$

- $W_S^+ = W_L^+ \cup W_R^+ - \{W_L^+(p_i, T_1) \mid p_i \in T_1^\bullet\} - \{W_R^+(p_i, T_2) \mid p_i \in T_2^\bullet\} \cup \{W^+(p_i, T_{syn}) \mid p_i \in (T_1^\bullet \cup T_2^\bullet\}$; $\forall p_i \in T_{syn}^\bullet : [W_S^+(p_i, T_{syn}) = W_L^+(p_i, T_1)$ if $p_i \in T_1^\bullet]$ or $[W_S^+(p_i, T_{syn}) = W_R^+(p_i, T_2)$ if $p_i \in T_2^\bullet]$;

- $W_S^h = W_L^h \cup W_R^h$;

- $\forall t \in T_S$ :

$$\Phi_S(t) = \begin{cases} \Phi_L(t) & t \in T_L - \{T_1\} \\ \Phi_R(t) & t \in T_R - \{T_2\} \\ \Phi_L(T_1) \wedge \Phi_R(T_2) & t = T_{syn} \end{cases}$$

- $\pi : T \rightarrow \mathbb{N}$ the priority function;

$$\pi_S(t) = \begin{cases} \pi_L(t) & t \in T_L - \{T_1\} \\ \pi_R(t) & t \in T_R - \{T_2\} \\ min(\pi_L(T_1), \pi_R(T_2)) & t = T_{syn} \end{cases}$$

- The initial marking $MP_S$ is defined as:

$$\forall p \in P_S, MP_S(p) = \begin{cases} MP_L(p) & p \in P_L \\ MP_R(p) & p \in P_R \end{cases}$$

- $$\theta_S(t) = \begin{cases} \theta_L(t) & t \in T_L - \{T_1\} \\ \theta_R(t) & t \in T_R - \{T_2\} \\ 0 & t = T_{syn} \;\; \theta_L(T_1) = \theta_R(T_2) = 0 \\ \theta_L(T_1) + \theta_R(T_2) & t = T_{syn} \text{ and the operation is an} \\ & \text{Impolite communication} \\ min(\theta_L(T_1), \theta_R(T_2)) & t = T_{syn} \text{ and the operation is a} \\ & \text{Patient communication} \end{cases}$$

**Timed Synchronisation and Polite Communication** The polite communication and the timed synchronisation can also be considered together because of the form of the resulting component and because the visibility of the transitions that are synchronised is subsequently lost. The polite communication represents the situation where two timed transitions are allowed to communicate in an interleaved manner. In the models proposed we are only considering exponentially distributed timed transitions, therefore the representation of a Coxian-2 distribution required for polite communication [20], cannot be represented by a single transition in the net (see Figure 7). For this reason polite communication between more than two transitions is not allowed. The same applies for the distribution of the timed synchronisation, which cannot be represented by a single transition (see Figure 8). Through the timed synchronisation we synchronise two timed transitions in such a way that the duration of the interaction will be distributed

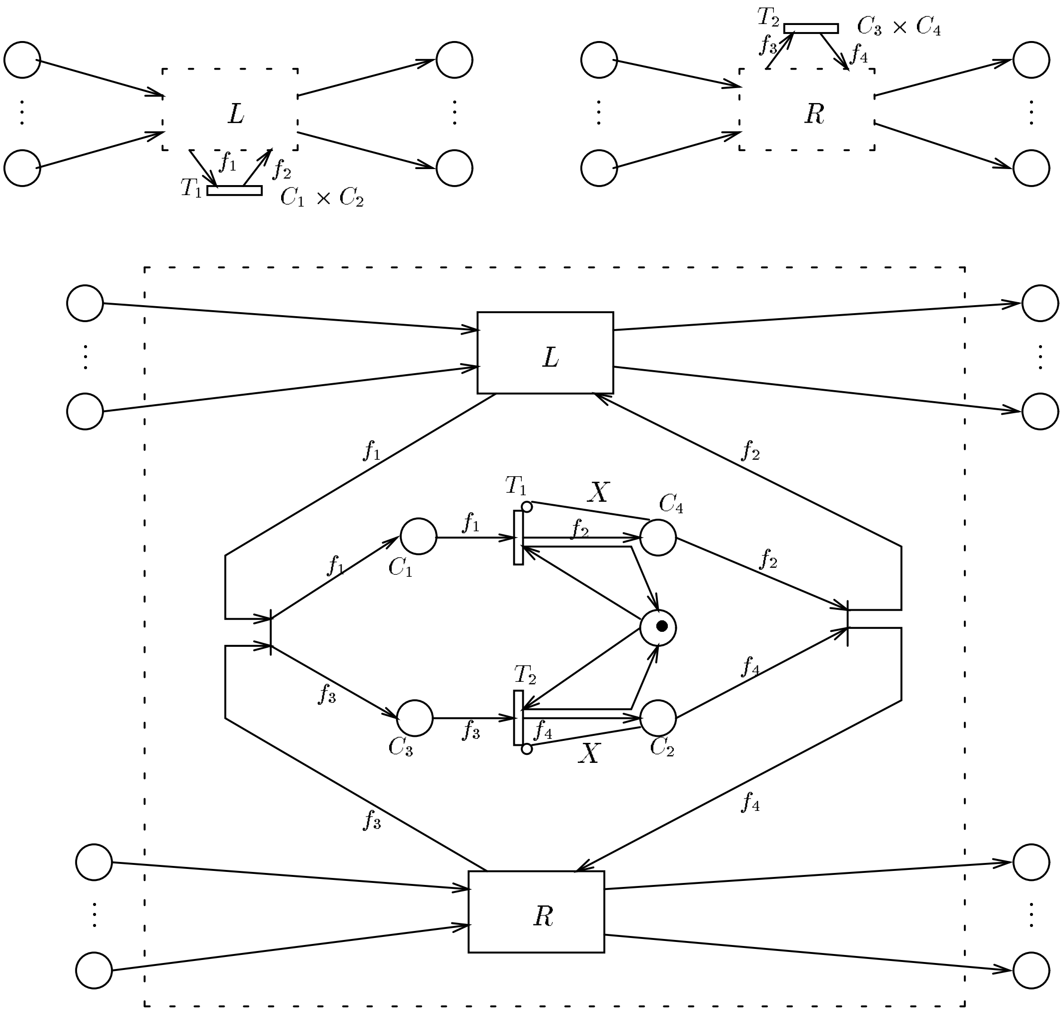
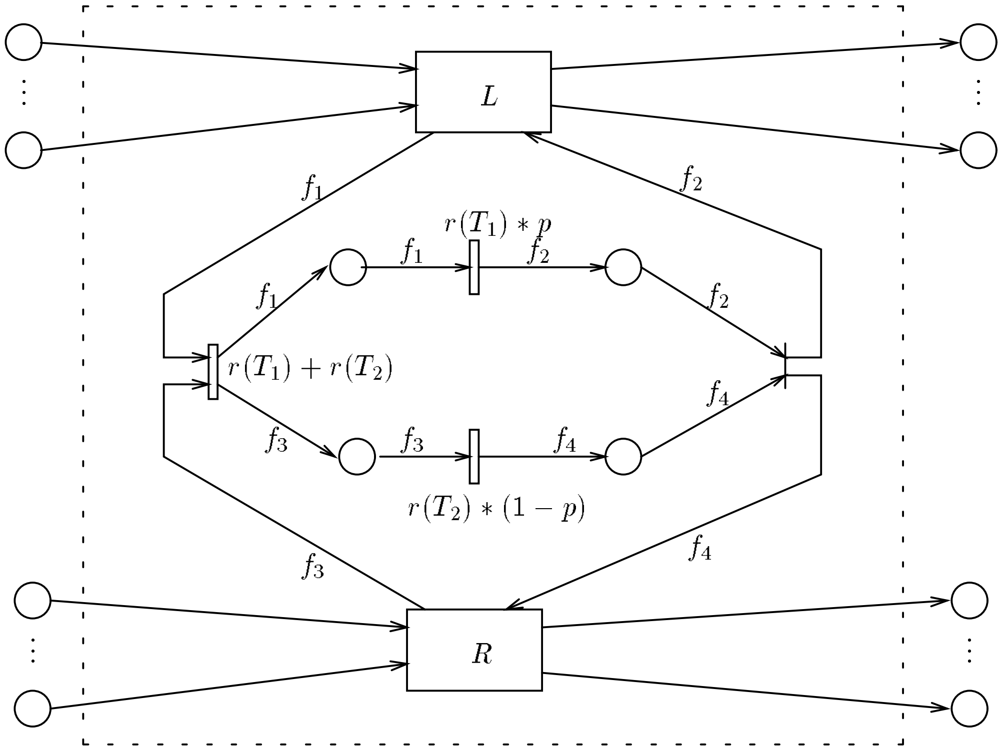



Figure 7: Polite Communication of two components

as the maximum of the delays of the individual transitions. Unfortunately the maximum of two exponential distributions is not an exponential distribution. If it is necessary to synchronise more than two timed transitions by either of these operations, then they must be all synchronised at once, making the timed synchronisation and the polite communication multiple argument operations. The resulting transitions will be considered non-synchronise-able for the reasons mentioned above.




$$p = \frac{r(t_2)}{r(T_1)+r(T_2)}$$

Figure 8: Timed Synchronisation of two components

# 4 Example of a compositional construction of a SWN model

Let us now study a small example where we illustrate how these operations can be applied over the sub-components of a model. The example chosen is the well-known problem of the "Dining Philosophers". There is a group of philosophers sitting around a table, on which are as many forks as philosophers and a huge bowl of spaghetti. However, because the spaghetti is very tangled, a philosopher requires two forks to eat. Therefore, he will have to compete with his neighbours for the use of his forks. A philosopher thinks for a while, then eats and once he has finished eating he returns to thinking.

Before defining the basic components of the model we must determine the basic colour classes. The two types of elements that we must represent are philosophers and forks. As we have said before there is the same number of forks and philosophers. When a philosopher is thinking his fork is considered to be free or usable by another philosopher. When a philosopher is eating he uses his fork

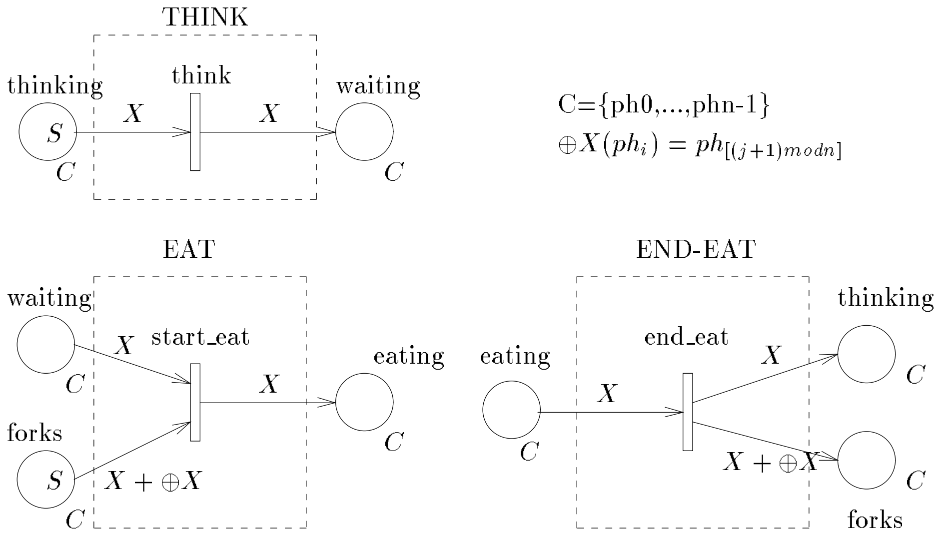



Figure 9: Basic Components of the Dining Philosophers Model

and that of another philosopher. This reflects a direct relation between the two sets of elements, allowing us in this case to represent both philosophers and forks with a single colour set $C = \{ph_o, \cdots, ph_n\}$, where $n$ is the number of forks and philosophers.

In Figure 9 the set of basic components for the dining philosophers models is presented. A philosopher will first think (component THINK), once hungry he will wait until he can eat (placing a token in place waiting). Component EAT models a philosopher eating, for which it requires a hungry philosopher and the availability of his forks, when he has both the philosopher can start-eating. Component END-EAT receives a philosopher who was eating and wants to stop, this component sends the philosopher back to thinking and releases the forks.

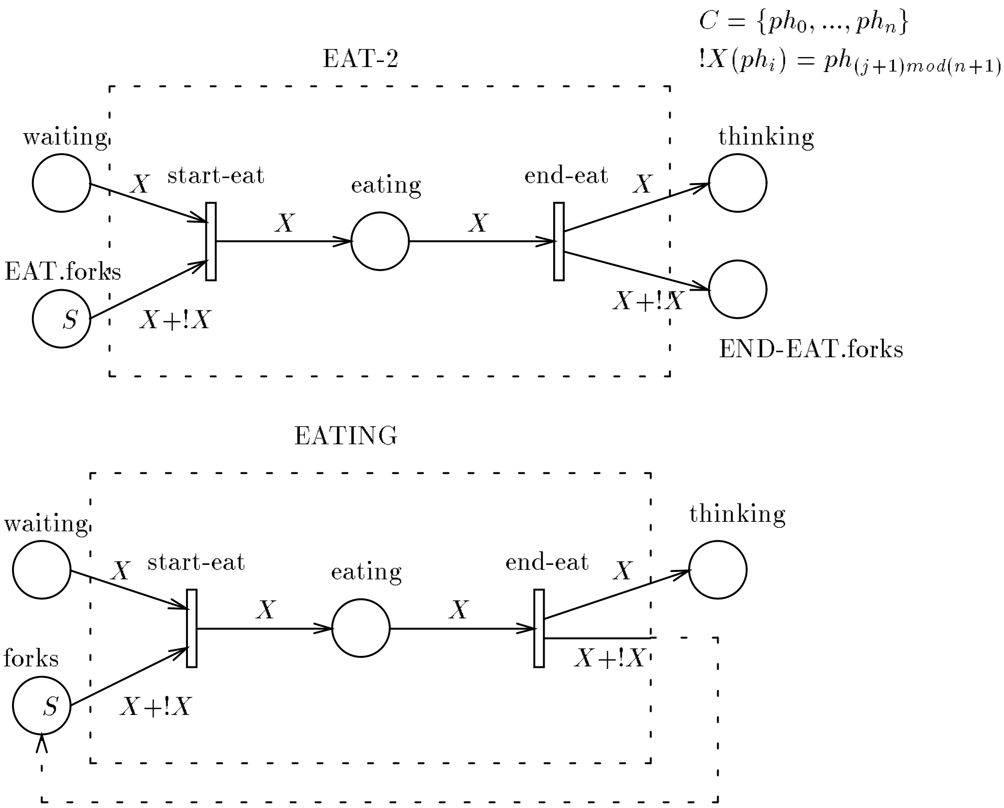
**Construction Steps:** Given the group of *bSWN* components presented in Figure 9, the following steps can be followed in order to obtain the complete model of the problem (see Figures 10 and 11). To identify instances of places that have the same name but which originate in different components we will employ a dot notation. The first element corresponds to the name of the component the place originally belongs to, and the second element the name of the place (component names are in capital letters and place names in small letters).

- Sequential composition of *EAT* with *END-EAT*.

  $EAT\text{-}2 = EAT; END\text{-}EAT$ with $\Gamma(EAT.eating) = END\text{-}EAT.eating$;

- Closing operation over *EAT-2*.

  $EATING = CL(EAT\text{-}2)$ with $\Theta(END\text{-}EAT.forks) = EAT.forks$;




Note: Colour Sets have been obmited because all places have colour set C.

Figure 10: Applying the Compositional operations to form the Dining Philosophers Model from existing components(A)

- Sequential composition of THINK with EATING.

  THINK-EAT $= THINK; EATING$ with $\Gamma(THINK.waiting) = EATING.waiting$;

- Closing operation over THINK-EAT.

  *DINING-PHIL* $= CL($*THINK-EAT*$)$ with $\Theta($*EATING.thinking*$) = THINK.thinking$.

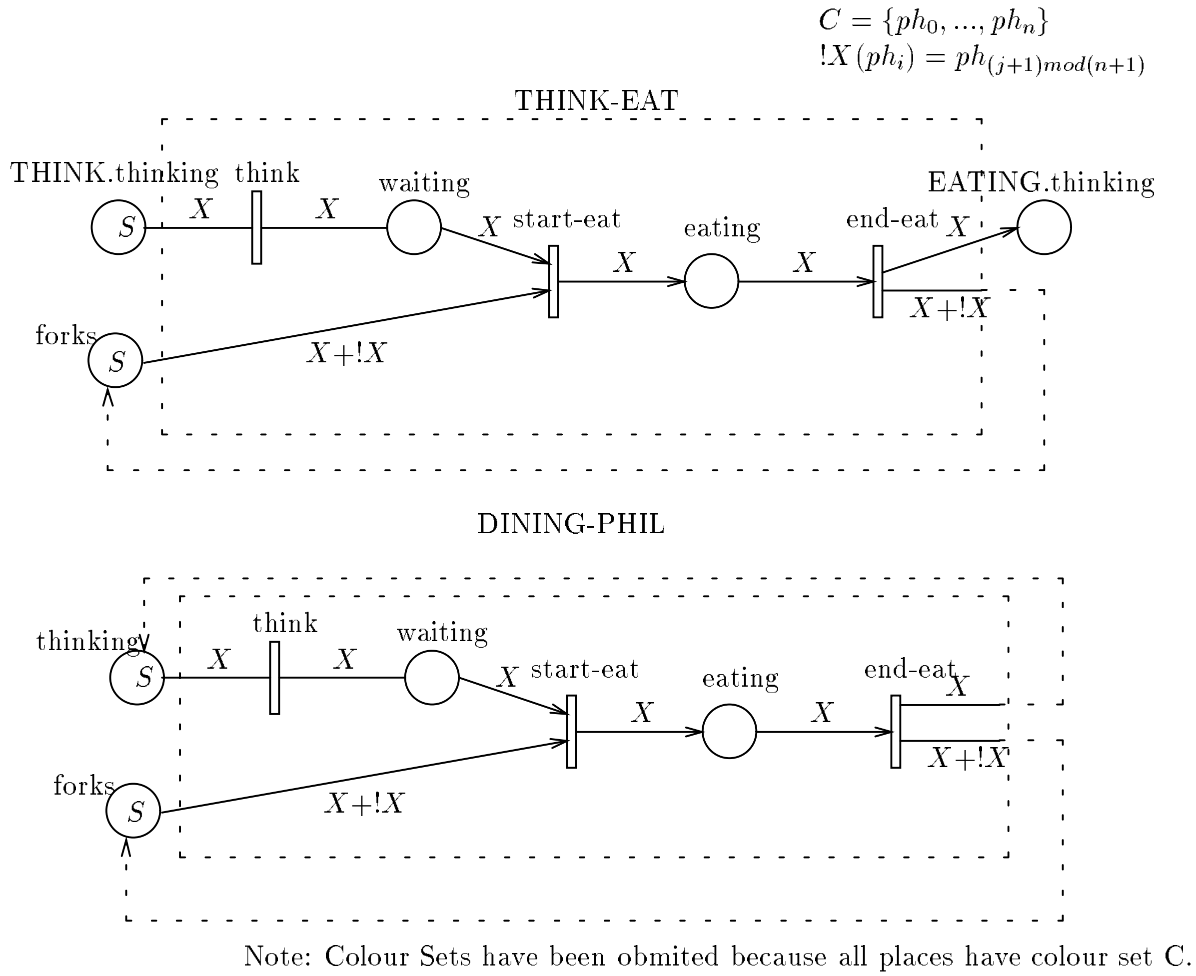



Figure 11: Applying the Compositional operations to form the Dining Philosophers Model from existing components(B)

# 5 Model Construction Guidelines

## 5.1 Identification of Colour Classes

Colour classes must be defined over the system as a whole to avoid ambiguity, repetitions and mismatches. If we allowed colour classes to be defined at the level of components we would have to support the creation of new colour classes, of new static sub-classes, and redefinition of colour functions and of transition predicates. All this in order to maintain the coherence of the system modelled. However, we should be able to identify the type of tokens, or elements that "flow" through the system, from a beginning, given the system's specification. In the example of the dining philosophers (Section 4) we could deduce, from the description of the problem, that philosophers and forks were the types of tokens to be represented. By further analysis of this types we deduced that one token type was sufficient.

The static colour classes of the basic colour classes must also be defined from the beginning. The un-coloured class (tokens with no colour) is tacitly considered to be within the set of the colour classes.

Having defined the colour classes we can assign colour domains to the places and transitions of the components of the system.

## 5.2 Hierarchy of the Operations

To guide the modeller in how to join a set of components to obtain the results desired, an ordering amongst the operations has been established. This ordering was defined according to the amount of information that is lost from the original sub-components when composed—the more information lost the stronger the operation. Based on this criterion the ordering is the following, from weakest to strongest:

**Independent Parallelism** : All the information about the subnets is preserved.

**Patient Communication, Impolite Communication and Untimed Synchronisation** : These generate new transitions, offering their information in the interface, preserving all information about the $ES$ and $FS$ of the components involved.

**Competing Parallelism** : All the information is preserved. The information about the places in the $ES'$ sets is kept through the fusion places.

**Choice (pre-selection)** : The information about the places in the $ES'$ sets is lost and information is augmented with the choice place.

**Closure** : Information about the final places that participate is lost.

**Sequence** : Information about the $ES'$ of the right hand component and on the $FS'$ of the left hand component is lost.

**Polite Communication and Timed Synchronisation** : These operators add a considerable number of places and transitions to the resulting component which are not visible in the resulting interface. The synchronised transitions are no longer visible after applying the operator.

The main inconvenience of this approach is that by first applying the independent parallelism operation we eliminate the possibility of applying operations that are only defined over pairs of components. We can also observe that the application of a choice operation can inhibit the application of a closing operation, because the information on the entry places that participate is lost. However, the contrary does not occur, because the closing operation preserves the $ES$. This observations have lead to the following changes in the priorities, now guided by the amount of information needed from the components in order to apply the operator and the amount of information preserved by each operator. From trongest to weakest we propose the following ordering:

**Closure** : Only requires information from one component. Preserves the $ES$.

**Competing Parallelism** : Requires information about the $ES$(s) of the participating component(s). All information about the entry places involved in the operator is kept through the fused places.

**Choice (pre-selection)** : Defined on different components. Requires information about the $ES$ of each component and loses this information once composition has occurred.

**Sequence** : Loses information about part or all of the final set of the left component and part or all of the entry set of the right hand component.

**Patient Communication, Impolite Communication and Untimed Synchronisation** : Can be defined over a single component or within a component or within components. It basically consists of superposition of transitions obtaining a new transition with a rate defined according to the synchronisation operator.

**Polite Communication and Timed Synchronisation** : The only information required is about the rate and arc functions of the synchronising transitions.

**Independent Parallelism** : All the information of the subnets is preserved.

## 5.3 Identifying the *bSWN*s and *cSWN*s required to model a system

This section proposes a series of steps to identify and define the *bSWN* and *cSWN* required to form the desired system model. We will assume that there exists a set of *cSWN* components representing commonly occurring functional modules, from which some can be re-used in the model constructed.

1. Divide the model, according to the system description, into functional modules.
2. Study the existing *cSWN* and see if the functional description of any of the module described coincides with a *cSWN* in the existing set.
3. For each module with no *cSWN* associated with it, identify its processes and analyse each of them in isolation.
4. For each process establish the type of information it requires: from where should it receive information, and what information it supplies and to whom.
5. Additionally analyse the timing characteristics of the process, keeping in mind that only exponentially distributed delays or untimed processes are supported.

## 5.4 Identifying how to compose the *cSWN* to form the model

Having defined the *bSWN*s and/or identified the *cSWN*s that model the different parts of the system, it is necessary to establish how, and in which order, they should be composed. Notice that after composing a pair of *cSWN*s we obtain another *cSWN* that might have to be composed, therefore the identification of which operator to apply is an iterative process until we obtain the model desired.

**Closing operation** : If in a *cSWN*, $L$, one or more places of $FS_L$ need to be fused to one or more places in $ES_L$, i.e. we need to feed information back from places in $FS_L$ into places of $ES_L$, then a closing operation is required. The closing operation only fuses a pair of places at a time, therefore if we want to fuse more than two places, we will have to perform as many closing operations as the number of places in $FS_L$ that need to be fused. We cannot fuse a single final place with several entry places, because when a final place participates in the closing operation the fused place is not included in the resulting $FS$.

**Choice and competing parallelism** : If we identify a pair of *cSWN*s, $L$ and $R$, with common information requirements, for which they have to compete, then there are two options for their composition. If which component the information is assigned to can be determined by a discrete probability distribution, then we define a choice composition between them. Otherwise, we apply the competing parallel composition. In the case of the choice operation, we must determine the probabilistic weight for each component. The parallel competing operation can be applied over a single component, but this is not the case for the choice operation.

**Sequential composition** : If the information offered by a place or a set of places in the $FS_L$ of a *cSWN*, $L$, is required by a place or a set of places in the $ES_R$ of a component $R$, then we should define a sequential composition over $L$ and $R$.

**Independent parallel composition** : If there is a pair of *cSWN*s, $L$ and $R$, for which:

- there is a third *cSWN*, $S$, which needs to be sequentially composed by the left with both $L$ and $R$, i.e. $L;S$ and $R;S$, and
- $L$ and $R$ do not require information from each other and do not require common input information,

then $L$ and $R$ need to be composed by a independent parallel composition.

**Synchronisation** : The synchronous communication between components must be defined according to the specifications of the system. The type of transitions (timed or immediate), the function each represents and the meaning given to their synchronisation (see Section 3.4.5) will determine which composition operation to define. We must keep in mind that the synchronisation operations work only on a pair of transitions at a time. Synchronisation can be defined between transitions of the same component.

## 5.5 Composing the system

Identifying which type of composition operations need to be applied is not enough information to guide the construction of the model. It is also necessary to establish an order in which these operations should be applied. We propose a set of guidelines based on the hierarchy suggested on the composition operations.

Given the initial set of components $G$ of a model, we propose the following iterative sequence of steps for the construction of the overall model of the system. The operations to be applied over the components are assumed to be defined according to the criteria offered in the previous subsection. These steps take into account that information about the $ES$, $FS$ and the $ST$ may be lost when applying certain operations.

1. Define $G' = G$.

2. If there is a component $A \in G'$ on which a **closing** operation needs to be applied, then: apply the operation over $A$ obtaining component $A'$, define $G = G' - \{A\} \cup \{A'\}$ and goto 1; otherwise continue.

3. If there is a **single** component $A \in G'$ on which a **parallel competing** operation needs to be applied, then: apply the operation over $A$ with itself obtaining component $A'$, define $G = G' - \{A\} \cup \{A'\}$ and goto 1; otherwise continue.

4. If there is a **pair** of components $B, C \in G$ on which a **parallel competing** composition needs to be applied, then: apply the operation over $B$ and $C$ obtaining component $D$, define $G = G' - \{B, C\} \cup \{D\}$ and goto 1; otherwise continue.

5. If there is a pair of components $B, C \in G$ for which a **choice** composition has been identified, then: apply the operation over $B$ and $C$ obtaining component $D$, define $G = G' - \{B, C\} \cup \{D\}$ and goto 1; otherwise continue.

6. If there is a pair of components $B, C \in G$ on which an **independent parallel** composition needs to be applied, then: apply the operation over $B$ and $C$ obtaining component $D$, define $G = G' - \{B, C\} \cup \{D\}$ and goto 1; otherwise continue.

7. Order the set of pairs of components in $G$, that need to be composed by a **sequential composition**, according to the number of places in the $ES$ of the right-hand component with initial marking different from zero. We consider parametric markings with values not assigned as different from zero. If the set is not empty, then take the pair $(L_i, R_i)$ with the lowest count and apply the operation to them obtaining component D, define $G = G' - \{L_i, R_i\} \cup \{D\}$ and goto 1; otherwise continue.

8. If there is a **single** component $A \in G'$ on which a **patient communication**, an **impolite communication** or an **untimed synchronisation** operation needs to be applied, then: apply the operation over $A$ with itself obtaining component $A'$, define $G = G' - \{A\} \cup \{A'\}$ and goto 1; otherwise continue.

9. If there is a **pair** of components $B, C \in G$ for which a **patient communication**, and **impolite communication** or an **untimed synchronisation** operation needs to be applied, then: apply the operation over $B$ and $C$ obtaining component $D$, define $G = G' - \{B, C\} \cup \{D\}$ and goto 1; otherwise continue.

10. If there is a **single** component $A \in G'$ on which a **polite communication** or a **timed synchronisation** operation needs to be applied, then: apply the operation over $A$ with itself obtaining component $A'$, define $G = G' - \{A\} \cup \{A'\}$ and goto 1; otherwise continue.

11. If there is a **pair** of components $B, C \in G$ for which a **polite communication** or a **timed synchronisation** operation needs to be applied, then: apply the operation over $B$ and $C$ obtaining component $D$, define $G = G' - \{B, C\} \cup \{D\}$ and goto 1; otherwise continue.

12. Compose all remaining components in $G'$ with the use of **independent parallel composition** in an associative manner. The resulting component constitutes the final model.

## 5.6 Model of a Multiprocessor Architecture

We will apply the guidelines described to the example of a Multiprocessor system presented in [21], and later studied in [12] for the application of the $SWN$ formalism (see Figure 12).

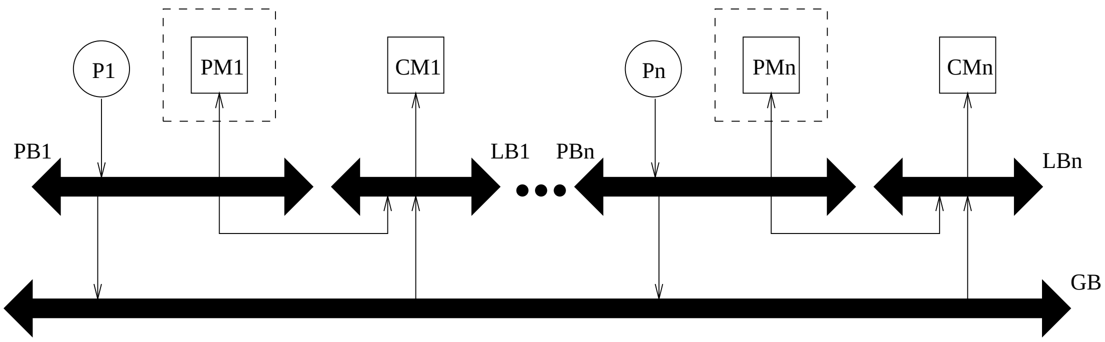



Figure 12: The multiprocessor architecture been studied

The multiprocessor architecture is composed of a group of processors $(p_1, ..., p_n)$. Each processor $p_i$ is associated with a local memory composed of two sections, a *private memory* $(PM)$ and a *common memory* $(CM)$. The $PM$ can only be accessed by the corresponding processor through its *private bus* $(PB)$. The $CM$ of a processor $p_i$ $(CM_i)$ is accessible to all processors in the system. The associated processor accesses it through its $PB$ together with its *local bus* $(LB)$. Other processors must access it by using the *global bus* $(GB)$ plus the $LB$ of the destination $CM$ module. It is assumed that the external access requests to $CM$ modules have priority over the local $CM$ accesses and cause their preemption.

Let us first analyse the colour classes of tokens of the system to be modelled. The elements that need to access the different components of the system are processors. These constitute our first colour class. We do not distinguish among different types of processors, therefore we do not define any static subclass over this class. Each processor has an associated local common memory, therefore the same colour class (processors) can be used in order to identify the common memory block that a certain processor wants to access. The same deduction applies for the local buses. The global bus is unique and can therefore be represented by a un-coloured token. As we have said, the un-coloured tokens are considered to be tacitly included in the definition of the basic colour classes, therefore our set of basic colour classes $C$ is defined as:

$C = \{Proc\}$ where $Proc = \{p_1, \cdots, p_n\}$

In order to create the $SWN$ model we identify the functions involved in the system:

1. Generate memory request
2. Access $PM$
3. Request local $CM$ module access
4. Request remote $CM$ module access
5. Access local $CM$ module through local $LB$
6. Request $GB$ for remote $CM$ access
7. Access remote $CM$ module through remote $LB$

Let us now analyse the requirements and results of each of these functions to define the *bSWN*s of the model:

1. We will assume that a processor consumes some time processing its data (it is *active*) and then generates a memory request. This is represented by a timed transition. As input it has the active processors and will generate as output a memory request (see Figure 13.a).
2. As stated in the problem description a processor that wants to access its private memory can directly do it through the use of its $PB$. Therefore the only input required for this function is a requirement from the local processor to access the $PM$. The output of this function would be the processor which is now free to process further memory requirements. As there is only one private memory module per processor, and only the associated processor can access it, the identifier of the processor also acts as

the identifier of the PM module requested (see Figure 13.b). The transition associated with this process is timed, where the delay represents the time that the processor consumes accessing its $PM$.

3. If a processor $p_j$ wants to access the $CM$ module $i$ ($CM_i$), then if $j = i$, i.e. it wants to access the local module, it should require access to its $LB$ ($LB_j$). This is accomplished by defining a immediate transition with predicate $j = i$, that has as input a place with processors that require a $CM$ module, and produces as output a request to access the local $CM$ module, i.e. tokens of the type $\langle i, j \rangle$ where $j = i$ (see Figure 13.c).

4. A processor may also request a remote $CM$ module access. In order to generate this request, the process needs as input a processor that wants to access a $CM$ and will generate a request for an external $CM$ access . This is accomplished by adding the predicate $j \neq i$ to the transition associated with the process, where $j$ is the identifier of the processor making the request and $i$ is the identifier of the $CM$ module it wants to access. As this is a decision process the transition representing it has zero delay (immediate transition). The output of the process is a request for a remote $CM$ access (see Figure 13.d).

5. A processor can access its local $CM$ module if it has requested it and if its $LB$ is available. To check these two conditions this process has as input. The place where requests for all $LB$ accesses are located and the place where we have information about all the $LB$s available. In order to verify that the bus requested is available we add the predicate $j = i$ to the transition associated with this process, where $j$ is the identifier of the requesting processor and $i$ represents both the $CM$ requested and the $LB$ required. After the processor operates over its local memory, represented by a timed transition ($LocalLB$, see Figure 13.e), it frees its $LB$ and returns to *active*, where it can request further memory access.

6. If a processor $p_j$ is requesting a remote $CM$ access, then it must first acquire the system's $GB$. The access to the $GB$ is controlled by a queue, if the resource is available it grants it immediately to a requesting processor, otherwise the processor has to wait for the $GB$ to become available. Therefore the $bSWN$ that represents the queue for access of the $GB$, has as input a place with tokens representing requests for remote $CM$ access and a place where the tokens represent the availability of the $GB$ (place $GB$ in Figure 13.f). The queueing discipline employed will depend on how we determine which of the instantiations of the transition of the $bSWN$ will fire, i.e. in which order do we evaluate the the instantiations of the transition with the elements of the colour class *Proc*. We will assume that this is done at

random. If the $GB$ is available the transition will fire, generating as output a request for the local bus $LB$ of the remote $CM$ module.

7. Once a processor has been granted access to the $LB$ of a remote $CM$ module, then it can execute its remote $CM$ operation. This is represented by a timed transition, whose delay reflects the amount of time that the processor takes in processing its memory requirements. As input it needs a remote $CM$ request for which the access to the associated $LB$ has been granted. Once the processor has performed its memory access operation it can then liberate the $GB$ (through place $GB$, see Figure 13.g), free the $LB$ bus associated with the remote $CM$ module accessed and and returns to *active*, where it can request further memory access.

8. In order to reflect the preemption property of the remote $CM$ requests over the local $CM$ requests, we incorporate a *bSWN* not directly associated with any of the functions previously described. This process is represented as an immediate transition, so that when a request of access to a $LB$ from a remote processor arrives this will be processed with priority over the local processor's request. To identify that the request is remote the transition associated with this *bSWN* has the predicate $i \neq j$. The inputs of the transition are the local buses and information about which processor is requesting access and to which $CM$ module (see Figure 13.h).

Once we have identified the *bSWN*s and/or *cSWN*s that participate in the system modelled, we can proceed to compose them following the guidelines described in Sections 5.4 and 5.5.

Based on the *bSWN*s presented in Figure 13, we execute the following steps to obtain the model of the Multiprocessor System in a compositional manner. To distinguish two places which originally have the same name, we use a dot notation, prefixing to the name of each of the places involved, the name of the component to which the place belongs to. We do not talk about a renaming operation because it is not defined as such, on the contrary it is considered as an "automatic" operation, which requires no intervention from the modeller.

1. $A = CL(d)$ with $\Theta(LBs') = LBs$. Perform a close operation over component $d$. See Figure 14.A.

2. $B = A|_c g$ with $\Lambda(A.LBs) = g.LBs$ and $\Lambda(A.ReqLB) = g.ReqLB$. Compose components $A$ and $g$ by a competing parallelism operation. See Figure 14.B.

3. $C = b|_c c$ with $\Lambda(b.CMacc) = c.CMacc$. Compose components $b$ and $c$ by a competing parallelism operation. See Figure 14.C.

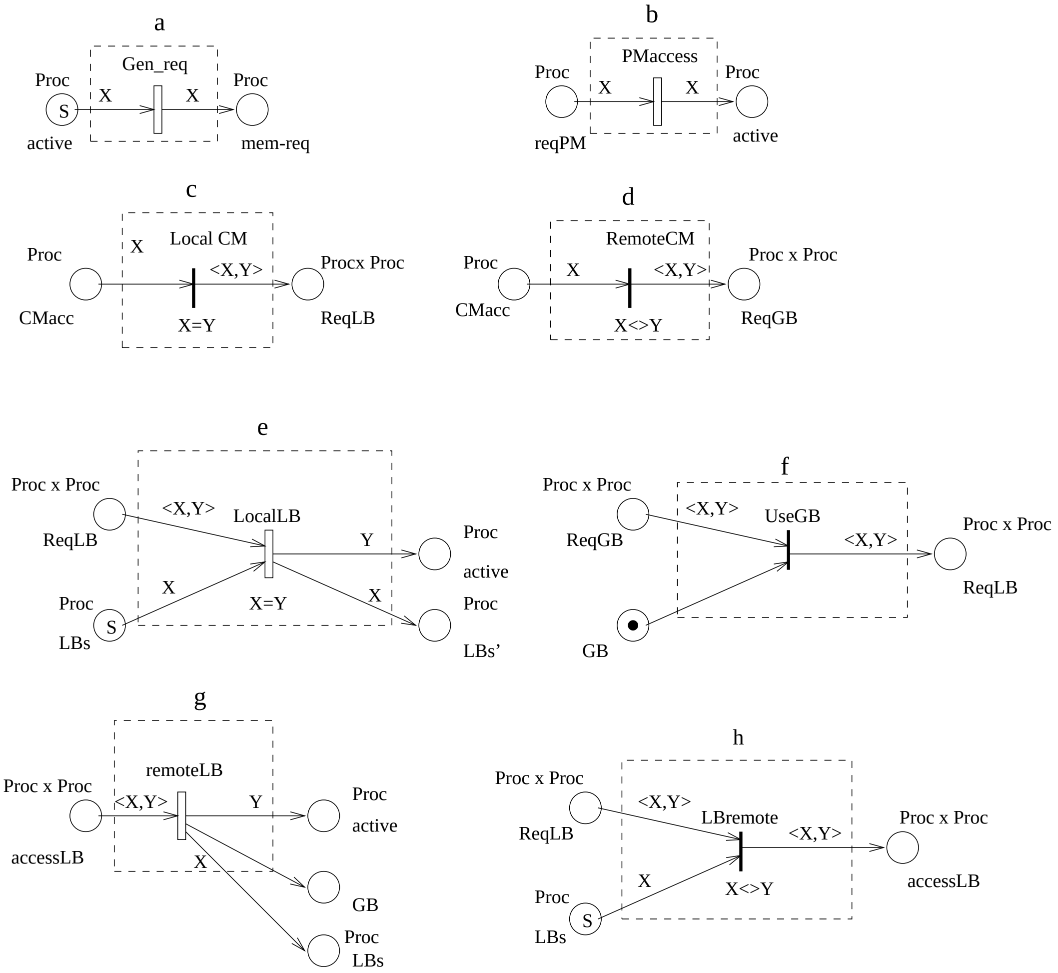



Figure 13: Basic SWN components of the Multiprocessor model

4. $D = a + C\{1/3, 2/3\}$ with $Choice(a) = reqPM$ and $Choice(C) = CMacc$. Define a choice operation over components $a$ and $C$. The weight values assigned are to give equal probability of occurrence to all types of memory request. We rename the choice place *mem-req*. See Figure 14.D.

5. $E = h; D$ with $\Gamma(h.\textit{mem-req}) = D.\textit{mem-req}$. Sequentially compose component $h$ with component $D$. Place $D.active$ is renamed $active'$ in component $E$. See Figure 14.E.

6. $F = CL(E)$ with $\Theta(active') = active$. Perform a close operation over component $D$. See Figure 15.F.

7. $G = F; e$ with $\Gamma(F.ReqGB) = e.ReqGB$. Sequentially compose component $E$ with component $e$. Place $e.ReqLB$ is renamed as $ReqLB'$ in component

$G$. See Figure 15.G.

8. $H = G; B$ with $\Gamma(G.ReqLB) = B.ReqLB$ and $\Gamma(G.ReqLB') = B.ReqLB$. Sequentially compose component $F$ with component $B$, by defining $\Gamma$ : $\{G.ReqLB, G.ReqLB'\} \rightarrow B.ReqLB$. Place $B.active$ is renamed $active'$ in component $H$. See Figure 16.H .

9. $I = CL(H)$ with $\Theta(active') = active$. Perform a close operation over component $d$. See Figure 16.I.

10. $J = I; f$ with $\Gamma(I.accessLB) = f.accessLB$. Sequentially compose component $H$ with component $f$, by defining $\Gamma(I.accessLB) = f.accessLB$. Places $f.GB$, $f.active$ and $f.LBs$ are renamed to $GB'$, $active'$ and $LBs'$, respectively in component $J$. See Figure 17.J.

11. For the sake of space we have combined the following three operations as one step from component $J$ to the final model presented as component $FINAL$ (See Figure 17).

    - $K = CL(J)$ with $\Theta(active') = active$.
    - $L = CL(K)$ with $\Theta(GB') = GB$.
    - $FINAL = CL(L)$ with $\Theta(LBs') = LBs$.

Let us now compare the *cSWN* model obtained with the one presented in [22] (See Figure 18). First of all it is important to point out a small difference in the interpretation of the system. In [22], they consider a processor to be *active* executing in its $PM$ and after some time to generate a request to access a memory module (either its $PM$, its local $CM$ or a remote $CM$). If the request is for the $PM$ then it is immediately granted and the processor returns to its active status executing in its $PM$. The $PM$ access and the memory request are joined in a single timed transition. In the model presented in this report, we separate these functions into two timed transitions. One, *mem-req*, representing the period in which the processor operates over the data and generates a memory request; and another, *PMaccess*, to represent the time that the processor consumes accessing its $PM$.

Leaving aside this interpretation difference, we find that the models obtained are very similar. We can observe that in our model there are more immediate transitions than in the model presented in [22]. One of the reasons behind this is the fact that the choice operation (as it has been defined) only allows us to make a choice between two *cSWN*s at a time. Therefore, the selection of which type of request is generated ($PM$, local $CM$ or remote $CM$), is made in two steps: first we distinguish between $PM$ or $CM$, and then distinguish between local and remote $CM$. This type of drawback can always be overcome by the multiplication of the weights of the immediate transitions within a tree of choice operations. In this case we could eliminate transition *CMreq* and the place *CMacc*, add an arc from the place *mem-req* to the transition *LocalCM*, and another to the transition *RemoteCM*, and assign to *LocalCM* and *RemoteCM* weight 1/3.

Although the model obtained can be considered as "naive" with respect to the number of immediate transitions employed, it clearly reflects the functional behaviour of the system. The functions of the system are modelled as basic components. The combination of functions can be created by the adequate composition of the components reflecting these functions. Under a state-based construction approach, composition does not come about in a straightforward manner. Trying to see a component as representing a set of states or sub-states, to then compose it with other components in order to form more complex states, does not logically or easily emerge from the analysis of a system. In general the creation of Petri net models depends very much on the abilities of the modeller. We propose a series of steps that guide the modeller in the construction process, in order to create a simple but understandable model. We can obtain a more compact model by applying net reduction methods, such as the elimination of unnecessary immediate transitions.

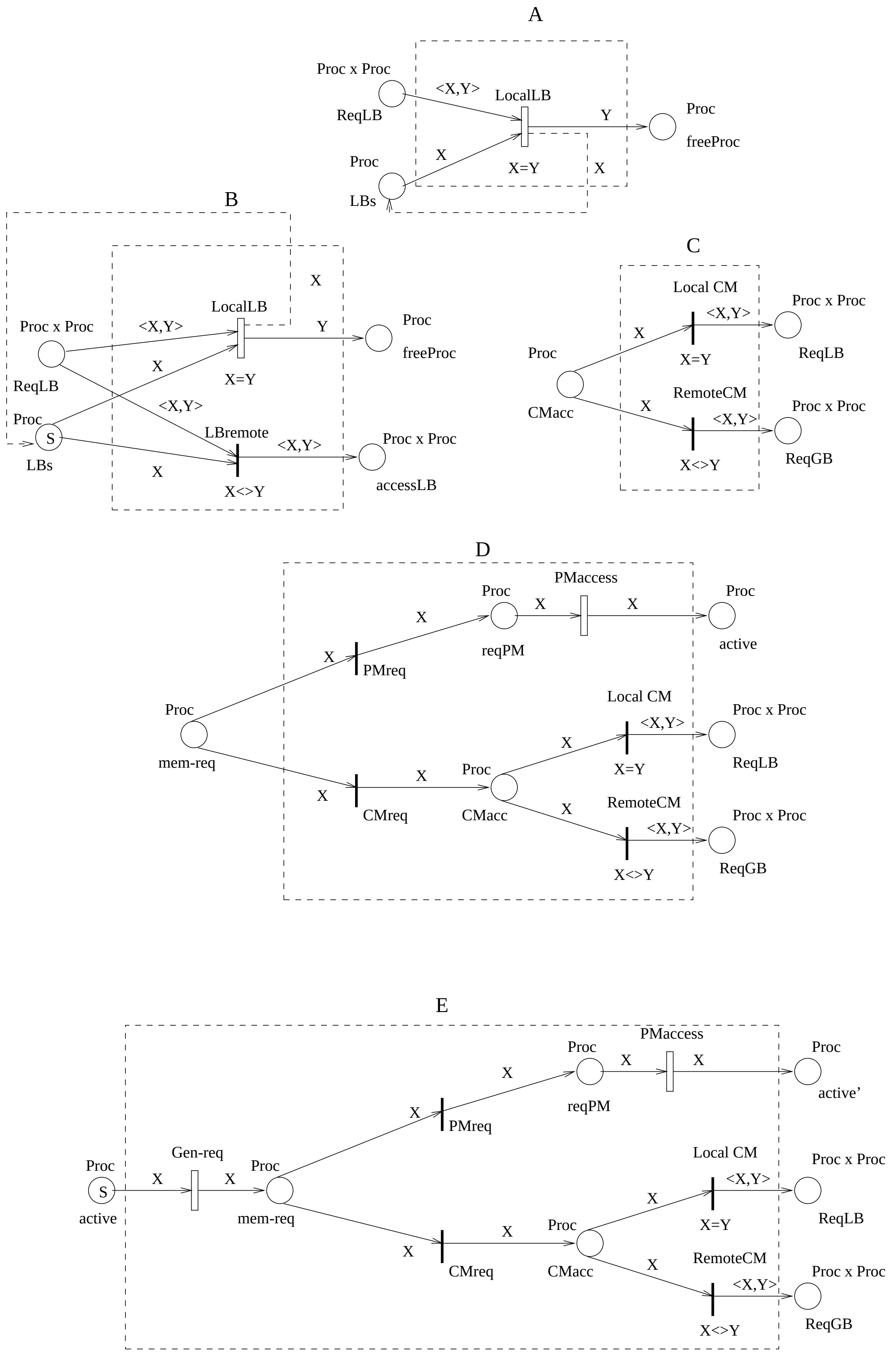



Figure 14: Creating the Multiprocessor model in a compositional manner (A)

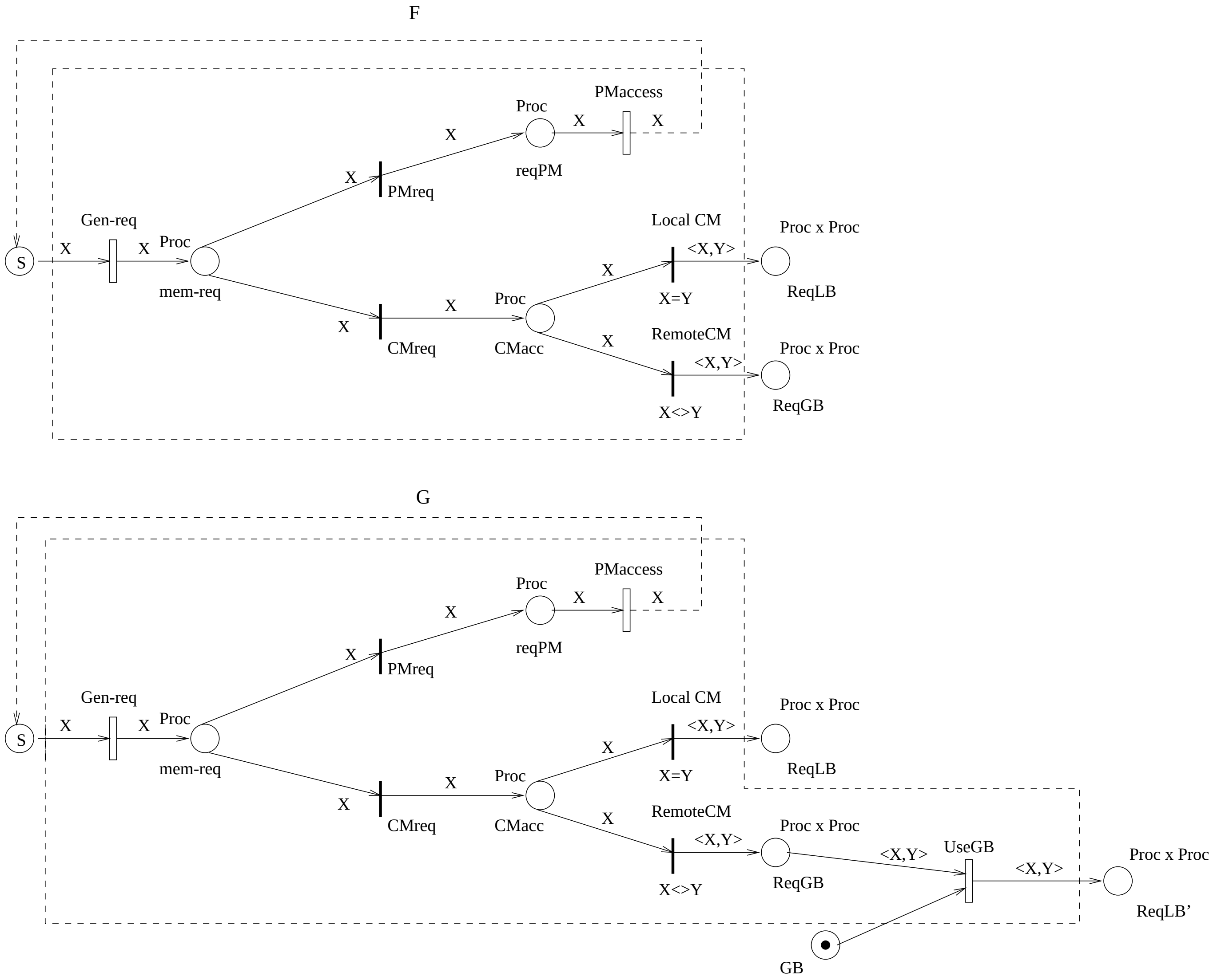



Figure 15: Creating the Multiprocessor model in a compositional manner (B)

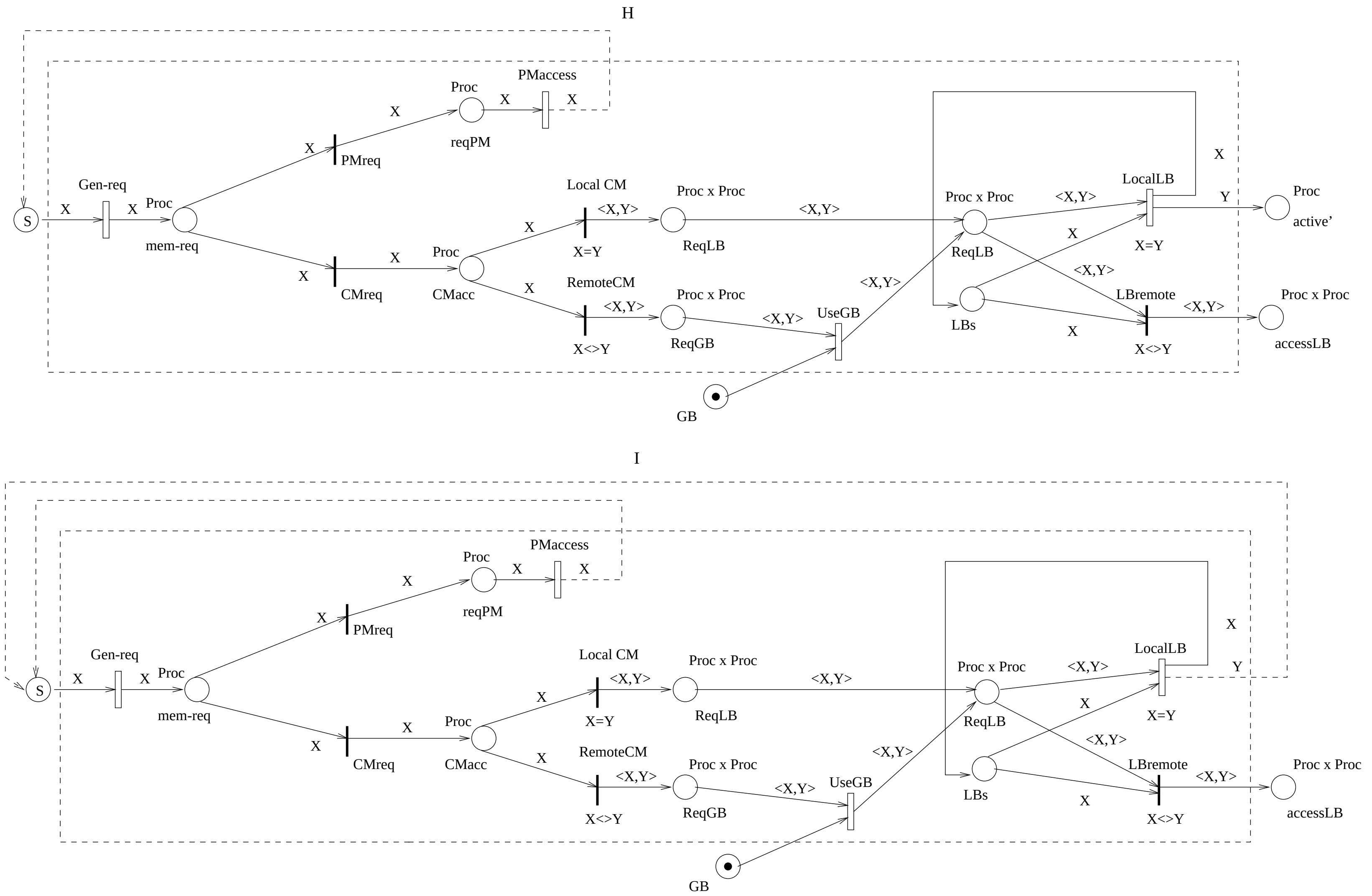



Figure 16: Creating the Multiprocessor model in a compositional manner (C)

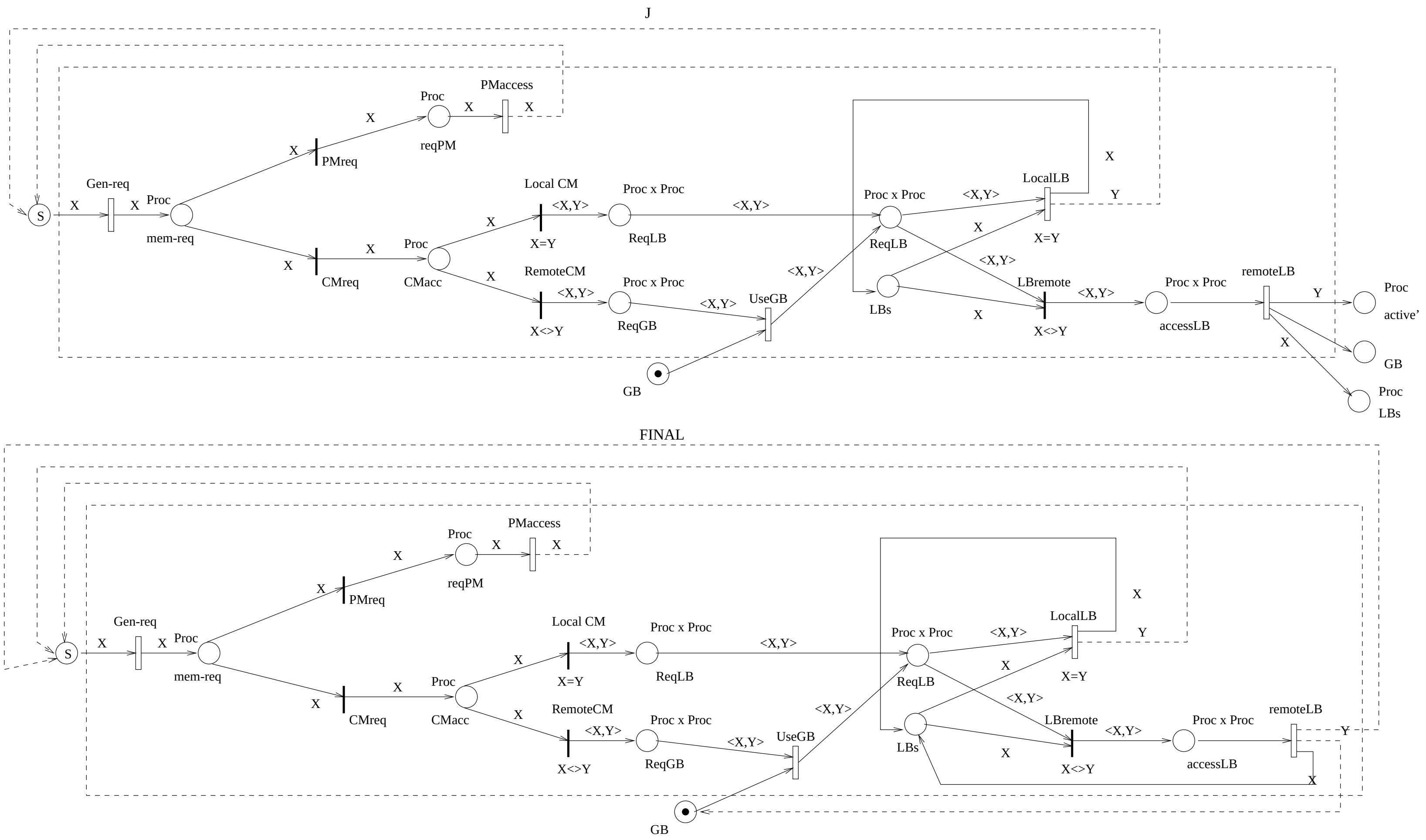
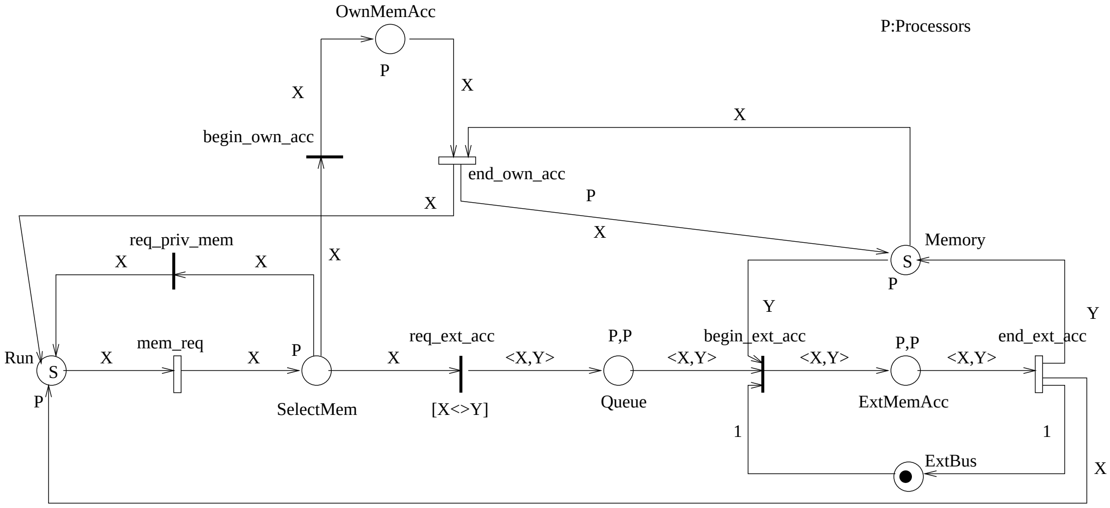
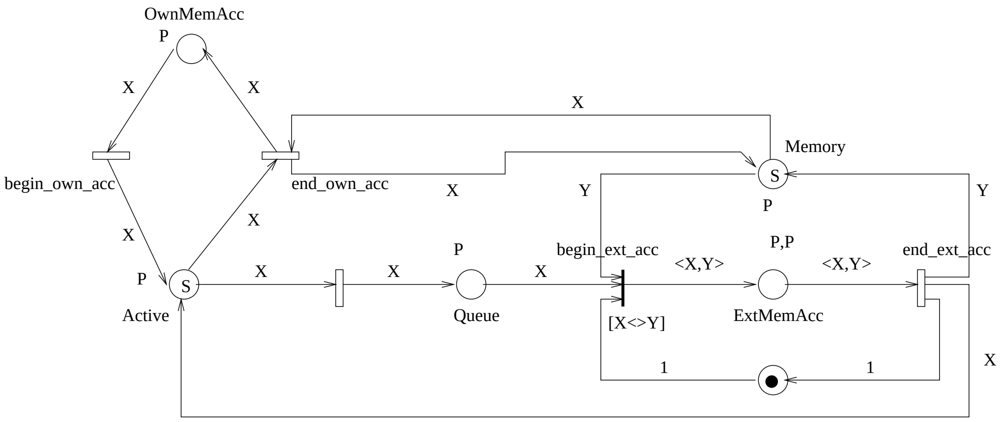



Figure 17: Creating the Multiprocessor model in a compositional manner (D)




a) An intuitive model.




b) A more compact WN model

Figure 18: Multiprocessor model presented by Chiola *et al.*

## 6 Conclusions and Future Work

In this report we have defined a set of composition operations for the creation of SWN models from SWN components. The operations presented differ from existing compositional PN formalisms: they preserve the functional structure of the model and support several types of communication between components. In this way we support the modelling of distributed and parallel systems where both synchronous and asynchronous communication is required. SWN have been used because of the intrinsic lumping facilities that the formalism offers.

We suggest a series of steps to identify and define the *basic SWN*s in the system modelled. Once all *compose-able SWN*s (*cSWN*) have been identified we suggest a series of guidelines to determine the composition operations that have to be used to correctly compose the *cSWN*s. However, determining which type of composition operations need to be applied is not enough information to guide the construction of the model. It is also necessary to establish an order in which these operations should be applied. A set of guidelines for the compositional construction of SWN models have been suggested. This set is guided by an hierarchy defined on the compositional operations based on the amount of information required/lost and/or preserved from each component by each operation.

Other work now in progress is investigating the structural properties preserved by each compositional operation. In this way we will be able to derive knowledge of the structural properties of a component from the structural properties of its sub-components. We derive the incidence matrix of a *cSWN* from the incidence matrix of its sub-components and by knowing the compositional operation employed. Knowing how the incidence matrix can be constructed, we study how from the generative family of positive flows of the sub-components, we can derive the generative family of the resulting *cSWN*.

## References


[1] Milner R. *Communication and Concurrency.* Prentice-Hall, 1989.

[2] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[3] J. Kramer, J. Magee, and A. Finkelstein. A Constructive Approach to the Design of Distributed Systems. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, Paris, France, May 1990. IEEE Computer Society Press.

[4] Y. Souissi and G. Memmi. Composition of nets via a communication medium. *Advances in Petri Nets 1990*, LNCS 483:457–470, 1990.

[5] A. Valmari. Compositional State Space Generation. *Advances in Petri Nets 1993*, LNCS 674:427–457, 1993.

[6] W. Volger. Failures Semantics and deadlocking of Modular Petri nets. *Acta Informatica*, 26:333–348, 1989.

[7] C. Sibertin-Blanc. A Client-Server Protocol for the Composition of Petri Nets. In *Proceedings of the 14th International Conference on Applications and Theory of Petri Nets*, pages 377–396, Illinois, U.S.A., June 1993. Springer-Verlag.

[8] W. Sanders and J. Meyer. Reduced Base Model Construction Methods for Stochastic Activity Networks. *IEEE Journal on Selected Areas in Communication*, 9:25–36, January 1991.

[9] S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–36, 1993.

[10] S. Donatelli. Superposed generalised stochastic Petri nets: definition and efficient solution. In *Proceedings of the 15th International Conference on Applications and Theory of Petri Nets*, volume LNCS 815, Zaragoza, Spain, June 1994. Springer-Verlag.

[11] P. Buchholz. Aggregation and Reduction Techniques for Hierarchical GCSPNs. In *Proceedings of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE Computer Society Press.

[12] G. Chiola, C. Dutheillet, G. Franceschinis, and Haddad S. Stochastic well-formed coloured nets and multiprocessor modelling applications. *High-Level Petri Nets, Theory and Application*, 1991.

[13] J. Hillston. A Compositional Approach to Performance Modelling. PhD Thesis CST-107-94, Department of Computer Science, University of Edinburgh, April 1994.

[14] N. Gotz, U. Herzog, and M. Rettelbach. TIPP - Introduction and application to protocol performance analysis. Technical report, University of Erlangen, March 1993.

[15] M. Ribaudo. On the Relationship between Stochastic Petri Nets and Stochastic Process Algebras. Dottorato di ricerca in informatica, Dipartamento di Informatica, Universita di Torino, May 1995.

[16] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modeling with Generalized Stochastic Petri Nets*. John Wiley & Sons Ltd., 1995.

[17] K. Jensen. *Coloured Petri Nets.* Springer-Verlag, 1992.

[18] P. Buchholz. A Hierarchical View of GCSPNs and Its Impact on Qualitative and Quantitative Analysis. *Journal of Parallel and Distributed Computing*, 15:207–224, July 1992.

[19] G. Findlow. Obtaining deadlock-Preserving Skeletons for Coloured Nets. In K. Jensen, editor, *Proceedings from the 13th International Conference on Application and Theory of Petri Nets*, pages 173–192, Sheffield, UK, June 1992.

[20] J. Hillston. The Nature of Synchronisation. In U. Herzog and M. Rettelbach, editors, *2nd Workshop on Process Algebras and Performance Modelling*, Erlangen, Germany, 1994.

[21] M. Ajmone Marsan, G. Balbo, and G. Conte. Comparative performance analysis of single bus multiprocessor architectures. *IEEE Transactions on Computers*, C-31(12), December 1982.

[22] 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*, 19:89–107, February 1993.