ISSN 0105-8517

Coloured Petri Nets: A High Level Language for System Design and Analysis

Kurt Jensen

DAIMI PB – 338
November 1990

COMPUTER SCIENCE DEPARTMENT
AARHUS UNIVERSITY
Ny Munkegade, Building 540
DK-8000 Aarhus C, Denmark

This paper is also published in: G. Rozenberg (Ed.): Advances in Petri Nets 1990, Lecture Notes in Computer Science, Springer-Verlag.

Coloured Petri Nets: A High Level Language for System Design and Analysis

Kurt Jensen
Computer Science Department, Aarhus University
Ny Munkegade, Bldg. 540
DK-8000 Aarhus C, Denmark

Phone: +45 86 12 71 88
Telefax: +45 86 13 57 25
Telex: 64767 aausci dk
E-mail: kjensen@daimi.aau.dk

Abstract

This paper describes how Coloured Petri Nets (CP-nets) have been developed – from being a promising theoretical model to being a full-fledged language for the design, specification, simulation, validation and implementation of large software systems (and other systems in which human beings and/or computers communicate by means of some more or less formal rules).

First CP-nets are introduced by means of a small example and a formal definition of their structure and behaviour is presented. Then we describe how to extend CP-nets by a set of hierarchy constructs (allowing a hierarchical CP-net to consist of many different subnets, which are related to each other in a formal way). Next we describe how to analyse CP-nets, how to support them by various computer tools, and we also describe some typical applications. Finally, a number of future extensions are discussed (of the net model and the supporting software).

The non-hierarchical CP-nets in the present paper are analogous to the CP-nets defined in [35] and the High-level Petri Nets defined in [33]. In all three papers CP-nets (and HL-nets) have two different representations: The *expression representation* uses arc expressions and guards, while the *function representation* uses linear functions between multi-sets. Moreover, there are formal translations between the two representations (in both directions). In [33] and [35] we used the expression representation to describe systems, while we used the function representation for all the different kinds of analysis. It has, however, turned out that it only is necessary to turn to functions when we deal with invariant analysis, and this means that we now use the expression representation for all purposes – except for the calculation of invariants. This change is important for the practical use of CP-nets – because it means that the function representation and the translations (which are a bit mathematically complex) no longer are parts of the basic definition of CP-nets. Instead they are parts of the invariant method (which anyway demands considerable mathematical skills).

The development of CP-nets has been supported by several grants from the Danish National Science Research Council.

Contents

1. Informal Introduction to Non-Hierarchical CP-nets

High-level nets, such as Coloured Petri Nets (CP-nets) and Predicate/Transition Nets are now in widespread use for many different practical purposes.[1] The main reason for the large success of these kinds of net models is that they – *without loosing the possibility of formal analysis* – allow the modeller to make *much more succinct and manageable descriptions* than can be produced by means of low-level nets (such as Place/Transition Nets and Elementary Nets). In high-level nets the complexity of a model can be divided between the net structure, the net inscriptions and the declarations. This means that it is possible to handle the description of much larger and more complex systems. It also means that we can describe simple data manipulation (such as the addition of two integers) by means of arc expressions (such as x+y) – instead of having to describe this by a complex set of places, transitions and arcs. The step from low-level nets to high-level nets can be compared to the step from assembly languages to modern programming languages with an elaborated type concept: In low-level nets there is only one kind of token and this means that the state of a place is described by an integer (and in many cases even by a boolean). In high-level nets each token can carry a complex information (which e.g. may describe the entire state of a process or a data base).[2]

However, looking at the history of high-level programming languages, it is obvious that their success also to a very large degree depends upon issues that do not concern typing. In particular, the development of subroutines and modules has played a key role, because they have made it possible to divide a large description into smaller units which can be investigated more or less independently of each other. In fact, the absence of compositionality has been one of the main critiques raised against Petri net models. To meet this critique hierarchical CP-nets have been developed. In this net model it is possible to create a number of individual CP-nets, which then can be related to each other in a formal way – i.e. in a way which has a well-defined behaviour and thus allows formal analysis.

The remaining parts of this chapter contains an informal introduction to non-hierarchical CP-nets and their behaviour.

1.1 A simple example of a non-hierarchical CP-net

The non-hierarchical CP-net in Fig. 1 describes a system in which a number of processes compete for some shared resources. As in all other kinds of Petri nets there is a set of places (drawn as circles/ellipses) and a set of transitions (drawn as rectangles). The places and their tokens represent states, while the transitions represent state changes. However, each place may contain several tokens and each of these carries a data value – which may be of arbitrarily complex type (e.g. a record where the first

1 A selection of references can be found in section 6.5.

2 We shall in this paper not use any more space to compare high-level and low-level Petri nets. The reason is that we primarily are interested in the practical applications of Petri nets – and in this field the superiority of high-level nets is now generally accepted. A more detailed comparison of high-level and low-level nets can be found in [20] and [22].

field is a real, the second a text string, while the third is a list of integer pairs). The data value which is attached to a given token is referred to as the **token colour**.

In Fig. 1 there are two kinds of processes: three q-processes start in state A and cycle through five different states (A, B, C, D and E), while two p-processes start in state B and cycle through four different states (B, C, D and E). Each of these five processes is represented by a token – where the token colour is a pair such that the first element tells whether the token represents a p-process or a q-process while the second element is an integer telling how many full cycles that process has completed. In the initial marking, there are three (q,0)-tokens at place A and two (p,0)-tokens at place B.

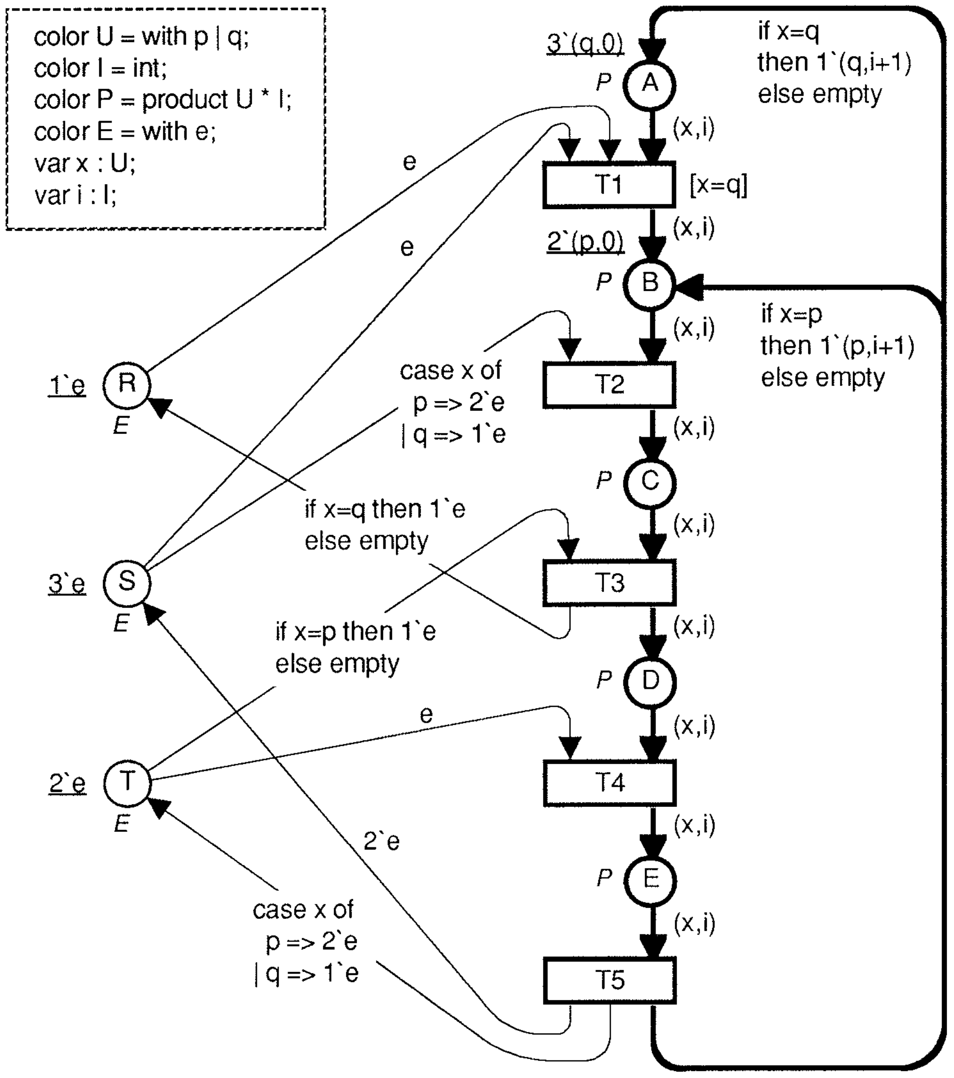


Figure 1. Non-hierarchical CP-net describing a simple resource allocation

There are three different kinds of resources: one r resource, three s resources and two t resources (each resource is represented by an e-token, on R, S or T). The arc expressions tell how many resources the different kinds of processes reserve/release. As an example, "case x of p=>2`e | q=>1`e" (at the arc from S to T2) tells that a p-process needs two s resources in order to go from state B to C, while a q-process only needs one.[3] Analogously, "if x=q then 1`e else empty" (at the arc from T3 to R) tells that each

[3] The operator ` takes an integer n and a colour c and it returns the multi-set that contains n appearances of c (and nothing else).

q-process releases an r resource when it goes from state C to D, while a p-process releases none.[4] It should be noticed that the processes in this system neither consume nor create tokens (during a full cycle the number of releases matches the number of reservations). Now let us take a closer view of the CP-net in Fig. 1. It consists of three different parts: the **net structure**, the **declarations** and the **net inscriptions**.

The net structure is a directed graph with two kinds of nodes, **places** and **transitions**, interconnected by **arcs** – in such a way that each arc connects two different kinds of nodes (i.e. a place and a transition).[5] In Fig. 1 the right hand part of the net (describing how processes change between different states) is drawn with thick lines. This distinguishes it from the rest of the net (describing how resources are reserved and released). It should, however, be stressed that such graphical conventions have no formal meaning. The only purpose is to make the CP-net more readable for human beings.

The declarations in the upper left corner tell us that we in this simple example have four **colour sets**, (U, I, P, and E) and two **variables** (x and i). The use of colour sets in CP-nets is analogous to the use of types in programming languages: Each place has a colour set attached to it and this means that each token residing on that place must have a colour (i.e. attached information) which is a member of the colour set. Analogously to types, the colour sets not only define the actual colours (which are members of the colours sets), they also define the operations and functions which can be applied to the colours. In this paper we shall define the colour sets using a syntax that is similar to the way in which types are defined in most programming languages. It should be noticed that a colour set definition often implicitly introduces new operators and functions (as an example the declaration of a colour set of type integer introduces the ordinary addition, subtraction, and multiplication operators). In our present example, the colour set U contains two elements (p and q) while the colour set I contains all integers.[6] The colour set P is the set of all pairs, where the first component is of type U while the second is of type I. Finally, the colour set E only contains a single element – and this means that the corresponding tokens carry no information (often we think of them as being "ordinary" or "uncoloured" tokens).

Each net inscription is attached to a place, transition or arc. In Fig. 1 places have three different kinds of inscriptions: **names**, **colour sets** and **initialization expressions**, transitions have two kinds of inscriptions: **names** and **guards**, while arcs only have one kind of inscription: **arc expressions**. All net inscriptions are positioned next to the corresponding net element – and to make it easy to distinguish between them we write names in plain text, colour sets in italics, while initialization expressions are underlined and guards are contained in square brackets.

Names have no formal meaning. They only serve as a mean of identification that makes it possible for human beings and a computer systems to refer to the individual places and transitions. Names can be omitted and one can use the same name for several nodes (although this may create confusion). As explained above each place must have a

4 *empty* denotes the empty multi-set.

5 Such a graph is called a bipartite directed graph.

6 To be more precise, I only contains the integers in the interval MinInt..MaxInt – where MinInt and MaxInt are determined by the implementation of the Integer data type. In general, each colour set is demanded to be finite, although it (as I) may have very many elements.

colour set and this determines the kind of tokens which may reside on that place. The initialization expression of a place must evaluate to a multi-set over the corresponding colour set. Multi-sets are analogous to sets except that they may contain multiple appearances of the same element. In the case of CP-nets, this implies that two tokens on the same place may have identical colours. By convention we omit initialization expressions which evaluate to the empty multi-set.

The guard of a transition is a boolean expression which must be fulfilled before the transition can occur. By convention we omit guards which always evaluate to true. The arc expression of an arc is an expression, and it may (as the guard) contain variables, constants, functions and operations that are defined in the declarations (explicitly or implicitly). When the variables of an arc expression are bound (i.e. replaced by colours from the corresponding colour sets) the arc expression must evaluate to a colour (or a multi-set of colours) that belong to the colour set attached to the place of the arc. When the same variable appears more than once, in the guard/arc expressions of a *single* transition, all these appearances must be bound to the same colour. In contrast to this appearances, in the guard/arc expressions of *different* transitions, are totally independent, and this means that they may be bound to different colours. As explained in the sequel, a CP-net may have several other kinds of inscriptions (e.g. used to describe hierarchical relationships and time delays).

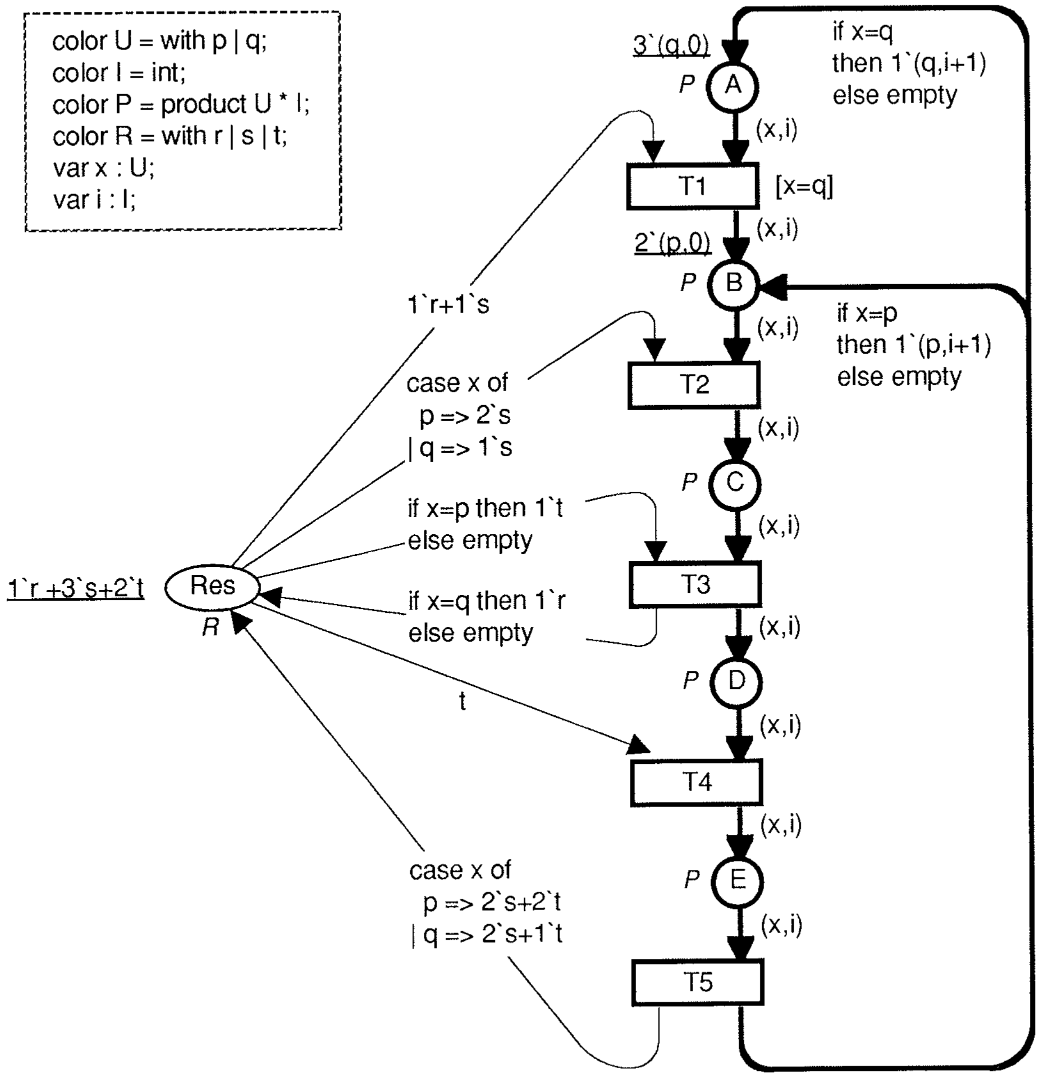


Figure 2. A slightly different CP-net describing the same resource allocation

As mentioned above a CP-net consists of three different parts: the net structure, the declarations and the net inscriptions. The complexity of a description is distributed among these three parts and this can be done in many different ways. As an example, each arc to or from a resource place could have had a very simple arc expression of the form f(x), where the function f was defined in the declaration part. As another, and perhaps more sensible example, we could have represented all resources by means of a single place RES, as shown in Fig. 2. The + operator in the arc expressions denotes addition of multi-sets. As an example, 2\`s+1\`t is the multi-set that contains two appearances of s and one appearance of t:

1.2 Dynamic behaviour of non-hierarchical CP-nets

One of the most important properties of CP-nets (and other kinds of Petri nets) is that they – in contrast to many other graphical description languages – have a well-defined semantics which in an unambiguous way defines the behaviour of the system. The ideas behind the semantics are very simple, as we shall demonstrate by means of Fig. 3 - which contains one of the transitions from Fig. 1.[7]

The transition has two variables (x and i) and before we can consider an occurrence of the transition these variables have to be bound to colours of the corresponding types (i.e. elements of the colour sets U and I). This can be done in many different ways: One possibility is to bind x to p and i to zero. Then we get: b_1 = <x=p,i=0>. Another possibility is to bind x to q and i to 37. Then we get: b_2 = <x=q,i=37>.

For each **binding** we can check whether the transition, with this binding, is **enabled** (in the current marking). This is done by evaluating the guard and all the input arc expressions: In the present case the guard is trivial (a missing guard always evaluates to true). For the binding b_1 the two arc expressions evaluate to (p,0) and 2\`e, respectively. Thus we conclude that $b_1$ is enabled – because each of the input places contains at least the tokens to which the corresponding arc expression evaluates (one (p,0)-token on B and two e-tokens on S). For the binding $b_2$ the two arc expressions evaluate to (q,37) and 1\`e. Thus we conclude that b_2 is *not* enabled (there is no (q,37)-token on B). A transition can be executed in as many ways as the variables (in its arc expressions and guard) can be bound. However, for a given state, it is usually only a few of these bindings that are enabled.

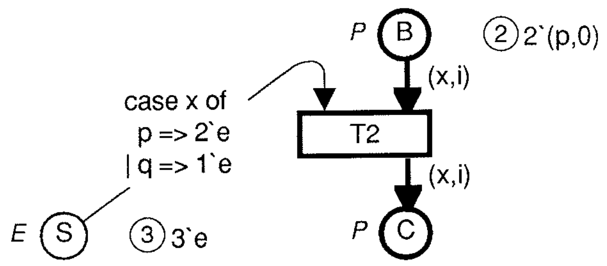


Figure 3. A transition from the resource allocation system

When a transition is enabled (for a certain binding) it may **occur** and it then removes tokens from its input places and adds tokens to its output places. The number of re-

[7] The inscriptions at the right hand side of the places indicate the current marking. The number of tokens are indicated in the small circle while the colours are described by the multi-set next to the circle.

moved/added tokens and the colours of these tokens are determined by the value of the corresponding arc expressions (evaluated with respect to the binding in question). An occurrence of the binding b_1 removes a (p,0)-token from B, removes two e-tokens from S and adds a (p,0)-token to C.[8] The binding b_2 is not enabled and thus it cannot occur.

A distribution of tokens (on the places) is called a **marking**. The **initial marking** is the marking determined by evaluating the initialization expressions. A pair, where the first element is a transition and the second element a binding of that transition, is called an **occurrence element**. Now we can ask whether an occurrence element O is enabled in a given marking M_1 – and when this is the case we can speak about the marking M_2 which is **reached** by the occurrence of O in M_1. It should be noticed that several occurrence elements may be enabled in the same marking. In that case there are two different possibilities: Either there are enough tokens (so that each occurrence element can get its own share) or there too few tokens (so that several occurrence elements have to compete for the same input tokens). In the first case the occurrence elements are said to be **concurrently enabled**. They can occur in the same **step** and they each remove their own input tokens and produce their own output tokens. In the second case the occurrence elements are said to be in **conflict** with each other and they cannot occur in the same step.

In the initial marking of Fig. 1 we observe that the occurrence element O_1 = (T1,<x=q,i=0>) is concurrently enabled with O_2 = (T2,<x=p,i=0>). This means that we can have a step where both O_1 and O_2 occur. Such a step is denoted by the multi-set 1\`$O_1$ + 1\`O_2 and when it occurs a (q,0)-token is moved from A to B and a (p,0)-token from B to C. Moreover, an e-token is removed from R and three e-tokens from S. It should be noticed that the effect of the step 1\`$O_1$ + 1\`O_2 is the same as when the two occurrence elements occur after each other in arbitrary order. This is an example of a general property: Whenever an enabled step contains more than one occurrence element, it can (in any thinkable way) be divided into two or more steps, which then are known to be able to occur after each other (in any thinkable order) and together have the same total effect as the original step.[9]

The above informal explanation of the occurrence rule, tells us how to understand the behaviour of a CP-net – and it explains the intuition on which CP-nets build. It is, however, very difficult (probably impossible) to make an informal explanation which is complete and unambiguous, and thus it is extremely important that the intuition is complemented by a more formal definition (which we shall present in chapter 2). It is the formal definition that has formed the basis for the implementation of a CP-net simulator, and it is also the formal definition that has made it possible to develop the analysis methods by which it can be *proved* whether a given CP-net has certain properties (e.g. absence of dead-locks).

---

8 We often think of the (p,0)-token as being moved from B to C. However, in the formal definition of CP-nets, the (p,0)-token added to C has no closer relationship to the (p,0)-token removed from B than it has to the two e-tokens removed from S.

9 Without this property it is very difficult to construct occurrence graphs because it no longer is sufficient to consider steps that correspond to single occurrence elements. It is, however, easy to violate this property and this is in fact done by many of the ad hoc extensions which are presented in the Petri net literature (e.g. the use of inhibitor arcs and some definitions of capacity).

Consider again the resource allocation system. It can easily be proved that this system has no dead-lock.[10] Now let us, in the initial marking, add an extra s resource (i.e. an extra e-token on S). Obviously, this cannot lead to a dead-lock (dead-locks appear when we have too few resource tokens and thus an extra resource token cannot cause a dead-lock). Is this argumentation convincing? At a first glance: yes! However, the argument is *wrong*, and the extra s resource actually means that we can reach a dead-lock (by letting the two p-processes advance to state D while the q-processes remain in state A). Hopefully, this small exercise demonstrates that informal arguments about behavioural properties are dangerous – and this is our motivation for the development of more formal analysis methods. We shall return to such methods in chapter 4.

2. Formal Definition of Non-Hierarchical CP-nets

The non-hierarchical CP-nets in the present paper are analogous to the CP-nets defined in [35] – but not identical to them (for more details see the abstract).

2.1 Multi-sets and expressions

In this section we define multi-sets and introduce the notation which we use to talk about expressions:

Definition 2.1: A **multi-set** m, over a non-empty and *finite* set S, is a function $m \in [S \to \mathbb{N}]$.[11] The non-negative integer $m(s) \in \mathbb{N}$ is the **number of appearances** of the element s in the multi-set m.

We usually represent the multi-set m by a formal sum:

$$\sum_{s \in S} m(s)\, s.$$

By S_{MS} we denote the **set of all multi-sets** over S. The non-negative integers $\{m(s) \mid s \in S\}$ are called the **coefficients** of the multi-set m, and m(s) is called the **coefficient** of s. An element $s \in S$ is said to **belong** to the multi-set m iff $m(s) \neq 0$ and we then write $s \in m$. The **empty multi-set** is the multi-set in which all coefficients are zero, and it is denoted by Ø (or by *empty)*.[12]

As an example, consider the set S = {a,b,c,d,e} and the two multi-sets $m_1 = a+2c+e$, and $m_2 = a+2b+3c+e$ which both are members of S_{MS}. As it can be seen, we usually omit S-values which have a zero coefficient and we also omit coefficients which are equal to one.[13] For multi-sets we define the following operations:

10 This can e.g. be done by means of occurrence graphs or by means of place invariants.

11 $\mathbb{N}$ denotes the set of all non-negative integers while $[A \to B]$ denotes the set of all functions from A to B.

12 To be precise there is an empty multi-set for each element set S. We shall, however, ignore this and allow ourselves to speak about *the* empty multi-set – in a similar way as we speak about *the* empty set and *the* empty list.

13 When the CPN tools described in chapter 5 was designed, it turned out to be convenient to insert an explicit operator between the coefficients and the S-values and include coefficients which are equal to

Definition 2.2: Summation, scalar-multiplication, comparison, and **multiplicity** of multi-sets are defined in the following way, for all m, $m_1, m_2 \in S_{MS}$ and $n \in \mathbb{N}$:

(i)	$m_1 + m_2 = \sum_{s \in S} (m_1(s) + m_2(s))\, s$		(summation).
(ii)	$n * m = \sum_{s \in S} (n \cdot m(s))\, s$		(scalar-multiplication).
(iii)	$m_1 \neq m_2 = \exists s \in S: [m_1(s) \neq m_2(s)]$		(comparison).
	$m_1 \leq m_2 = \forall s \in S: [m_1(s) \leq m_2(s)]$		(the relations $<$, $\geq$, $>$, and $=$ are defined analogously to $\leq$).
(iv)	$\lvert m \rvert = \sum_{s \in S} m(s)$		(multiplicity).

When $m_1 \leq m_2$ we also define **subtraction**:

(v)	$m_2 - m_1 = \sum_{s \in S} (m_2(s) - m_1(s))\, s$		(subtraction).

It can be shown that the multi-set operations above have a number of nice properties. As an example $(S_{MS}, +)$ is a commutative monoid.

For CP-nets, we use the terms *variables* and *expressions* in the same way as in typed lambda-calculus and functional programming languages. This means that expressions do *not* have side-effects and variables are *bound* to values (instead of being assigned to). It also means that complex expression are built, from variables and simpler subexpressions, by means of functions and operations. To give the abstract definition of CP-nets it is not necessary to fix the concrete syntax in which the modeller writes expressions, and thus we shall only assume that such a syntax exists (together with a well-defined semantics) – making it possible in an unambiguous way to talk about:

- The *type of a variable* v – denoted by Type(v).
- The *type of an expression* expr – denoted by Type(expr).
- The *set of variables in an expression* expr – denoted by Var(expr). The set of variables only includes the free variables – i.e. those which are *not* bound internally in the expression (e.g. by a local definition).
- A *binding of a set of variables* $V = \{v_1, v_2, \ldots, v_n\}$ – denoted by $<v_1=c_1, v_2=c_2, \ldots, v_n=c_n>$. It is demanded that $c_i \in \mathrm{Type}(v_i)$ for each variable $v_i \in V$.[14]
- The *value obtained by evaluating an expression* expr *in a binding* b – denoted by expr<b>. It is demanded that Var(expr) is a subset of the variables of b, and the evaluation is performed by substituting each variable $v_i \in \mathrm{Var(expr)}$ with the value $c_i \in \mathrm{Type}(v_i)$ determined by the binding.

one. In this case we write m_1=1\`a+2\`c+1\`e and $m_2$=1\`a+2\`b+3\`c+1\`e. This makes it easier to perform type checking (and it makes it easier to deal with multi-sets over integers, e.g. 3\`1+2\`35+1\`59).

14 For a type A we also use A to denote the *set of elements* in A, and we use $c \in A$ to denote that the value c is an element of A.

As an example, illustrating this notation, we may have:

$\text{Type}(x) = \text{Type}(y) = S.$
$\text{Var}(2 * (x + 3y)) = \{x,y\}.$
$\text{Type}(2 * (x + 3y)) = S_{MS}.$
$(2 * (x + 3y))\langle x=b,y=d\rangle = 2\text{`}b+6\text{`}d.$

2.2 Definition of non-hierarchical CP-nets

In this section we define non-hierarchical CP-nets as a many-tuple. It should, however be understood, that the only purpose of this is to give a mathematically sound and unambiguous definition of CP-nets and their semantics. Any concrete net – created by a modeller - will always be specified in terms of a CP-graph (i.e. a diagram similar to Fig. 1). In the following Bool is the boolean type (containing the elements Bool = {False,True} and having the standard logic operations). Some motivation and explanation of the individual parts of the definition is given immediately below the definition:

Definition 2.3: A **non-hierarchical** CP-net is a tuple CPN = (Σ, P, T, A, N, C, G, E, IN) satisfying the requirements below:

(i) Σ is a finite set of types, called **colour sets**. Each colour set must be finite and non-empty.

(ii) P is a finite set of **places**.

(iii) T is a finite set of **transitions**.

(iv) A is a finite set of **arcs** such that:
- $P \cap T = P \cap A = T \cap A = \emptyset$.

(v) N is a **node** function. It is defined from A into $P\times T \cup T\times P$.

(vi) C is a **colour** function. It is defined from P into Σ.

(vii) G is a **guard** function. It is defined from T into expressions such that:
- $\forall t\in T$: $[\text{Type}(G(t)) = \text{Bool} \wedge \text{Type}(\text{Var}(G(t))) \subseteq \Sigma]$.

(viii) E is an **arc expression** function. It is defined from A into expressions such that:
- $\forall a\in A$: $[\text{Type}(E(a)) = C(p(a))_{MS} \wedge \text{Type}(\text{Var}(E(a))) \subseteq \Sigma]$

where p(a) is the place of N(a).

(ix) IN is an **initialization** function. It is defined from P into expressions such that:
- $\forall p\in P$: $[\text{Type}(IN(p)) = C(p)_{MS} \wedge \text{Var}(IN(p)) = \emptyset]$.

(i) The set of **colour sets** determines the types, operations and functions that can be used in the net inscriptions (i.e. arc expressions, guards, initialization expressions, colour sets, etc.). If desired, the colour sets (and the corresponding operations and functions) can be defined by means of a many-sorted sigma algebra (in the same way as known from the theory of abstract data types). We demand all colour sets to be finite - although they may have a very large cardinality (e.g. be equivalent to all the real numbers which can be represented on a given computer). This restriction means that the linear extension of a function $F\in[A\rightarrow B_{MS}]$ to a function $\hat{F}\in[A_{MS}\rightarrow B_{MS}]$ always is known to be convergent. Such functions are used in the theory of place invariants and transition invariants.

(ii) + (iii) + (iv) The **places, transitions** and **arcs** are described by three sets P, T and A which are demanded to be finite and pairwise disjoint. In contrast to classical Petri nets, we allow the net structure to be empty (i.e. $P \cup T = \emptyset$). The reason is pragmatic: It allows the user to define and syntax check a set of colour sets without having to invent a dummy net structure.

(v) The **node** function maps each arc into a pair where the first element is the source node and the second the destination node. The two nodes have to be of different kind (i.e. one must be a place while the other is a transition). In contrast to classical Petri nets, we allow a CP-net to have several arcs between the same ordered pair of nodes (and thus we define A as a separate set and not as a subset of $P \times T \cup T \times P$). The reason is pragmatic: We often have nets where each occurrence element moves exactly one token along each of the surrounding arcs, and it is then awkward to be forced to violate this convention in the cases where an occurrence element removes/adds two or more tokens to/from the same place. It is of course easy to combine such multiple arcs to a single arc by adding the corresponding arc expressions (which must be of the same multi-set type). We also allow nodes to be isolated. Again the reason is pragmatic: When we build computer tools for CP-nets we want to be able to check whether a diagram is a CP-net (i.e. fulfils the definition above). There is, however, no conceptual difference between an isolated node and a node where all the arc expressions of the surrounding arcs always evaluate to the empty multi-set (and the latter is difficult to detect in general, since arc expressions may be arbitrarily complex). It is of course easy to exclude such degenerate nets when this is convenient for theoretical purposes.

(vi) The **colour** function C maps each place p into a set of possible **token colours** C(p). Each token on p must have a colour that belongs to the type C(p).

(vii) The **guard** function G maps each transition t into an expression of type boolean, i.e. a predicate. Moreover, all variables in G(t) must have types that belong to Σ.[15]

(viii) The **arc expression** function E maps each arc *a* into an expression which must be of type $C(p(a))_{MS}$. This means that each evaluation of the arc expression must yield a multi-set over the colour set that is attached to the corresponding place. We shall, as a shorthand, also allow an arc expression to be of type C(p(a)). In this case the arc expression evaluates to a colour in C(p(a)) which we then consider to be a multi-set with only one element.

(ix) The **initialization** function IN maps each place p into an expression which must be of type $C(p)_{MS}$ – i.e. a multi-set over C(p). The expression is not allowed to contain any variables. Analogously to (viii), we shall, as a shorthand, also allow an initial expression to be of type C(p).

As mentioned in the abstract, the "modern version" of CP-nets (presented in this paper) uses the expression representation (defined above) – not only when a system is being described, but also when it is being analysed. It is only during invariant analysis that it may be adequate/necessary to translate the expression representation into a function representation.

[15] For a set of variables Vars we use Type(Vars) to denote the set $\{\text{Type}(v) \mid v \in \text{Vars}\}$.

In addition to the concepts introduced in Def. 2.3, we use $X = P \cup T$ to denote the set of all **nodes**, and we define the following functions:[16]

- $s \in [A \to X]$ maps each arc *a* into the **source** of *a*, i.e. the first component of N(a).
- $d \in [A \to X]$ maps each arc *a* into the **destination** of *a*, i.e. the second component of N(a).
- $p \in [A \to P]$ maps each arc *a* into the **place** of N(a), i.e. that component of N(a) which is a place.
- $t \in [A \to T]$ maps each arc *a* into the **transition** of N(a), i.e. that component of N(a) which is a transition.
- $A \in [(P \times T \cup T \times P) \to A_S]$[17] maps each ordered pair of nodes (x_1,x_2) into the set of **connecting arcs**, i.e. the arcs that have the first node as source and the second as destination:
  $A(x_1,x_2) = \{a \in A \mid N(a) = (x_1,x_2)\}$.
- $A \in [X \to A_S]$[18] maps each node x into the set of **surrounding arcs**, i.e. the arcs that have x as source or destination:
  $A(x) = \{a \in A \mid \exists x' \in X: \ [N(a) = (x,x') \ \vee \ N(a) = (x',x)]\}$.
- $X \in [X \to X_S]$ maps each node x into the set of **surrounding nodes**, i.e. the nodes that are connected with x by an arc:
  $X(x) = \{x' \in X \mid \exists a \in A: \ [N(a) = (x,x') \ \vee \ N(a) = (x',x)]\}$.

All the functions above can, in the usual way, be extended to take sets as input (then they all return sets and thus all the function names are written with a capital letter).

2.3 Dynamic behaviour of non-hierarchical CP-nets

Having defined the static structure of CP-nets we are now ready to consider their behaviour – but first we introduce the following notation where Var(t) is called the set of **variables** of t while $E(x_1,x_2)$ is called the **expression** of (x_1,x_2):

- $\forall t \in T: \ [Var(t) = \{v \mid v \in Var(G(t)) \ \vee \ \exists a \in A(t): \ v \in Var(E(a))\}]$.
- $\forall (x_1,x_2) \in (P \times T \cup T \times P): \ [E(x_1,x_2) = \sum_{a \in A(x_1,x_2)} E(a)]$.[19]

Next we define what we mean by a binding. Intuitively, a binding, of a transition t, is a substitution that replaces each variable of t with a colour. It is demanded that each colour is of the correct type and that the guard evaluates to true:

16 Each function name indicates the range of the function – as an example p maps into places, while A maps into *sets* of arcs.

17 A_S denotes the set of all subsets of A.

18 From the argument(s) it will always be clear whether we deal with the function $A \in [X \to A_S]$, the function $A \in [(P \times T \cup T \times P) \to A_S]$ or the set A.

19 The summation indicates addition of expressions (and it is well-defined because all the participating expressions have a common multi-set type). From the arguments(s) it will always be clear whether we deal with the function $E \in [A \to Exp]$ or the function $E \in [(P \times T \cup T \times P) \to Exp]$.

Definition 2.4: For a transition $t \in T$ with variables $Var(t) = \{v_1,v_2,\ldots,v_n\}$ we define the **binding type** BT(t) as follows:

$$BT(t) = Type(v_1) \times Type(v_2) \times \ldots \times Type(v_n).$$[20]

Moreover, we define the set of all **bindings** B(t) as follows:

$$B(t) = \{(c_1,c_2,\ldots,c_n) \in BT(t) \mid G(t)\langle v_1=c_1,v_2=c_2,\ldots,v_n=c_n\rangle\}.$$[21]

For convenience we denote bindings in two different ways: Either in the form $\langle v_1=c_1,v_2=c_2,\ldots,v_n=c_n\rangle$ or in the form $(c_1,c_2,\ldots,c_n)$. In both cases this denotes an element of BT(t). Next we define token distributions, binding distributions, markings and steps:[22]

Definition 2.5: A **token distribution** is a function M, defined on P such that $M(p) \in C(p)_{MS}$ for all $p \in P$. The set of all token distributions (for a given CP-net CPN) is denoted by TD_{CPN}, and for all $M_1,M_2 \in TD_{CPN}$ we define the relations $\neq$ and $\leq$ in the following way:

(i) $M_1 \neq M_2 \quad \Leftrightarrow \quad \exists p \in P: [M_1(p) \neq M_2(p)]$.

(ii) $M_1 \leq M_2 \quad \Leftrightarrow \quad \forall p \in P: [M_1(p) \leq M_2(p)]$.

The relations $<$, $\geq$, $>$, and $=$ are defined analogously to $\leq$. When $c \in M(p)$ for some $c \in C(p)$, we say that the pair (p,c) is an **element** of M, and we write $(p,c) \in M$. Moreover, we say that M is **non-empty** iff it has at least one element.

A **binding distribution** is a function Y, defined on T such that $Y(t) \in B(t)_{MS}$ for all $t \in T$.[23] The set of all binding distributions (for a given CP-net CPN) is denoted by BD_{CPN}, and the relations $\neq$, $\leq$, $<$, $\geq$, $>$, and $=$ are defined analogously to the way they were defined for token distributions. When $b \in Y(t)$ for some $b \in B(t)$, we say that the pair (t,b) is an **element** of Y, and we write $(t,b) \in Y$. Moreover, we say that Y is **non-empty** iff it has at least one element.

A **marking** of a CP-net is a token distribution and a **step** is a *non-empty* binding distribution. The set of all markings (for a given CP-net CPN) is denoted by M_{CPN}, and the set of all steps is denoted by Y_{CPN}. The **initial marking** M_0 is the marking which is obtained by evaluating the initialization expressions, i.e. the marking where $M_0(p) = IN(p)\langle\rangle$ for all $p \in P$.[24]

20 We assume that the set of variables Var(t) is ordered – in some arbitrary way.

21 As defined in section 2.1, $G(t)\langle v_1=c_1,v_2=c_2,\ldots,v_n=c_n\rangle$ denotes the evaluation of the guard expression G(t) in the binding $\langle v_1=c_1,v_2=c_2,\ldots,v_n=c_n\rangle$.

22 There is no difference between the set of token distributions and the set of markings, and there is very little difference between the set of binding distributions and the set of steps. In this paper we only use token/binding distributions to define markings/steps and thus it may seem unnecessary to introduce all four sets. Token/binding distributions are however, general concepts, which are useful in a number of other contexts (in which it would be misleading to talk about markings/steps).

23 It should be noticed that all bindings of a binding distribution, according to Definition 2.4, automatically satisfy the corresponding guard.

24 $IN(p)\langle\rangle$ denotes the evaluation of IN in the empty binding $\langle\rangle$ (which is used because IN(p) has an empty set of variables).

Definition 2.6: A step Y is **enabled** in a marking M iff the following property is satisfied:

$$\forall p \in P: [\sum_{(t,b) \in Y} E(p,t)\langle b\rangle \leq M(p)].$$

Let Y be an enabled step, with respect to a given marking M. When $(t,b) \in Y$, we say that t is **enabled** in M for the **binding** b. We also say that the pair (t,b) is enabled in M, or simply that t is enabled in M. When two different transitions $t_1, t_2 \in T$ satisfy $Y(t_1) \neq \emptyset \neq Y(t_2)$, we say that t_1 and t_2 are **concurrently enabled**. When a transition $t \in T$ satisfies $|Y(t)| \geq 2$, we say that t is **concurrently enabled with itself** and when it for a binding $b \in B(t)$ satisfy $Y(t) \geq 2`b$, we say that (t,b) is **concurrently enabled with itself.**

When a step is enabled it may occur and this means that tokens are removed from the input places and added to the output places of the occurring transitions. The number and colours of the tokens are determined by the arc expressions, evaluated for the occurring bindings:

Definition 2.7: When a step Y is enabled in a marking M_1 it may **occur**, changing the marking M_1 to another marking M_2, defined by:

$$\forall p \in P: [M_2(p) = (M_1(p) - \sum_{(t,b) \in Y} E(p,t)\langle b\rangle) + \sum_{(t,b) \in Y} E(t,p)\langle b\rangle].$$

The first sum is called the **removed** tokens while the second is called the **added** tokens. Moreover we say that M_2 is **directly reachable** from M_1 by the occurrence of the step Y, which we also denote:

$$M_1 [Y\rangle M_2.$$

Definition 2.8: A **finite occurrence sequence** is a sequence of markings and steps:

$$M_1 [Y_1\rangle M_2 [Y_2\rangle M_3 \ldots\ldots M_n [Y_n\rangle M_{n+1}$$

such that $n \in \mathbb{N}$, and $M_i [Y_i\rangle M_{i+1}$ for all $i \in 1..n$.[25] The marking M_1 is called the **start marking** of the occurrence sequence, while the marking M_{n+1} is called the **end marking**. The non-negative integer n is called the **number of steps** in the occurrence sequence, or the **length** of it.

Analogously, an **infinite occurrence sequence** is a sequence of markings and steps:

$$M_1 [Y_1\rangle M_2 [Y_2\rangle M_3 \ldots\ldots$$

such that $M_i [Y_i\rangle M_{i+1}$ for all $i \in \mathbb{N}_+$.[26] The marking M_1 is called the **start marking** of the occurrence sequence, which is said to have **infinite length.**

25 By 1..n we denote the set of all integers i that satisfy $1 \leq i \leq n$.

26 $\mathbb{N}_+$ denotes the set of all positive integers.

The start marking of an occurrence sequence will often, but not always, be identical to the initial marking of the CP-net. We allow the user to omit some parts of an occurrence sequence and e.g. write:

$$M_1 [Y_1 Y_2 \ldots Y_n \rangle M_{n+1}.$$

Definition 2.9: A marking M" is **reachable from** a marking M' iff there exists a finite occurrence sequence having M' as start marking and M" as end marking – i.e. iff there, for some $n \in \mathbb{N}$, exists a finite sequence of steps such that:

$$M' [Y_1 Y_2 \ldots Y_n \rangle M''.$$

We then also say that M" is reachable from M' in **n steps**. The *set* of markings which are reachable from M' is denoted by $[M'\rangle$. As a shorthand, we say that a marking is **reachable** iff it is reachable from the initial marking M_0 – i.e. contained in $[M_0\rangle$.

It should be obvious that behavioural properties, such as dead-lock, liveness, home markings, boundedness and fairness, can be defined for CP-nets in a similar way as for Place/Transition Nets (PT-nets). It is well-known that each CP-net has an equivalent PT-net, and each behavioural property is defined in such a way that a given CP-net has the property iff the equivalent PT-net has. The definitions of the behavioural properties are outside the scope of this paper.

2.4 Some historical remarks about the development of CP-nets

The foundation of Petri nets was presented by Carl Adam Petri in his doctoral-thesis [48]. The first nets were called Condition/Event Nets (CE-nets). This net model allows each place to contain at most one token – because the place is considered to represent a boolean condition, which can be either true or false. In the following years a large number of persons contributed to the development of new net models, basic concepts, and analysis methods. One of the most notable results was the development of Place/Transition Nets (PT-nets). This net model allows a place to contain several tokens. The first coherent presentation of the theory and application of Petri nets was given in the course material developed for the First Advanced Course on Petri Nets [5] and later this was supplemented by the course material for the Second Advanced Course on Petri Nets [6] and [7].

For theoretical considerations CE-nets turned out to be more tractable than PT-nets and much of the theoretical work concerning the definition of basic concepts and analysis methods has been performed on CE-nets. Later, a new net model called Elementary Nets (EN-nets) has been proposed in [51] and [57]. The basic ideas of this net model are very close to CE-nets – but EN-nets avoid some of the technical problems which have turned out to be present in the original definition of CE-nets.

For practical applications, PT-nets were used. However, it often turned out that this net model was too low-level to cope with the real-world applications in a manageable way, and different researchers started to develop their own extensions of PT-nets - adding concepts such as: priority between transitions, time delays, global variables to be tested and updated by transitions, zero testing of places, etc. In this way a large number of different net models were defined. However, most of these net models were designed with a single – and often very narrow – application area in mind. This created a serious problem: Although some of the net models could be used to give adequate descriptions of certain systems, most of the net models possessed nearly no analytic power. The main reason for this was the large variety of different net models. It often turned out to be a difficult task to translate an analysis method developed for one net model to another – and in this way the efforts to develop suitable analysis methods were widely scattered.

The breakthrough with respect to this problem came when Predicate/Transition Nets (PrT-nets) were presented in [20]. PrT-nets were the first kind of high-level nets which was constructed without any particular application area in mind. PrT-nets form a nice generalization of PT-nets and CE-nets (exploiting the same kind of reasoning that leads from propositional logic to predicate logic). PrT-nets can be related to PT-nets and CE-nets in a formal way – and this makes it possible to generalize most of the basic concepts and analysis methods that have been developed for these net models – so that they also become applicable to PrT-nets. Later, an improved definition of PrT-nets has been presented in [22]. This definition draws heavily on sigma algebras (as known from the theory of abstract data types).

However, it soon turned out that PrT-nets present some technical problems when the analysis methods of place invariants and transition invariants are generalized. It is possible to calculate invariants for PrT-nets, but the interpretation of the invariants is difficult and must be done with great care to avoid erroneous results. The problem arises because of the variables which appear in the arc expressions of PrT-nets. These variables also appear in the invariants, and to interpret the invariants it is necessary to bind the variables, via a complex set of substitution rules. To overcome this problem the first version of Coloured Petri Nets (CP^{81}-nets) was defined in [32]. The main ideas of this net model are directly inspired by PrT-nets, but the relation between an occurrence element and the token colours involved in the occurrence is now defined by functions and not by expressions as in PrT-nets. This removes the variables, and invariants can now be interpreted without problems.

However, it often turns out that the functions attached to arcs in CP^{81}-nets are more difficult to read and understand than the expressions attached to arcs in PrT-nets. Moreover, as indicated above, there is a strong relation between PrT-nets and CP^{81}-nets and from the very beginning it was clear that most descriptions in one of the net models could be informally translated to the other net model and vice versa. This lead to the idea of an improved net model – combining the qualities of PrT-nets and CP^{81}-nets. This net model was defined in [33] where it was called High-level Petri Nets (HL-nets). Unfortunately, this name has given rise to a lot of confusion since the term "high-level nets" at that time started to become used as a generic name for PrT-nets, CP^{81}-nets, HL-nets, and several other kinds of net models. To avoid this confusion it was necessary to rename HL-nets to Coloured Petri Nets (CP^{87}-nets). CP^{87}-nets have two different representations (and formal translations between them). The expression representation is nearly identical to PrT-nets (as presented in [20]), while the function representation is nearly identical to CP^{81}-nets. The first coherent presentation of CP^{87}-nets and their analysis methods was given in [35].

Today most of the practical applications of Petri nets (reported in the literature) use either PrT-nets or CP-nets – although several other kinds of high-level nets have been proposed. There is very little difference between PrT-nets and CP-nets (and many modellers do not make a clear distinction between the two kinds of net models). The main differences between the two net models are today hidden inside the methods to calculate and interpret place and transition invariants (and this is of course not surprising when you think about the original motivation behind the development of CP^{81}-nets). Instead of viewing PrT-nets and CP-nets as two different modelling languages it is, in our opinion, much more adequate to view them as two slightly different dialects of the same language.

3. Hierarchical CP-nets

Hierarchical CP-nets were first presented in [31] and it should be understood that this (as far as we know) was the very first successful attempt to create a set of hierarchy concepts for a class of high-level Petri nets. This means that the proposed concepts are likely to undergo many improvements and refinements (in the same way as the first very simple concept of subroutines has undergone dramatical changes to become the procedure concept of modern programming languages). In other words: We do not claim that our current proposal will be the "final solution". However, we do think that it constitutes a good starting point for further research and practical experiences in this area. In chapter 6 we describe a number of industrial applications of hierarchical CP-nets and more information about some of these can be found in [49], [54] and [55].

In [31] individual CP-nets, called pages, are related in five different ways, known as the five **hierarchy constructs**: substitution of transitions, substitution of places, invocation of transitions, fusion of places and fusion of transitions. In the present paper we shall, however, only deal with the first and fourth of these hierarchy constructs.[27] For an explanation of the other three hierarchy constructs the reader is referred to [31].

The intention has been to make a set of hierarchy constructs, which is general enough to be used with many different development methods and with many different analysis techniques. This means that we present the hierarchy constructs *without* prescribing specific methods for their use. Such methods have to be developed and written down – but this can only be done as we get more experiences with the practical use of the hierarchy constructs. Eventually the new development methods and analysis techniques will influence the definition of the hierarchy constructs – in the same way as modern programming languages have been influenced by the progress in the areas of programming methodology and verification techniques.[28]

3.1 Substitution of transitions

The intuitive idea behind substitution transitions is to allow the user to replace a transition (and its surrounding arcs) by a more complex CP-net – which usually gives a more precise and detailed description of the activity represented by the substituted transition.

[27] These are the two hierarchy constructs that are supported by the current version of the CPN tools described in chapter 5 – and they are the easiest to define, understand and use.

[28] During the design of the hierarchy constructs we have, of course, been influenced by the constructs and methods used with other graphical description languages and with modern programming languages.

The idea is analogous to the hierarchy constructs found in many graphical description languages (e.g. IDEF/SADT diagrams [43] and Yourdon diagrams [64]) – and it is also, in some respects, analogous to the module concepts found in many modern programming languages: At one level we want to give a simple description of the activity (without having to consider internal details about how it is carried out). At another level we want to specify the more detailed behaviour. Moreover, we want to be able to integrate the detailed specification with the more crude description and this integration must be done in such a way that it becomes meaningful to speak about the behaviour of the *combined* system.

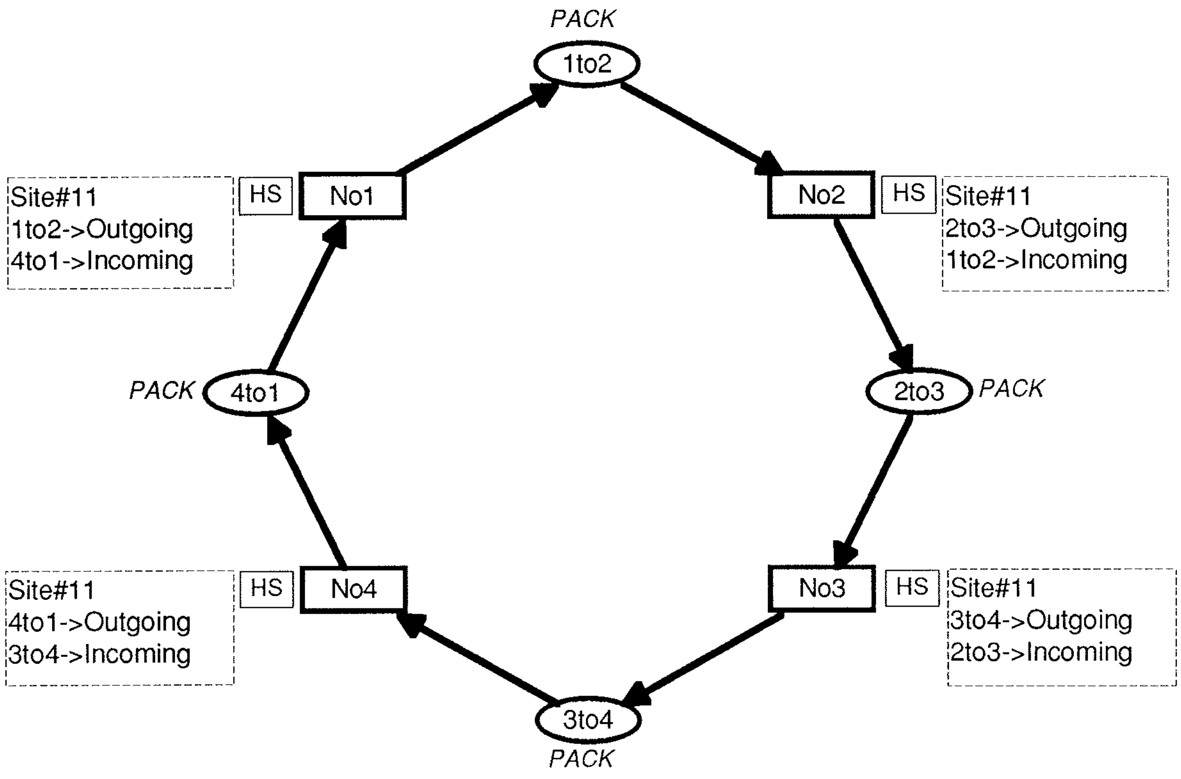
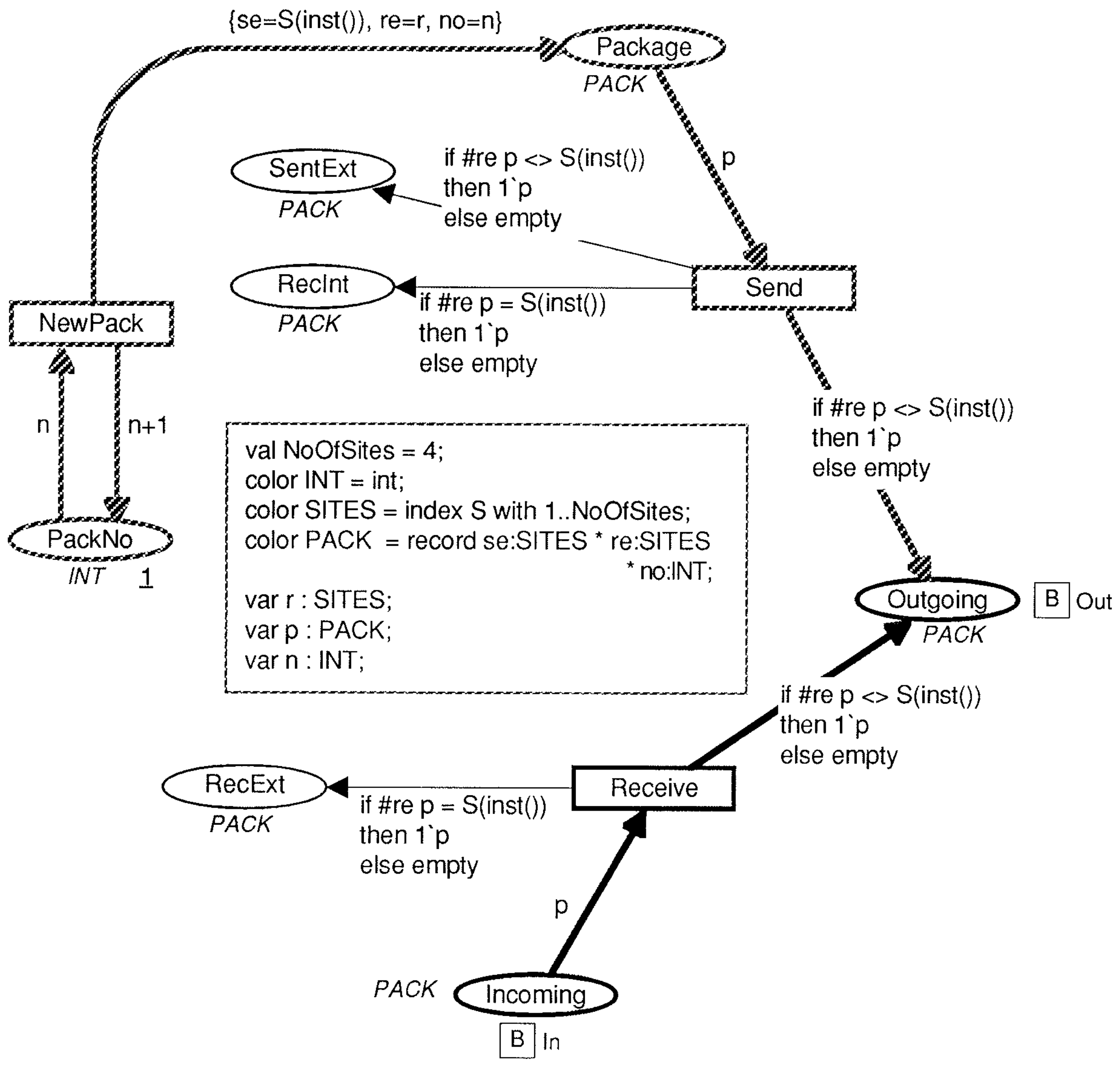


Figure 4. NetWork#10 describes a ring network with four different sites

As mentioned above, we want to relate individual CP-nets to nodes, which are members of other CP-nets, and this means that our description will contain a *set* of (non-hierarchical) CP-nets – which we shall call **pages**. Now let us consider a small example.[29] Imagine that we have a ring network with four different sites. This can be described by the page NetWork#10 in Fig. 4.[30] The four sites are represented by the four substitution transitions – NO1, NO2, NO3 and NO4 – each of which has an HS-tag adjacent (HS ≈ Hierarchy + Substitution). The inscription next to the HS-tag is called a **hierarchy inscription** and it defines the details of the actual substitution. We shall return to the hierarchy inscriptions in a moment, but let us first consider Site#11 in Fig. 5. This page describes an individual site.

[29] The purpose of the example is to explain the semantics of substitution transitions. The described ring network is far too simple to be realistic.

[30] To be able to refer to the individual pages we give each of them a page name (e.g. NetWork) and a page number (e.g. 10).



```
val NoOfSites = 4;
color INT = int;
color SITES = index S with 1..NoOfSites;
color PACK  = record se:SITES * re:SITES
                                  * no:INT;
var r : SITES;
var p : PACK;
var n : INT;
```

Figure 5. Site#11 describes an individual site of the ring network

Some of the declarations in the middle of Fig. 5 need a little explanation: The first line declares a constant NoOfSites. It is used in one of the other declarations – and it could also have been used e.g. in the arc expressions of NetWork#10 and Site#11, if desired. The colour set SITES contains four different elements which are denoted by S(1), S(2), S(3) and S(4).[31] The colour set PACK contains all records which have an se-field (identifying the sender), an re-field (identifying the receiver) and a no-field (containing a package number).

Site#11 has three different transitions: Each occurrence of NEWPACK creates a new package. The se-field of the new package becomes identical to S(inst()) where the pre-declared function inst() returns the identity number of the page instance on which the transition occurs[32] while the no-field is read from PACKNO. The re-field is determined

31 The idea behind index colour sets is to make it easy for the user to define colour sets which are of the form $\{S_1, S_2, \ldots S_n\}$.

32 Allowing net inscriptions (such as arc expressions, guards and initialization expressions) to be dependent on the page instance is a generalization – with respect to the class of hierarchical CP-nets formally defined in section 3.4. The extension, which has turned out to be extremely useful, will be

by the variable r – which does not appear anywhere else, and this means that r can take an arbitrary value (from SITES). The created packages are handled by SEND, which inspects the re-field of the package.[33] When the re-field indicates that the receiver is different from the present site, the package is transferred to the network via OUTGOING (and a copy is put on SENTEXT). Otherwise the package is sent directly to RECINT. Finally, RECEIVE inspects all packages which are transferred from the network via INCOMING. Again the re-field is inspected, and based on this inspection the package is routed, either to OUTGOING or to RECEXT.

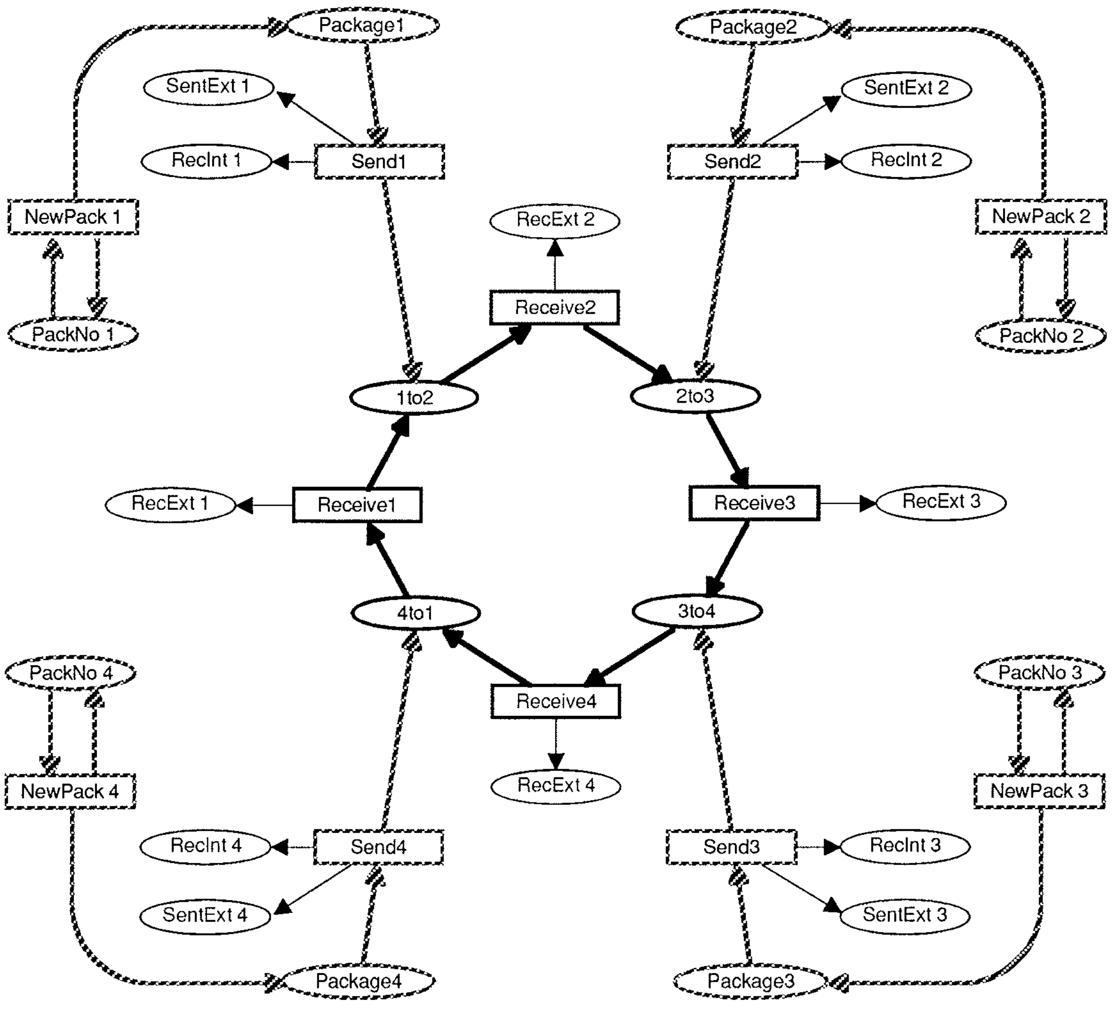


Figure 6. Non-hierarchical CP-net with the same behaviour as the hierarchical CP-net that contains the pages NetWork#10 and Site#11

Now let us return to the hierarchy inscriptions of the four substitution transitions on NetWork#10: The first line of each hierarchy inscription identifies the **subpage**, i.e. the page that is going to replace the substitution transition. In our present example, the four substitution transitions are being replaced by the same subpage Site#11. Each substitution transition gets, however, its own "private copy" of Site#11.

supported by one of the next versions of the CPN tool described in chapter 5. The page instance numbers are consecutive positive numbers, starting from 1 (i.e. in this case: 1, 2, 3, and 4).

33 We use *#re p* to denote the re-field of a package p.

The remaining lines of the hierarchy inscription contain the **port assignment** which tells how the subpage (Site#11) is going to be inserted into the **superpage** which contains the substitution transition (NetWork#10). Each line of the port assignment relates a **socket node** on the superpage (i.e. one of the nodes surrounding the substitution transition) to a **port node** on the subpage (i.e. one of the nodes which have a B-tag next to it (B ≈ Border)). In our example, let us now consider the hierarchy inscription next to NO2. The first line of the port assignment tells us that the socket node 2TO3 is assigned to (i.e. "glued" together with) the port node OUTGOING. Analogously, the second line tells us that 1TO2 is assigned to INCOMING. The remaining three hierarchy inscriptions (of NO1, NO3 and NO4) are interpreted in a similar way – and this tells us that the hierarchical CP-net with the two pages NetWork#10 and Site#11 is equivalent – i.e. has the same behaviour – as the non-hierarchical CP-net in Fig. 6 (where we for clarity have omitted the net inscriptions).

When we consider the behaviour of a hierarchical CP-net each page has its own marking. We allow a single page to replace several substitution transitions, and then the page has several **page instances**, each having its own marking. In the example above, their are four instances of Site#11 – and thus four different markings.[34]

3.2 Fusion of places

The intuitive idea behind fusion of places is to allow the user to specify that a set of places are considered to be identical – i.e. they all represent a single place even though they are drawn as individual places. This means that when a token is added/removed at one of the places, an identical token has to be added/removed at all the others. The places of a fusion set may belong to a single page or to several different pages.

When all members of a fusion set belong to a single page and that page only has one instance, place fusion is nothing other than a drawing convenience that allows the user to avoid too many crossing arcs. However, things become much more interesting when the members of a fusion set belong to several different pages *or* to a page that has many different page instances. In that case fusion sets allow the user to specify a behaviour which it might be cumbersome to describe without fusion. To allow modular analysis of hierarchical CP-nets, global fusion sets should be used with care.

There are three different kinds of fusion sets: Global fusion sets are allowed to have members from many different pages, while page and instance fusion sets only have members from a single page. The difference between the last two is the following: A page fusion unifies all the instances of its places (independently of the page instance at which they appear), and this means that the fusion set only has one "resulting place" which is "shared" by all instances of the corresponding page. In contrast to this, an instance fusion set only identifies place instances that belong to the *same* page instance, and this means that the fusion set has a "resulting place" for each page instance. A global fusion set is analogous to a page fusion set, in the sense that it only has one "resulting place" (which is common for all instances of all the participating pages).

[34] When a CP-net is simulated by means of the CPN tools described in chapter 5, we have a window for each page. The window shows the marking of one instance at a time, and it is possible for the user to switch from one instance to another.

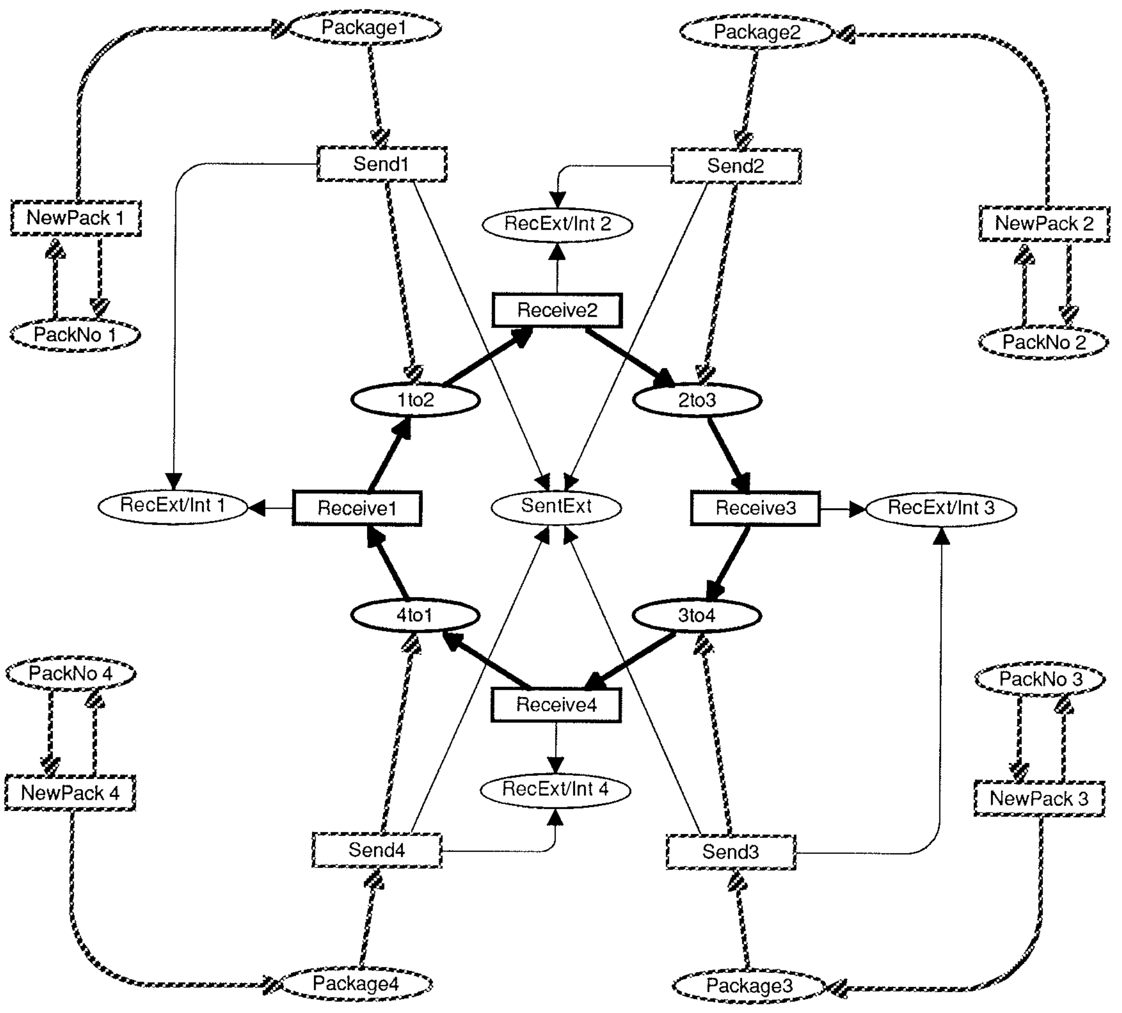


Figure 7. Non-hierarchical equivalent of a CP-net with two fusion sets

The difference between page and instance fusion sets can be illustrated by the ring network. In Fig. 7 we show how the non-hierarchical CP-net in Fig. 6 is modified when we on Site#11 define two fusion sets: An instance fusion set containing {RECEXT, RECINT} and a page fusion set containing {SENTEXT}.

Above we have illustrated the difference between page and instance fusion sets by drawing a non-hierarchical CP-net which is behaviourally equivalent to our hierarchical CP-net. It should, however, be understood that hierarchical CP-nets is a modelling language in its own right. This means that it is possible (and desirable) to model and analyse a complex system by a hierarchical CP-net – without ever constructing the equivalent non-hierarchical CP-net.

3.3 Partitions

To give a formal definition of hierarchical CP-nets we need the concept of a partition. Intuitively, a partition is a division of a set into a number of subsets, which are non-empty and pairwise disjoint:

Definition 3.1: Let a finite set Z be given. A **partition** of Z is a family of sets $X = \{X_i\}_{i \in I}$ such that:

(i) The **index set** I is a finite set.

(ii) Each **component** X_i is a non-empty subset of Z:
- $\forall i \in I$: $[\emptyset \subset X_i \subseteq Z]$.

(iii) The components are pairwise disjoint:
- $\forall (i,k) \in I$: $[i \neq k \Rightarrow X_i \cap X_k = \emptyset]$.

The **range** of the partition is the set:

$$X_R = \bigcup_{i \in I} X_i \subseteq Z.$$

The partition is said to be **total** iff $X_R = Z$. Otherwise it is **partial**.

It should be obvious that there is a very close relationship between partitions and equivalence relations: On the one hand, each partition determines an equivalence relation for its range (two elements are equivalent iff they belong to the same component - and each component is an equivalence class). On the other hand, each equivalence relation determines a total partition (two elements belong to the same component if they are equivalent – and each equivalence class is a component).

3.4 Formal definition of hierarchical CP-nets

This section contains the formal definition of hierarchical CP-nets. Some motivation and explanation of the individual parts of the definition is given immediately below the definition:

Definition 3.2: A **hierarchical CP-net** is a tuple HCPN = (S, SN, SA, PN, PA, FS, FT, PP) satisfying the requirements below:

(i) $S = \{S_i \mid i \in I\}$ is a finite set of **pages** such that:
- Each page is a non-hierarchical CP-net:
  $S_i = (\Sigma_i, P_i, T_i, A_i, N_i, C_i, G_i, E_i, IN_i)$.
- The sets of net elements are pairwise disjoint:
  $\forall (i,k) \in I$: $[i \neq k \Rightarrow (P_i \cup T_i \cup A_i) \cap (P_k \cup T_k \cup A_k) = \emptyset]$.

When XX_i is a set, relation or a function, defined for all $i \in I$, we use XX to denote the union.[35] When YY is a set, relation or function, defined for HCPN, we use YY_i (and YY_{S_i}) to define the restriction to S_i.

(ii) $SN \subseteq T$ is a set of **substitution nodes**.

(iii) SA is a **page assignment** function. It is defined from SN into S such that:
- No page is a subpage of itself:[36]
  $\{i_0\, i_1 \ldots i_n \in I^* \mid n \in \mathbb{N}_+ \wedge i_0 = i_n \wedge \forall k \in 1..n: S_{i_k} \in SA(SN_{i_{k-1}})\} = \emptyset$. *(cont.)*

35 The union of a set of functions is the union of the corresponding set of relations and this is known to become a function (because the set of domains are pair-wise disjoint).

36 I* denotes all finite sequences with elements from I, and we extend SA so that it can be used on a *set* of substitution nodes..

(iv) $PN \subseteq P$ is a set of **port nodes**.

(v) PA is a **port assignment** function. It is defined from SN into binary relations such that:
- Socket nodes are related to port nodes:
  $PA(x) \subseteq X(x) \times PN_{SA(x)}$.
- Related nodes have identical colour sets and equivalent initialization expressions:
  $\forall x \in SN\ \forall (x_1,x_2) \in PA(x)$: $[C(x_1) = C(x_2)\ \wedge\ IN(x_1)\langle\rangle = IN(x_2)\langle\rangle]$.

(vi) $FS = \{FS_r\}_{r \in R}$ is a finite set of **fusion sets** such that:
- FS is a partition of P.
- Members of a fusion set have identical colour sets and equivalent initialization expressions:

$\forall r \in R\ \forall x_1,x_2 \in FS_r$: $[C(x_1) = C(x_2)\ \wedge\ IN(x_1)\langle\rangle = IN(x_2)\langle\rangle]$.

(vii) FT is a **fusion type** function. It is defined from fusion sets such that:
- Each fusion set is of type: global, page or instance.
- Page and instance fusion sets belong to a single page:
  $\forall r \in R$: $[FT(FS_r) \neq \text{global} \Rightarrow \exists i \in I: FS_r \subseteq P_i]$.

(viii) $PP \in S_{MS}$ is a multi-set of **prime pages**.

(i) Each **page** is a non-hierarchical CP-net. We use Σ to denote the union of all the colour sets Σ_i of the individual pages (these colour sets do *not* need to be disjoint). The pages have pairwise disjoint sets of nodes and arcs, and this means that for functions and relations, defined on places, transitions and arcs, we can omit the page index without any ambiguity. As an example we can write C(p), G(t) and E(a) instead of $C_i(p)$, $G_i(t)$ and $E_i(a)$. Analogously, we use P, T and A to denote the set of all places, the set of all transitions and the set of all arcs in HCPN. The notational conventions described above allows us to move our point of focus from a given page to the entire CP-net by omitting the page index. It is, however, also possible to do the opposite and this means that we restrict a set, relation or function, defined for elements of the entire CP-net, to elements of a particular page. As an example, we use SN_i (and SN_{S_i}) to denote the subset of substitution nodes that belong to page S_i.

(ii) Each **substitution node** is a transition.[37]

(iii) The **page assignment** relates substitution transitions to pages. When a transition $t \in SN_i$ is related to a page S_k, we say that S_k is a *direct subpage* of the page S_i which is a *direct superpage* of S_k. Analogously, we say that S_k is a *direct subpage* of the node x which is a *direct supernode* of S_k. These four relations are in the usual way extended by taking the transitive closure and we then omit the word "direct" and talk about subpages, superpages and supernodes. It is demanded that no page is a subpage of itself. Otherwise, the process of substituting supernodes with their direct subpages will be infinite and it would be impossible to construct an equivalent non-hierarchical CP-net (without allowing P, T and A to be infinite).

(iv) Each **port node** is a place. It should be noticed that we allow a page to have port nodes even when it is not a subpage. Such port nodes have no semantic meaning

[37] As described in [31] it is also possible to allow places to be substitution nodes. The semantics of such a model is, however, slightly more complex.

(and thus they can be turned into non-ports without changing the behaviour of the CP-net).

(v) The **port assignment** relates *socket nodes* (i.e. the places surrounding a substitution transition) with port nodes (on the corresponding direct subpage). Each related pair of socket/port nodes must have identical colour sets and equivalent initialization expressions. It should be noticed that it is possible to relate several sockets to the same port and vice versa. It is also possible to have sockets and ports which are totally unrelated. Usually, most port assignments are bijective functions and in that case there is a one to one correspondence between sockets and ports.

(vi) The **fusion sets** are the components in a partition of P and this means that a place can belong to at most one fusion set. All members of a fusion set must have identical colour sets and equivalent initialization expressions. Usually, it is only a few places that belong to fusion sets and thus the partition is partial.

(vii) The **fusion type** divides the set of fusion sets into global, page and instance fusion sets. For the last two kinds of fusion sets all members must belong to the same page.

(viii) The **prime pages** is a multi-set over the set of all pages and they determine, together with the page assignment, how many instances the individual pages have. Often the multi-set contains only a single page (with coefficient one).

It should be obvious that each non-hierarchical CP-net is a hierarchical CP-net with a single page. There are no substitution, port and fusion nodes – and thus the page assignment, port assignment and fusion type functions become trivial. The single page belongs to the multi-set of prime pages, with coefficient one.

3.5 Page, place, transition and arc instances

A page may have several instances: There is a page instance for each time the page appears in the multi-set PP and, moreover there is a page instance for each way in which the page is a subpage of an element of PP.

In the following definition s and n identify the element of PP from which the page instance is constructed, while $x_1 x_2 \ldots x_m$ identifies the sequence of substitution nodes that leads to the page instance (in this sequence each node x_{k+1} belongs to the direct subpage of x_k). It should be noticed that the sequence may be empty:

Definition 3.3: The **page instances** of a page $S_i \in S$ is the set SI_i of all triples $(s, n, x_1 x_2 \ldots x_m)$ that satisfy the following requirements:

(i) $s \in PP \;\wedge\; n \in 1..PP(s)$.

(ii) $x_1 x_2 \ldots x_m$ is a sequence of substitution nodes, with $m \in \mathbb{N}$, such that:
$m > 0 \;\Rightarrow\; (x_1 \in SN_s \;\wedge\; [k \in 1..(m-1) \;\Rightarrow\; x_{k+1} \in SN_{SA(x_k)}] \;\wedge\; SA(x_m) = S_i)$.

When a page has several page instances each of these have their own instances of the corresponding places, transitions and arcs. It should, however, be noticed that substitution nodes and their surrounding arcs do not have instances (because they are replaced by instances of the corresponding direct subpages):

Definition 3.4: The **place instances** of a page $S_i \in S$ is the set PI_i of all pairs (p,id) that satisfy the following requirements:
(i) $p \in P_i$.
(ii) $id \in SI_i$.

The **transition instances** of a page $S_i \in S$ is the set TI_i of all pairs (t,id) that satisfy the following requirements:
(iii) $t \in T_i - SN_i$.
(iv) $id \in SI_i$.

The **arc instances** of a page $S_i \in S$ is the set AI_i of all pairs (a,id) that satisfy the following requirements:
(v) $a \in A_i - A(SN_i)$.
(vi) $id \in SI_i$.

Place instances may be related to each other, either by means of fusion sets or by means of port assignments and this leads to the following concepts:

Definition 3.5: The **place instance relation** is the smallest equivalence relation on PI[38] containing all those pairs $((p_1,(s_1,n_1,xx_1)),(p_2,(s_2,n_2,xx_2))) \in PI$ that satisfy one of the following conditions:
(i) $\exists r \in R$: $[p_1,p_2 \in FS_r \wedge (FT(FS_r) = \text{instance} \Rightarrow (s_1,n_1,xx_1) = (s_2,n_2,xx_2))]$.
(ii) $\exists t \in SN$: $[(p_1,p_2) \in PA(t) \wedge (s_1,n_1) = (s_2,n_2) \wedge xx_1\text{^}t = xx_2]$.[39]

An equivalence class of the place instance relation is called a **place instance group** and the set of all such equivalence classes is denoted by PIG.

3.6 Equivalent non-hierarchical CP-net

In sections 3.1 and 3.2 we have sketched how to define the behaviour of a hierarchical CP-net – by constructing a non-hierarchical CP-net that is behaviourally equivalent. In this section we define the non-hierarchical equivalent in a much more formal way, but before doing this we again want to stress that the construction of the non-hierarchical equivalent plays a similar role as the unfolding of a CP-net to a behaviourally equivalent PT-net: The construction is only performed in order to define and understand the semantics. When we describe a system we directly use hierarchical CP-nets – and we never construct the non-hierarchical equivalent. Analogously, we directly analyse a hierarchical CP-net – without having to construct the non-hierarchical equivalent. The existence of the non-hierarchical equivalent is, however, very important – because it tells us how to generalize the basic concepts and the analysis methods of non-hierarchical CP-nets to hierarchical CP-nets.

[38] Following our notational conventions we use PI to denote the set of all place instances in the entire CP-net (i.e. the union of PI_i over $i \in I$).

[39] The ^ operator denotes concatenation of two sequences.

Definition 3.6: Let a hierarchical CP-net HCPN = (S, SN, SA, PN, PA, FS, FT, PP) be given. Then we define the **equivalent non-hierarchical** CP-net to be CPN = (Σ^*, P^*, T^*, A^*, N^*, C^*, G^*, E^*, IN^*) where:

(i) $\Sigma^* = \Sigma$.

(ii) $P^* = PIG$.

(iii) $T^* = TI$.

(iv) $A^* = AI$.

(v) $\forall a^*=(a,id)\in AI\ \forall (p,t)\in P\times T$:

$$[\,N(a) = (p,t) \Rightarrow N^*(a^*) = ([(p,id)],(t,id)) \wedge N(a) = (t,p) \Rightarrow N^*(a^*) = ((t,id),[(p,id)])].$$[40]

(vi) $\forall p^*=[(p,id)]\in PIG$: $[C^*(p^*) = C(p)]$.

(vii) $\forall t^*=(t,id)\in TI$: $[G^*(t^*) = G(t)]$.

(viii) $\forall a^*=(a,id)\in AI$: $[E^*(a^*) = E(a)]$.

(ix) $\forall p^*=[(p,id)]\in PIG$: $[IN^*(p^*) = IN(p)]$.

(i) The non-hierarchical CP-net has the same set of colour sets as the hierarchical CP-net.

(ii) The non-hierarchical CP-net has a place for each place instance group of the hierarchical CP-net. This means that there is place for each place instance – unless that place instance either belongs to a fusion set (in which case the place instance is merged with the other members of the fusion set) or it is an assigned socket/port node (in which case it is merged with the place instance to which it is assigned).

(iii) + (iv) The non-hierarchical CP-net has a transition for each transition instance of the hierarchical CP-net. Analogously, it has an arc for each arc instance of the hierarchical CP-net.

(v) The basic idea behind the definition of the node function is that each page instance has the same arcs as the original page. This means that a place instance and a transition instance only can have connecting arcs if they belong to the same page instance – and in that case they have connecting arcs iff the original place and transition have. It should, however, be noticed that the node function (due to place fusion and socket/port assignment) maps into place instance groups (and not into individual place instances). This is done in such a way that each place instance group gets a set of surrounding arcs that is the union of those arcs that the corresponding place instances would have got (if they had not participated in any fusion or socket/port assignment).

(vi) The colour set of a place instance group is determined by the colour set of the participating places. From Def. 3.2 (v) + (vi) it follows that all these places must have identical colour sets.

(vii) The guard of a transition instance is determined by the guard of the corresponding transition.

(viii) The arc expression of an arc instance is determined by the arc expression of the corresponding arc.

(ix) The initialization expression of a place instance group is determined by the initialization expression of one of the participating places. From Def. 3.2 (v) + (vi) it

[40] We use [(p,id)] to denote the equivalence class to which (p,id) belongs.

follows that all these places must have initialization expressions which evaluate to the same value, and thus it does not matter which one we choose.

3.7 Dynamic behaviour of hierarchical CP-nets

Having defined the static structure of CP-nets we are now ready to consider their behaviour – but first we introduce the following notation, where E'(p',t') and E'(t',p') are called the **expressions** of (p',t') and (t',p'):[41]

- $\forall p'=(p,id_p)\in PI\ \forall t'=(t,id_t)\in TI$:
$$[\ id_p = id_t \Rightarrow (E'(p',t') = E(p,t) \wedge E'(t',p') = E(t,p)) \wedge$$
$$id_p \neq id_t \Rightarrow (E'(p',t') = E'(t',p') = 0)\ ].$$

Next we define token distributions, binding distributions, markings and steps:

Definition 3.7: A **token distribution** is a function M, defined on PIG such that $M(p^*)\in C(p)_{MS}$ for all $p^* = [(p,id)]\in PIG$ and a **binding distribution** is a function Y, defined on TI such that $Y(t^*)\in B(t)_{MS}$ for all $t^* = (t,id)\in TI$. We define TD_{HCPN}, BD_{HCPN}, $\neq$, $\leq$, $<$, $\geq$, $>$, $=$, **element** and **non-empty** in exactly the same way as for non-hierarchical CP-nets.

A **marking** is a token distribution and a **step** is a *non-empty* binding distribution. The set of all markings is denoted by M_{HCPN}, and the set of all steps is denoted by Y_{HCPN}. The **initial marking** M_0 is the marking where $M_0(p^*) = M_0(p)$[42] for all $p^* = [(p,id)]\in PIG$.

Finally we define enabling and occurrence:

Definition 3.8: A step Y is **enabled** in a marking M iff the following property is satisfied:
$$\forall p^*\in PIG: [\ \sum_{\substack{(t',b)\in Y \\ p'\in p^*}} E'(p',t')\langle b\rangle \leq M(p^*)].$$
We define **enabled** transition instances and **concurrently enabled** transition instances/bindings analogously to the corresponding concepts in a non-hierarchical CP-net.

(continues)

[41] We use p' and t' to denote a place instance and a transition instance, respectively.

[42] We use M_0 for two different purposes: On the left-hand side of the equation it denotes a marking of HCPN (i.e. a function defined on PIG). On the right-hand side it denotes the union constructed from the initial markings M_{0i} of the individual pages (i.e. a function defined on P). From the argument it will always be clear which of the two functions we deal with.

When a step is enabled in a marking M_1 it may **occur**, changing the marking M_1 to another marking M_2, defined by:

$$\forall p^* \in PIG: [M_2(p) = \left(M_1(p) - \sum_{\substack{(t',b) \in Y \\ p' \in p^*}} E(p,'t)'\langle b \rangle\right) + \sum_{\substack{(t',b) \in Y \\ p' \in p^*}} E(t',p')\langle b \rangle].$$

The first sum is called the **removed** tokens while the second is called the **added** tokens. Moreover we say that M_2 is **directly reachable** from M_1 by the occurrence of the step Y, which we also denote:

$$M_1 [Y\rangle M_2.$$

We define **occurrence sequences** and **reachability** analogously to the corresponding concepts for a non-hierarchical CP-net.

The following theorem shows that there is a one to one correspondence between the behaviour of a hierarchical CP-net and the corresponding non-hierarchical equivalent:

Theorem 3.9: Let HCPN be a hierarchical CP-net and CPN the non-hierarchical equivalent. Then we have the following properties:

(i) $\mathbb{M}_{HCPN} = \mathbb{M}_{CPN}$.

(ii) $\mathbb{Y}_{HCPN} = \mathbb{Y}_{CPN}$.

(iii) $\forall M_1,M_2 \in \mathbb{M}_{HCPN}\ \forall Y \in \mathbb{Y}_{HCPN}$: $[\ M_1 [Y\rangle_{HCPN} M_2 \Leftrightarrow M_1 [Y\rangle_{CPN} M_2\ ]$.

Proof: Property (i) is an immediate consequence of Def. 2.5, Def. 3.6 (ii) and Def. 3.7, while property (ii) is an immediate consequence of Def. 2.5, Def. 3.6 (iii) and Def. 3.7. Property (iii) follows from Def. 2.6, Def. 2.7, Def. 3.6 and Def. 3.8. The proof is omitted. It is straightforward but tedious – due to the large number of details which have to be considered.

4. Analysis of CP-nets

This chapter describes how CP-nets can be analysed. The most straightforward kind of analysis is simulation – which is very useful for the understanding and debugging of a system, in particular in the design phase and the early validation phases. There are, however, also more formal kinds of analysis – by which it is possible to *prove* that a given system has a set of desired properties (e.g. absence of dead-lock, the possibility to return to the initial state, and an upper bound on the number of tokens). This chapter contains a brief introduction to the main ideas behind the most important analysis methods and it contains references to papers in which the technical details of these methods can be found.

4.1 Simulation

Simulation can be supported by a computer tool or it can be totally manually (e.g. performed on a blackboard or in the head of the modeller). Simulation is similar to the debugging of a program, in the sense that it can reveal errors, but in practice never be sufficient to prove the correctness of a system. Some people argue that this makes simulation uninteresting and that the user instead should concentrate on the more formal analysis methods. We do not agree with this conclusion but consider simulation to be just as important and necessary as the more formal analysis methods.

In our opinion, all users of CP-nets (and other kinds of Petri nets) are forced to make simulations – because it is impossible to construct a CP-net without thinking about the effects of the individual transitions. Thus the proper question is not whether the modeller should make simulations or not, but whether he wants computer support for this activity. With this rephrasing the answer becomes trivial: Of course, we want computer support. This means that the simulation can be done much faster and with no errors. Moreover, it means that the modeller can use all his mental capabilities to interpret the simulation results (instead of using most of his efforts to calculate the possible occurrence sequences). Simulation is often used in the design phases and the early investigation of a system design (while the more formal analysis methods are used for the final validation of the design). In section 5.5 we give a detailed description of an existing CPN simulator.

4.2 Occurrence graphs

The basic idea behind occurrence graphs is to construct a graph which contains a node for each reachable state and an arc for each possible change of state. Obviously such a graph may, even for small CP-nets, become very large (and perhaps infinite). Thus we want to construct and inspect the graph by means of a computer – and we want to develop techniques by which we can construct a reduced occurrence graph without loosing too much information. The reduction can be done in many different ways:[43]

One possibility is to reduce by means of covering markings. This method looks for occurrence sequences leading from a system state to a larger system state (one with additional tokens) and the method guarantees that the reduced occurrence graph always becomes finite. The method has, however, some drawbacks. First of all it only gives a reduction for unbounded systems (and most practical systems are bounded). Secondly, so much information is lost by the reduction that several important properties (e.g. liveness and reachability) no longer are decidable. For more information see [18] and [40].

A second possibility is to reduce by ignoring some of the occurrence sequences which are identical, except for the order in which the elements occur. This method often gives a very significant reduction, in particular when the modelled system contains a large number of relatively independent processes. Unfortunately, it is with this method necessary to construct several different occurrence graphs (because the construction method depends upon the property which we want to investigate). For more information see [59].

A third possibility is to reduce by means of the symmetries which often are present in the systems which we model by CP-nets. To do this the modeller defines, for each colour set, an algebraic group of allowed bijections (each bijection defines a possible way in which the elements of the colour set can be interchanged with each other) – and

[43] For all the methods described below, it is possible to construct the reduced occurrence graph without first constructing the full occurrence graph.

this induces an equivalence relation on the set of all system states. The reduced occurrence graph only contains a node for each equivalence class and this means that it often is much smaller than the full occurrence graph. The reduced graph contains, however, exactly the same information as the full graph – and this means that the reduced graph can be used to investigate all the system properties which can be investigated by means of the full graph.[44] For more information see [30] and [35].

A fourth possibility is to construct an occurrence graph where each state is denoted by a symbolic expression (which describes a number of system states, in a similar way as the equivalence classes in method three). For more information see [9] and [42].

Finally, it is possible to construct occurrence graphs in a modular way. The model is divided into a number of submodels, an occurrence graph is constructed for each submodel, and these subgraphs are combined to form an occurrence graph for the entire model. For more information see [60].

When an occurrence graph has been constructed it can be used to prove properties about the modelled system. For bounded systems a large number of questions can be answered: Dead-locks, reachability and marking bounds[45] can be decided by a simple search through the nodes of the occurrence graph, while liveness and home markings can be decided by constructing and inspecting the strongly connected components. One problem with occurrence graph analysis is the fact that it, usually, is necessary to fix all system parameters (e.g. the number of sites in a ring protocol) before an occurrence graph can be constructed – and this means that the found properties always are specific to the chosen values of the system parameters. In practice the problem isn't that big: If we e.g. understand how a ring protocol behaves for a few sites we also know a lot about how it behaves when it has more sites.[46]

As described above, the occurrence graph method can be totally automated – and this means that the modeller can use the method, and interpret the results, without having much knowledge about the underlying mathematics. For the moment it is, however, only possible to construct occurrence graphs for relatively small systems and for selected parts of large systems. This doesn't mean that the method is uninteresting. On the contrary, the method seems to be a very effective way to debug new subsystems (because trivial errors such as the omission of an arc or a wrong arc expression often means that some of the system properties are dramatically changed). In the future, it may also be possible to use occurrence graph analysis for larger systems. This can be done by combining some of the reduction techniques described above – and by using the increased computing power of the next generations of hardware. In section 7.2 we

---

[44] The reduced occurrence graph (called an OE-graph) has more complex node and arc inscriptions than the full occurrence graph (called an O-graph). The OE-graph is a folded version of the O-graph, in the same way as a CP-net is a folding of the equivalent PT-net. The O-graph can be constructed from the OE-graph, but this is never necessary since the analysis can be done directly on the OE-graph.

[45] There are two kinds of marking bounds. Integer bounds only deal with the number of tokens while multi-set bounds also deal with the token colours. It can be proved that a place is integer bounded if and only if it is multi-set bounded. There are, however, situations in which the integer bound gives more information than the multi-set bound (and vice versa) – and thus it is useful to calculate both kinds of bounds.

[46] This is of course only true when we talk about the correctness of the protocol, and not when we speak about the performance.

describe the plans to implement a CPN tool to support the calculation and analysis of occurrence graphs.

4.3 Place and transition invariants

The basic idea behind place invariants is to find a set of equations which characterize all reachable markings, and then use these equations to prove properties of the modelled system (in a way which is analogous to the use of invariants in program verification). To illustrate the idea, let us consider the resource allocation system from Fig. 1. This system has the five place invariants shown below.[47] A place invariant is a linear sum of the markings of the individual places: Each place marking is by a weight function (attached to the place) mapped into a new multi-set. All the new multi-sets are over the same colour set and thus they can be added together – to give a **weighted sum** (determined from the given marking by the given set of weight functions).

The invariants use the three functions *P*, *Q* and *PQ* as weight functions. Each of them maps P-colours into multi-sets of E-colours. Intuitively, *P* "counts" the number of p-tokens (it maps (p,i) into 1`e and (q,i) into the empty multi-set). Analogously *Q* counts the number of q-tokens and *PQ* counts the number of p/q-tokens (i.e. the total number of tokens).[48] The invariants also use identity functions and zero functions as weights. The five invariants are satisfied for all reachable markings M (later we shall discuss how this can be proved). The right hand side of the invariants are found by evaluating the left hand side in the initial marking.

Intuitively PI_P and PI_Q tell what happens to the two different kinds of processes, while PI_R, PI_S and PI_T tell what happens to the three different kinds of resources. Each invariant can be seen as a way of extracting specific information – from the general information provided by the entire marking.

PI_P	$P(M(B) + M(C) + M(D) + M(E)) = 2`e$
PI_Q	$Q(M(A) + M(B) + M(C) + M(D) + M(E)) = 3`e$
PI_R	$M(R) + Q(M(B) + M(C)) = 1`e$
PI_S	$M(S) + Q(M(B)) + 2 * PQ(M(C) + M(D) + M(E)) = 3`e$
PI_T	$M(T) + P(M(D)) + (PQ + P)M(E) = 2`e$

The five invariants above can be used to prove that the resource allocation system doesn't have a dead-lock. *The proof is by contradiction:* Let us assume that we have a reachable state with no enabled transitions. From PI_P we know that there are two p-tokens distributed on the places B-E and from PI_Q that three are three q-tokens distributed on A-E. Now let us investigate in more detail where these tokens can be positioned. *First, assume that there are tokens on E:* Then T5 is enabled (and we have a contradiction with the assumption of no enabled transitions). *Secondly, assume that*

[47] There are many other place invariants for the system – but these are the most simple and useful.

[48] A weight function is usually specified as a function $f \in [C(p) \rightarrow A_{MS}]$ (i.e. a function from the colour set C(p) of the place into multi-sets over a colour set A). We always extend f to a function $f_{ext} \in [C(p)_{MS} \rightarrow A_{MS}]$ (for each multi-set $m \in C(p)_{MS}$ we calculate $f_{ext}(m)$ by adding the results of applying f to all the individual elements of m). Usually we do not distinguish between f and f_{ext} (and we use f to denote both functions).

there are tokens on C and/or D (and no tokens on E): From PI_S it follows that there can be at most one such token and then PI_T tells that there is at least one e-token on T (because P(M(D)) ≤ 1`e and $(PQ + P)$M(E) = empty). Thus T3 or T4 can occur. *Thirdly, assume that there are tokens on B (and no tokens on C, D and E):* From $PI_R$ it follows that there can be at most one q-token on B and then $PI_S$ tells us that there is at least two e-tokens on S (because $Q$(M(B)) ≤ 1`e and 2 * PQ(M(C) + M(D) + M(E)) = empty). Thus T2 can occur. *Now we have shown that it is impossible to position the two p-tokens (without violating the dead-lock assumption) – and thus we conclude that all reachable states have at least one enabled transition.* From the fact there are no dead-locks and the cyclic structure of the net, it is easy to prove other system properties e.g. that the initial marking is a home marking, that the system is live and that all reachable markings are reachable from each other.

Next let us discuss how we can find place invariants: As mentioned earlier, each CP-net has a function representation – which is a matrix where each element is a function (mapping multi-sets of bindings into multi-sets of token colours).[49] The matrix determines a homogeneous matrix equation and the place invariants are the solutions to this matrix equation (each solution is a vector of weight functions).[50] The matrix equation can be solved in different ways: One possibility is to translate the matrix of functions into a matrix of integers[51] for which the homogeneous matrix equation can be solved by standard Gauss elimination. Another, and more attractive, possibility is to work directly on the matrix of functions (this is, however, more complicated e.g. because some functions do not have an inverse). With both methods we do not explicitly find all solutions (there are usually infinitely many). Instead we find a basis from which all invariants can be constructed (as linear combinations). This leaves us with a second problem: How do we from the basis find the interesting place invariants – i.e. those from which it is easy to prove system properties? In our opinion, the best solution is to allow the user to tell the analysis program where to look for invariants – and thus calculate invariants in an interactive way. For more details about the calculation of invariants, see [12], [35], [44] and section 7.2.

Above, we have discussed how to calculate invariants by solving a homogeneous matrix equation. The problem is, however, often of a different nature – because we (instead of starting from scratch) already have a set of weight functions and just want to verify that these are invariants. This task is much easier and it can, without any problems, be done totally automatically. The potential invariants, to be checked, can be derived from the system specification and the modellers knowledge of the expected system properties. The potential invariants may be specified after the system design has been finished. It is, however, much more useful (and easier) to use CP-nets during the design and construct the invariants as an integrated part of the design (in the same way as a good programmer specifies a loop invariant at the moment he creates the loop). For this use of invariants it is important to notice that the check of invariants are constructive – in the sense that it, in the case of failure, is told where in the CP-net the

[49] The translation into the function representation can easily be defined by means of the lambda calculus. For more details see [35].

[50] Each solution to the matrix equation is a place invariant. The other direction is, however, only true when it is known that each occurrence element is enabled in at least one reachable marking.

[51] This is exactly the same as unfolding the CP-net to the behavioural equivalent PT-net.

problems are. Thus it is often relatively easy to see how the CP-net (or the invariant) should be modified.

Transition invariants are the duals of place invariants and the basic idea behind them is to find occurrence sequences with no effects (i.e. with the same start and end marking). Transition invariants can be calculated in a similar way as place invariants[52] - but, analogously to place invariants, it is more useful to construct them during the system design. Transition invariants are used for similar purposes as place invariants (i.e. to investigate the behavioural properties of CP-nets).

Place/transition invariants have several very attractive properties: First of all invariant analysis can be used for large systems – because it can be performed in a modular way[53] and does not involve the same kind of complexity problems as occurrence graph analysis. Secondly, invariant analysis can be done without fixing system parameters (e.g. the number of sites in a ring protocol). Thirdly, the the use of invariants during the design of a system will (as described above) usually lead to a better design. The main drawback of invariant analysis is that the skills, required to perform it, are considerably higher than for the other analysis methods. In section 7.2 we describe the plans to implement a CPN tool to support the interactive calculation and use of place/transition invariants.

4.4 Other analysis methods

CP-nets can also be analysed by means of reduction. The basic idea behind this method is to select one or more behavioural properties (e.g. liveness and dead-locks), define a set of transformation rules, prove that the rules do not change the selected set of properties, and finally apply the rules to obtain a reduced CP-net – which usually is so small that it is trivially to see whether the desired properties are fulfilled or not. Reduction methods are well-known for PT-nets and they have in [25] been generalised to CP-nets. A serious problem with reduction methods is that they often are non-constructive (because the absence of a property in the reduced net, usually, do not tell much about why the original net doesn't have the property).[54]

Most applications of CP-nets are used to design and validate the correctness of a system (e.g. whether the system executes the desired functions and whether it is deadlock free). CP-nets can, however, also be used to investigate the performance of a system (i.e. how fast it executes). To perform this kind of analysis it is necessary to specify the time consumption in the modelled system, and this can be done in many different ways: As a delay between the enabling and occurrence of a transition, a delay between the removal of input tokens and the creation of output tokens, or as a delay between the creation of a token and the time at which that token can be used. In all three cases, the delay may be a fixed value, a value inside a given interval, or a value deter-

[52] Transition invariants are found by solving a homogeneous matrix equation (obtained by transposing the matrix used to find place invariants). Each transition invariant is a solution to the matrix equation. The opposite is, however, not always true (even for "nice" CP-nets).

[53] As shown in [45] invariants can be obtained by the composition of existing invariants and this means that we can construct invariants of a hierarchical CP-net – from invariants of the individual pages.

[54] An exception is the reduction method to calculate place/transition invariants, mentioned in section 7.2. In this case it is, from the reduced net, possible to determine a set of the invariants for the original net – and this means that the analysis results can be interpreted in terms of the original net.

mined by a probability distribution. Performance analysis is often made by simulation, and we shall in section 7.1 briefly describe how this can be done. For some kinds of delays, it is also possible to translate the net model into a Markovian chain – from which analytic solutions of the performance values can be calculated. For more information about performance analysis see [47].

For ordinary Petri nets at least two other kinds of analysis methods are known. One method translates the net structure into a set of logical equations, transforms the equations by a general theorem prover, and obtains results above the behaviour of the system. For more information see [10]. The other method uses structural properties[55] of a Petri net to deduce behavioural properties. For more information see [3]. Unfortunately, neither of these methods have yet been generalized to CP-nets (or other kinds of high-level Petri nets).

5. Computer Tools for CP-nets

The practical use of Petri nets is, just as all other description techniques, highly dependent upon the existence of adequate computer tools – helping the user to handle all the details of a large description. For CP-nets we need an editor (supporting construction, syntax check and modification of CP-nets) and we also need a number of analysis programs (supporting a wide range of different analysis methods). The recent development of fast and cheap raster graphics gives us the opportunity to work directly with the graphical representations of CP-nets (and occurrence graphs). This chapter describes some existing CPN tools (the CPN editor and CPN simulator from [1]). In chapter 7 we discuss other kinds of CPN tools that are needed, but have not yet been fully developed.

5.1 Why do we need computer tools for CP-nets?

The most important advantage of using computerized CPN tools is the possibility to create *better results*. As an example, the CPN editor provides the user with a precision and drawing quality, which by far exceeds the normal manual capabilities of humans beings. Analogously, computer support for complex analysis methods (e.g. occurrence graphs) makes it possible to obtain results, which could not have been achieved manually (since the calculations would have been too error-prone).

A second advantage is the possibility to create *faster results*. As an example, the CPN editor multiplies the speed by which minor modifications can be made: It is easy to change the size, form, position and text of the individual net elements without having to redraw the entire net. It is also possible to construct new parts of a net by copying and modifying existing subnets. Analogously, analysis methods may be fully or partially automated. As an example, the manual construction of an occurrence graph is an extremely slow process – while it can be done on a computer in a few minutes/hours (even when there are several hundred thousand nodes).

[55] Structural properties are properties which can be formulated *without* considering the behaviour (i.e. occurrence sequences). In a CP-net structural properties may involve properties of the net structure, but also properties of the net inscriptions and the declarations.

A third advantage is the possibility to make *interactive presentations* of the analysis results. The CPN simulator makes it easy to trace the different occurrence sequences in a CP-net. Between each occurrence step, the user can (on the graphical representation of the CP-net) see the transitions which are enabled, and choose between them in order to investigate different occurrence sequences. Analogously, it is possible to make an interactive investigation of a complex occurrence graph – using an elaborated search system.

A fourth advantage is the possibility of *hiding technical aspects* of the CP-net theory inside the tools. This allows the users to apply complicated analysis methods without having a detailed knowledge of the underlying mathematics. Often the analysis is performed in an interactive way: The user proposes the operations to be done. Then the computer checks the validity of the proposals, performs the necessary calculations (which often are very complex) and displays the results.

For industrial applications the possibility of producing fast results of good quality - without requiring too deep knowledge of Petri net theory – is a necessary prerequisite for the entire use of CP-nets. Furthermore it is important to be able to use CP-nets together with other specification/implementation languages (we shall return to this question in chapters 6 and 7).

The remaining sections of this chapter describe the basic design criteria behind the CPN editor and the CPN simulator. For a more complete and detailed description the user is referred to [36]. The sections can also be seen as a list of design criteria which is relevant for all high-quality Petri net editors and simulators. There are a large number of different groups which work with the development of Petri net tools. Many of the tools are, however, still research prototypes – and for the moment it is only few of them which are able to deal with large high-level nets and are sufficiently robust to be used in an industrial environment. A list of available Petri net tools can be found in [17].

5.2 CPN editor

The CPN editor allows the user to construct, modify and syntax check hierarchical CP-nets. It is also easy to construct and modify many other kinds of graphs (but they can of course not be syntax checked).[56] All figures in this paper has been produced by means of the CPN editor.

A CP-net constructed by means of the CPN editor is called a **CPN diagram** and it contains a large number of different types of graphical **objects**. Each object is either a **node**, a **connector** (between two nodes) or a **region** (i.e. a subordinate of another object). Places and transitions are nodes, arcs are connectors, while all the net inscriptions are regions. As examples, colour sets and initialization expressions are regions of the corresponding places, guards of the corresponding transitions and arc expressions of the corresponding arcs.

The division of objects into nodes, connectors and regions reflects the fact that the CPN editor works with the **graph** (and not just an unstructured set of objects, as it is the case for most general purpose drawing tools, such as MacDraw™ or MacDraft™).

[56] In this paper, the word graph denotes the mathematical concept of a graph (i.e. a structure which consists of a set of nodes interconnected by a set of edges).

This is important because it means that the construction and modification of the CPN diagrams become much faster (and with more accurate results): When the user constructs a connector he identifies the source and destination nodes (and perhaps some intermediate points). Then the editor automatically draws the connector in such a way that the two endpoints are positioned at the border of the two nodes. When the user changes the position or size of a node the regions and surrounding arcs are automatically redrawn by the editor. A repositioning implies that the regions keep their relative position (with respect to the node). A resizing implies that the relative positions of the regions are scaled while their sizes are either unchanged or scaled (depending upon an attribute of each region). When a node is deleted the regions and arcs are deleted too. This is illustrated by Fig. 8 where the node X is first repositioned, then resized and finally deleted. Similar rules apply for the repositioning, resizing and deletion of arcs and regions.

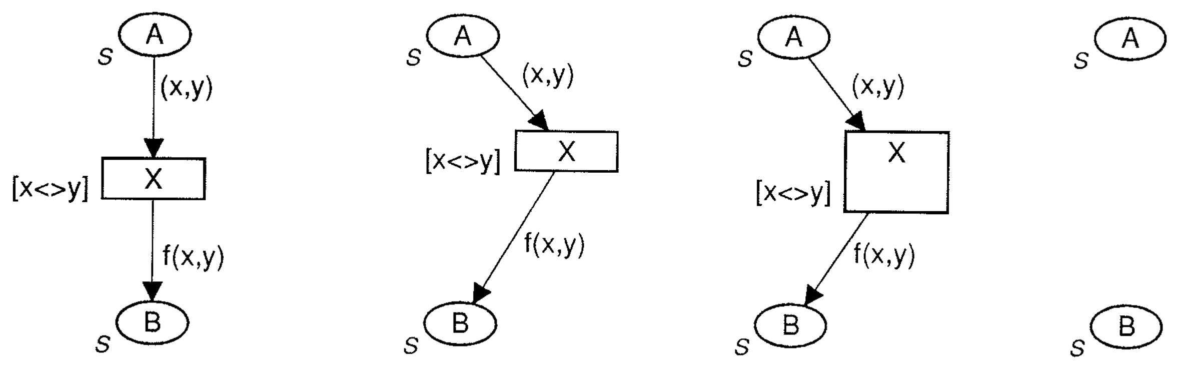


Figure 8. When a node is repositioned, resized or deleted, the regions and surrounding arcs are automatically updated.

In addition to the **CPN objects** (e.g. places, transitions, arcs and net inscriptions), which are formal parts of the model there may also be **auxiliary objects** which have no formal meaning but play a similar role as the comment facilities in programming languages. Finally, there are **system objects** which are special objects created and manipulated by the CPN editor itself. Each object has an **object type** and it should be noticed that it is the object type which determines the formal meaning of the object - independently of the object position and object form. The CPN editor distinguishes between nearly 50 different object types.

It is possible for the user to determine, in great detail, how he wants the CPN diagram to look. One of the most attracting features of CP-nets (and Petri nets in general) is the very appealing graphical representation, and it would be a pity to put narrow restrictions on how this representation can look (e.g. by making an editor in which the user cannot give two transitions different forms and/or sizes). In our opinion a good editor must allow the user to draw nearly all kinds of CP-nets which can be constructed by a pen and a typewriter. In the CPN editor each object has its own set of **attributes** which determine e.g. the position, shape, size, line thickness, line and fill patterns, line and fill colours and text type (including font, size, style, alignment and colour). There are 10-30 attributes for each object (depending upon the object type). When a new object is constructed the attributes are determined by a set of **defaults** (each object type has its own set of defaults). At any time the user can change one or more attributes for each individual objects.[57] Moreover, it is easy to change the defaults and it can be specified whether such changes apply to the current diagram or to future diagrams (or both).

In addition to the attributes the CPN editor (and in particular the CPN simulator) has a large set of **options** – which determines how the detailed functions in the editor are performed (e.g. the scroll speed, the treatment of duplicate arcs when two nodes are merged, and details about how the syntax check is performed). The difference between attributes and options is that the former relate to an individual object while the latter do not. Also options have defaults and these can be changed by the user.[58]

The CPN editor supports hierarchical CP-nets[59] and this means that each CPN diagram contains a number of pages. Each page is displayed in its own window (which in the usual way can be opened, closed, resized and repositioned). The relation between the individual pages is shown by the **page hierarchy** (which is positioned on a separate page called the **hierarchy page** and automatically maintained by the CPN editor). The page hierarchy is a graph in which each node represents a page and each connector a (direct) superpage/subpage relationship. The nodes are **page nodes** and each of them contains the corresponding page name and page number. The connectors are **page connectors** and each of them has a set of **page regions** containing the names of the involved supernodes.[60] The page objects can be moved and modified in exactly the same way as all other types of objects, and this means that the user can determine how the page hierarchy looks. The editor uses the line pattern of a page node to indicate whether the corresponding page window is active, open or closed. As an example, the ring network from Fig. 4-5 has a hierarchy page with three page nodes and one page arc, and it may look as shown in Fig. 9, where NetWork#10 is open but not active, Site#11 is closed, while the hierarchy page Hierarchy#10010 is open and active. In general, the hierarchy pages are much more complex.

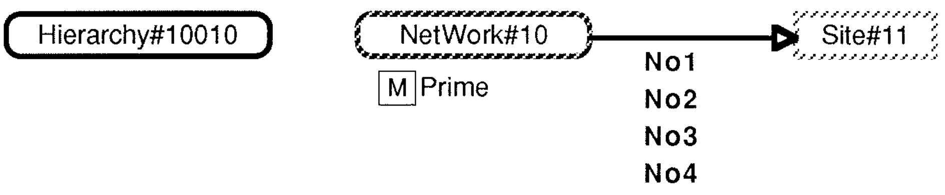


Figure 9. Hierarchy page for the ring network

The hierarchies in a CPN diagram can be constructed in many different ways – ranging from a pure top-down approach to a pure bottom-up: Part of a page can by a single editor operation be *moved to a new subpage:* The user selects the nodes to be moved and invokes the operation, then the editor checks the legality of the selection,[61] creates

57 This is done by specifying an explicit value, selecting another object (from which the attribute is copied) or by resetting the attribute to the current default.

58 For options a change in the default value only effects future diagrams (while a change in the option value itself, of course, effects the current diagram).

59 For the moment the CPN editor supports substitution transitions and place fusion. The other hierarchy constructs from [31] will be added later (some of them perhaps in an improved form).

60 Page nodes, page connectors and page regions are system objects.

61 All perimeter nodes (i.e. nodes with external arcs) must be transitions – in order to guarantee that the selection forms a closed subnet.

the new page, moves the subnet, creates the port nodes (by copying those nodes which were next to the selection), creates the border regions for the port nodes, constructs the necessary arcs between the the port nodes and the subnet, asks the user to create a new transition (which becomes the supernode for the new subpage), draws the arcs surrounding the new transition, creates a hierarchy inscription for it, and updates the hierarchy page. As it can be seen, a lot of rather complex checks, calculations and manipulations are involved. However, nearly all of these are automatically performed by the CPN editor. The user only selects the subnet and creates the new supernode.

There is also an editor operation to *turn an existing transition into a supernode* (by relating it to an existing page). Again most of the work is done by the editor: The user selects the transition and invokes the operation, then the editor makes the hierarchy page active and enters a mode in which the user by means of the mouse can select the desired subpage,[62] the editor creates the hierarchy inscription,[63] and updates the hierarchy page. To destroy the hierarchical relationship between a supernode and a subpage the user simply deletes the corresponding hierarchy inscription (or the corresponding page connector/region). It is also possible to *replace the supernode by the contents of the subpage:* This involves a lot of complex calculations and manipulations, but again all of them are carried out by the CPN editor. The user simply selects the supernode, invokes the operation and uses a simple dialogue box to specify how the operation shall be performed – e.g. he tells whether the page shall be deleted (in the case where no other supernodes exist).

The user works with a high-resolution raster graphical screen and a mouse.[64] The CPN diagram under construction can be seen in a number of windows (where it looks as close as possible to the final output obtained by a matrix or laser printer). The editor is menu driven and have self-explanatory dialogue boxes (as known e.g. from many Macintosh programs). The user moves and resizes the objects by direct manipulation - i.e. by means of the mouse (instead of typing coordinates and object identification numbers on the keyboard). This also applies to the pages which can be opened, closed, scrolled and scaled by means of the corresponding page node. When the user deletes a page node the corresponding page is deleted (after a confirmation). Analogously, the deletion of a page connector or a page region means that the corresponding hierarchical relationship is destroyed (and thus the corresponding supernodes become ordinary transitions).

One important difference between the CPN editor and many other drawing programs is the possibility to work with **groups** of objects. This means that the user is able to select a *set* of objects and *simultaneously* change the attributes, delete the objects, copy them, move them or reposition them (e.g. vertical to each other). The user can select groups in many different ways (e.g. by dragging the mouse over a rectangular area or by pressing a key while he points to a sequence of objects). The CPN editor

[62] When the mouse is moved over a page node it blinks – unless it is illegal (because selection of it would make the page hierarchy cyclic). Only blinking page nodes can be selected.

[63] The user can ask the editor to try to deduce the port assignment by means of a set of rules (which looks at the node names, the port types and the arcs between the transition and the sockets).

[64] For the moment the CPN tools are implemented on Macintosh, SUN and HP machines – and they can easily be moved to other machines running UNIX and X-Windows. It is recommended, but not necessary, to have a large colour screen.

allows the user to perform operations on groups in exactly the same way as they can be performed on individual objects[65] – and this has the same effect as when the corresponding operation is performed on each group member one at a time. All members of a group has to belong to the same page and be of the same kind – i.e. all be nodes, all be connectors or all be regions.[66] Otherwise there are no restrictions on the way in which groups can be formed. The group facility has a very positive impact upon the speed and ease by which editing operations are performed. By selecting a group of page nodes it is possible to work on several pages at the same time.

In the design of the CPN editor it has been important for us to make it as flexible as possible. As described above, this means that it is possible to construct CPN diagrams which *look* very different. However, it also means that each diagram can be *created* in many different ways. One example of this principle is the many different ways in which the page hierarchy can be constructed. Another example is the fact that the CPN editor allows the user to construct the various objects in many different orders: Some users prefer first to construct the net structure (i.e. the places, transitions and arcs). Later they add the net inscriptions (i.e. the CPN regions) – and doing this they either finish one node at a time or one kind of CPN regions at a time, and they either type from scratch or copy from existing regions. Other users prefer to create templates - e.g. a place with a colour set region and an initialization region. Then they create the diagram by copying the appropriate templates to the desired positions and modifying the texts (if necessary).[67] Finally, most users work in a way which is a mixture of the possibilities described above. We think that this kind of flexibility – where the user controls the detailed planning of the editing process – is extremely important for a good tool. Thus the CPN editor has been designed to allow most operations to be performed in several different ways.

A CPN diagram contains many different kinds of information and this means that the individual pages very easy become cluttered. To avoid this the user is allowed to make objects invisible (without changing the semantics of the objects). As an example the user may hide all colour set regions and instead indicate the colour sets by giving the corresponding places different attributes (e.g. different line patterns/colours). In this case it is still the invisible regions that determine the formal behaviour, and it is the responsibility of the user to keep the pattern/colour coding of the places correctly updated (there are several facilities in the CPN editor which helps him in this task). Another facility, which also helps avoiding cluttered diagrams, is the concept of key and popup regions, which are used for a number of different object types (both in the editor and the simulator). The idea is very simple: Instead of having a single region (containing a lot of information) we have both a **key region** (which is a region of the object to which we want to attach the information) and a **popup region** (which is a region of the key). The key region is small (it usually only contains one or two characters) and its main purpose is to give access to the popup region which contains the ac-

[65] There are only very few operations which do not make sense for groups.

[66] In a later version of the CPN editor, we may allow a group to have members from different pages. This is easy to implement and it creates no conceptual problems. It is, however, unlikely that we will allow mixed groups. The reason is that the semantics of many operations then become a bit obscure.

[67] When the user copies a node, the editor automatically copies the regions. Analogously, when a group of nodes is copied, the internal connectors (between two members of the group) are copied too.

tual information. A double click on the key region makes the popup region visible/invisible and in this way it is extremely easy to hide and show large amounts of information. For examples of key/popup regions see the hierarchy regions in Fig. 4 (with the HS-keys), the border regions in Fig. 5 (with the B-keys) and the marking regions in Fig. 3 (containing the current marking). It should be noticed that the use of key/popup regions is more general than the use of popup windows (in which information can be displayed on demand). The difference is that the popup regions are objects in the diagram itself and thus the user can leave all of them or some of them permanently visible. Actually, it is an attribute of each key region that determines whether the corresponding popup region is visible or not.[68]

It should be noticed that the generality of the CPN editor means that the user can create very confusing CPN diagrams. As examples, it may be impossible to distinguish between auxiliary objects and CPN objects (because they have been given identical attributes), transitions may be drawn as ellipses while places are boxes, and some or all of the objects may be invisible – just to mention a few possibilities. We do *not* believe it is sensible to try to construct a tool which makes it *impossible* to produce bad nets. Such a tool will, in our opinion, inevitably be far too rigid and inflexible. However, we do of course believe that the tool should make it easy for the user to make good nets.

There are many other facilities in the CPN editor: Operations to open, close, save and print diagrams.[69] An operation which allows the editor to import diagrams created by other tools (e.g. SADT diagrams created by the IDEF/CPN tool described in section 6.3). The standard Undo,[70] Cut, Copy, Paste and Clear operations known e.g. from the Macintosh concept. Operations to define fusion sets, specify port nodes and perform port assignments. Operations to create many different types of auxiliary objects (e.g. connectors, boxes, rounded boxes, ellipses, polygons, wedges and pictures[71]). Operations to turn auxiliary objects into CPN objects (and vice versa). An operation to syntax check the CPN diagram (and other operations to start/stop the ML compiler, see section 5.3). A large set of operations to change attributes and options – and their defaults. Operations which assist the user to select the correct object (when many are close to each other or on top of each other), move objects to another position (on the same page or on another page), change object size (e.g. to fit the size of the text in the object), change object shape (e.g. from ellipse to box),[72] merge a group of nodes into a

68 For efficiency reasons the popup region can also be missing. In this case a double click on the key implies that the popup is generated (with the correct information) and becomes visible.

69 It is also the intention to allow the user to save part of a diagram and later load it into another diagram. In this way it will be possible to create libraries of reusable submodels. This facility is, however, not yet implemented.

70 For the moment, Undo only works for a limited set of operations.

71 A picture is a bit map which is obtained from a CPN diagram (by copying part of a page) or from another program (via the clipboard). Pictures makes it easy to work with icons.

72 All objects can take many different shapes. Nodes and regions can e.g. be boxes, rounded boxes, ellipses, polygons, wedges and pictures. Connectors can be single headed, double headed and without heads. As an example, of a creative use of this generality, it is possible to let a substitution transition be a picture which is a diminished version of the corresponding subpage.

single node, duplicate a node[73], hide and show regions and change the graphical layering of the objects. Operations to redraw the page hierarchy – when this has become too cluttered (e.g. because the user has made a number of manual changes to the automatic layout proposed by the CPN editor). Operations to select groups (e.g. by means of fusion sets, text searches and object types).[74] Operations to search for specified text strings and replace them by others (either in the entire diagram, on a single page, or in one or more selected objects). Operations to search for matching brackets, create hyper text structures,[75] and copy the contents of external text files into nodes (and vice versa). A large number of alignment operations. Some of these make it easy to position nodes and regions relative to each other (e.g. vertically below each other, with equal distances, on a circle, with the same center, etc.). Others make it easy to create arcs with right angles and vertical/horizontal segments.

The CPN editor can be used at many different skill levels. Casual and novice users only have to learn and apply a rather small subset of the total facilities. The more frequent and experienced users gradually learn how to use the editor more efficiently: All the more commonly used commands can be invoked by means of key shortcuts, and these can be changed by the users. Many commands have one or more modifier keys, allowing the user, in one operation, to do things which otherwise would require several operations. The user can create a set of templates (e.g. a set of nodes with different attributes and object types). These nodes can then be positioned on special palette pages, from where they, in one operation, can be copied to the different pages of a diagram. In this way it is easy to make company standards for the graphics of CPN diagrams.

To make it easier to use the CPN editor we have tried to make the user interface as consistent and self-explanatory as possible. To do this, we have defined a set of concepts allowing us to give a precise description of the different parts of the interface: As an example, a list box with a scroll bar can behave in many slightly different ways: It may be possible to select only a single line at a time, a contiguous set of lines, an arbitrary set of lines, or no lines at all – and when the dialogue box is opened, the list box may have the same selection as last time, have the first line selected, have no lines selected, or have a selection which depends upon the current selection in the diagram. Hopefully, this simple example demonstrates that it is important to identify the possibilities – and use them in a consistent way.

When the user creates a CPN diagram, the editor stores all the semantic information in an abstract data base – from which it easily can be retrieved by the CPN simulator (and other analysis programs). The abstract data base was designed as a relational data base but for efficiency implemented by means of a set of list structures (making the most commonly used data base operations as efficient as possible). The existence of the abstract data base makes it much easier to integrate new/existing editors and analysis programs with the CPN tools – and for this purpose there are three sets of predeclared functions: The first set makes it possible to read the information which is present in the abstract data base (e.g. get information about the colour set of place). The second set

[73] The new node get a set of regions and connectors which are similar to the original node. By using the command on a group of nodes, it is possible to get a subnet which is identical to an existing subnet (and has the same connectors to/from the environment).

[74] Some of the group selection facilities are not yet implemented.

[75] This facility is not yet fully implemented.

makes it possible to create auxiliary objects (which have a graphical representation but no representation in the abstract data base). Finally, the third set makes it possible to convert auxiliary objects to CPN objects (which means that the abstract data base is updated accordingly). Using these three sets of predeclared functions it is a relatively straightforward task to write programs which translates textual/graphical representations of a class of Petri nets (or another formalism with a well-defined semantics) into CPN diagrams – and vice versa.

Finally, it should be mentioned that the CPN editor is designed to work with large CPN diagrams – i.e. diagrams which typically have 50-100 pages, each with 5-25 nodes (and 10-50 connectors plus 10-200 regions).

5.3 Inscription language for CP-nets

When the user creates a CPN diagram he simultaneously creates a *drawing* and a *formal model*. The behaviour of the formal model is determined by the objects, their object types, the relationships between the objects[76], and the text strings inside the objects. Obviously these text strings need to have a well-defined syntax and semantics, and this is achieved by using a programming language called Standard ML (SML). It is by means of this language we declare colour sets, functions, operations and specify arc expressions and guards. SML has been developed by a group at Edinburgh University and it is one of the most well-known functional languages. For details about SML and functional languages, see [26], [27], [50] and [63].

By choosing an existing programming language we obtained a number of advantages. First of all we got a much better, more general and better tested language than we could have hoped to develop ourselves.[77] Secondly, we only had to port the compiler to the relevant machines and integrate it with our editor (instead of developing it from scratch).[78] Thirdly, we can use the considerable amount of documentation and tutorial material which already exists for SML (and for functional languages in general).

Why did we choose SML? First of all, we need a functional language: Arc expressions and guards are not allowed to have side effects and when a CP-net is translated into matrix form (e.g. for invariant analysis) the arc expressions and guards are, via lambda expressions, translated into functions. Secondly, we need a strongly typed language: Because CP-nets use colour sets in a way which is analogous to types in programming languages. Thirdly, we need a language with a flexible and extendible syntax: This makes it possible to allow the user to write arc expressions and guards in a form which is very close to standard mathematics (as an example, multi-set plus is de-

[76] There are many different kinds of relationships – e.g. the relationship between connectors and their source/destination nodes, between nodes and their regions, and between substitution transitions and their subpages.

[77] The development of a new programming language is a very slow and expensive process that requires resources comparable with the entire CPN tool project.

[78] The CPN tools use two different SML compilers. On the Macintosh we use the original compiler developed at Edinburgh University. On the Unix machines we use a more modern compiler developed by AT&T. It is also possible to run the graphics on one machine and the SML compiler on another (connected to the first by a local area network).

noted by "+").[79] SML is only one out of a number of languages which fulfil the three requirements above. SML was chosen because it was one of the best known, it had commercially available compilers, and some of us already had good experiences with the language.

We have many times been amazed by the high quality of SML, the generality of the language, and the ease by which complex programs can be written.[80] Thus we consider the choice of SML as one of the most successful design decisions in the CPN tool project. This choice has given us a very powerful and general inscription language and it has saved a lot of implementation time. As we shall see in section 5.5, the use of SML also makes it easy to make a smooth integration between the net inscriptions of a CP-net and code segments (which are sequential pieces of code attached to the individual transitions and executed each time a binding of the transition occurs).

To make it easier for the user we have made three small extensions of SML – and this yields a language called CPN ML: As the first extension, syntactical sugar has been added for the declaration of colour sets. This makes it easy to declare the most common kinds of colour sets, and it also means that a large number of predeclared functions and operations can be made accessible, just by including their names in the colour set declaration.[81] As examples, each enumeration type has a function mapping colours into ordinal numbers, each product type has a function mapping a set of multi-sets into their product multi-set, and each union type has a set of functions performing membership tests. SML allows the user to declare integers, reals, strings, enumerations, products, records, discrete unions and lists – and nest the type constructors arbitrarily inside each other. As an example we may declare the following colour sets (which should be rather self-explanatory):

```
color Name = string;
color NameList = list Name;

color Year = int;
color Month = with  Jan | Feb | Mar | Apr | May | Jun |
                    Jul | Aug | Sep | Okt | Nov | Dec;
color Day = int with 1..31;
color Date = product Year * Month * Day;

color Person = record name : Name * BirthDay : Date * Children : NameList;
```

79 The "+" operator is infixed (i.e. written between the two arguments). It is polymorphic (i.e. it works for multi-sets over all different types) and it is overloaded (i.e. it uses the same operator symbol as integer plus and real plus).

80 Much of the more intrinsic code of the CPN simulator is written in SML. In particular, all the code that calculates the set of enabled bindings. This code is rather complex: It defines a function which maps an arbitrary set of arc expressions (plus a guard) into a function mapping a set a multi-sets into a set of enabled bindings.

81 This convention saves a lot of space in the ML heap, because it turns out that most CPN diagrams only use few of the predeclared functions. A later version of the CPN editor will automatically detect the predeclared functions applied by the user (and then it will no longer be necessary to list their names).

Via the syntactic sugar, it is in CPN ML easy to declare colour sets from all the SML types mentioned above (and from subranges, substypes, and indexed types, which do not exist as standard SML types). In SML it is also possible to declare function types and abstract data types. However, such types do not have an equality operation and thus it does not immediately make sense to use them as colour sets (because you cannot talk about multi-sets without being able to talk about equality).[82]

As the second extension, we have added syntax which allows the user to declare the CPN variables – i.e. the typed variables used in arc expressions and guards. This extension is necessary because SML do not have variable declarations (in SML a value is bound to a name and this determines the current type of the name; later the name may get a new value and a new type).

As the third extension, we have added syntax which allows the user to declare three different kinds of reference variables. This is a non-functional part of SML and we only allow reference variables to be used in code segments. We distinguish between global, page and instance reference variables – in the same way as we distinguish between global, page and instance fusion sets: A global reference variable can be used by all code segments in the entire CPN diagram, while a page and instance reference variable only can be used by the code segments on a single page. A page reference variable is shared by all instances of the page, while an instance reference variable has a separate value for each page instance.

SML (and thus CPN ML) can be viewed as being a syntactical sugared version of typed lambda calculus, and this means that it is possible to declare arbitrary mathematical functions (as long as they are computable). It should be noticed that the use of SML gives an immense generality: The user can declare arbitrarily complex functions[83] and, if he wants, he can turn them into operations (i.e. use infix notation). This generality has been heavily used in the implementation of the CPN tools. Multi-sets are implemented as a polymorphic type constructor "ms" which maps an arbitrary type *A* into a new type, denoted by *A ms* and containing all multi-sets over *A*. Then we have declared a large number of polymorphic and sometimes overloaded operations/functions – by which multi-sets can be manipulated (e.g. operations to add and subtract multi-sets and functions to calculate the coefficients and the size of multi-sets).

The generality of the CPN ML language means that some legal CPN diagrams cannot be handled by the CPN simulator. As an illustration consider the transition in Fig. 10, where x is a CPN variable of type X, while $f \in [X \rightarrow A]$ and $g \in [X \rightarrow B]$ are two functions. To calculate the set of all enabled bindings for such a transition it is either necessary to try all possible values of X or use the inverse relations of f and g (and neither is possible, in general – because X may have two may values and the inverse relations may be unknown).

[82] The user can, with some extra work, use an arbitrary ML type as a colour set – as long as the standard equality operator "=" exists (and the type is non-polymorphic). In this way it is possible to declare abstract data types and turn them into colour sets. Details are outside the scope of this paper.

[83] Many CPN diagrams use recursive functions defined on list structures.

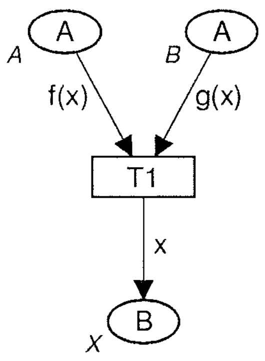


Figure 10. A syntactically legal CPN transition which cannot be handled by the CPN simulator

To avoid such problems the CPN simulator demands that each CPN variable, used around a transition, must appear either in an input arc expression without functions or operations[84] (then the possible values can be determined from the marking of the corresponding input place), be determinable from the guard, have a small colour set (in which case all possibilities can be tried),[85] or only appear on output arcs (in which case all possible values can be used). It is very seldom that these restrictions present any practical problems. Most net inscriptions, written by a typical user, fulfil the restrictions – and those which do not, can usually be rewritten by the user, without changing the semantics. As an example, consider the three transitions in Fig. 11. None of these can be directly handled by the CPN simulator. The first transition is identical to the transition in Fig. 10. The second transition has a guard which is a list of boolean expressions, and this means that each of the expressions must be fulfilled. The third transition uses the function exp(x,y) which takes two non-negative integers as arguments and returns x^y.

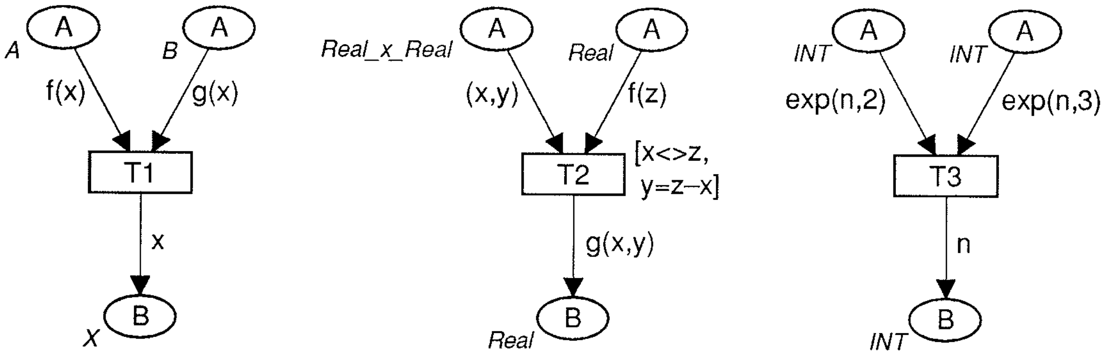


Figure 11. Three transitions which cannot be handled by the CPN simulator

Now let us assume that f has an inverse function f1 ∈ [A→X]. Then we can, as shown in Fig. 12, rewrite the three transitions – so that their semantics is unchanged and they can be handled by the CPN simulator. In the first transition z is a variable of type A. In the second transition z can now be determined from the guard – because there is an equality in which z appears on one side (alone or in a matchable pattern) while the value of

[84] The arc expression is allowed to contain matchable operations such as the tuple constructor („) in (x,y,z), the list constructor :: in head::tail, and the record constructor {„} in {se=S(inst()), re=r, no=n}. It is also allowed to contain multi-set "+" and "`".

[85] Intuitively a small colour set is a type with few values. A precise definition can be found in [36].

the other side is known (x and y are bound by one of the input arc expressions). In the third transition the function sq(x) takes a non-negative integer as argument and it returns the integer which is closest to √x.

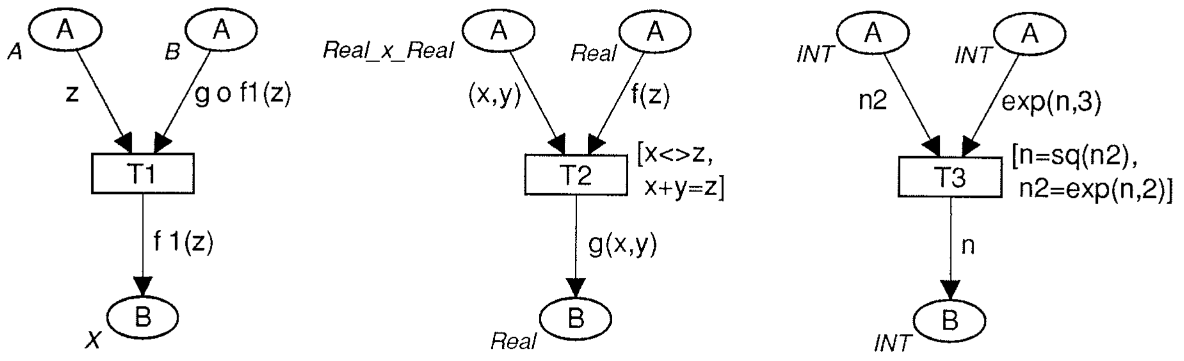


Figure 12. Three transitions that are behaviourally equivalent to those in Fig. 11 and which can be handled by the CPN simulator

It is important to understand that the general definition of CP-nets talks about expressions and colour sets – without specifying a syntax for these. It is only when we want to implement a CPN editor and a CPN simulator (and other kinds of CPN tools) that we need a concrete syntax. Thus it is for the CPN tools, and not for CP-nets in general, that CPN ML has been developed. Other implementations of CP-nets may use different inscription languages – and still they deal with CP-nets.

5.4 Syntax check

The CPN editor is syntax directed – in the sense that it recognizes the structure of CP-nets and prevents the user to make many kinds of syntax errors. This is done by means of a large number of **built-in** syntax restrictions. All the built-in restrictions deal with the net structure and the hierarchical relationships. As examples, it is impossible to make an arc between two transitions (or between two places), to give a place two colour set regions (or give a transition a colour set region), to create cycles in the substitution hierarchy, and to make an illegal port assignment (involving nodes which aren't sockets/ports or are positioned on a wrong page).

The CPN editor also operates with **compulsory** syntax restrictions. These restrictions are necessary in order to guarantee that the CPN diagram has a well-defined semantics – and thus they must be fulfilled before a simulation (and other kinds of behavioural analysis) is performed. Many of the compulsory restrictions deal with the net inscriptions and thus with CPN ML. As examples, it is checked that each colour set region contains the name of a declared colour set *A* (and that all surrounding arc expressions have a type which is identical to either *A* or *A ms),* that all members of a fusion set have the same colour set and equivalent initialization expressions, and that all identifiers in arc expressions and guards are declared (e.g. as CPN variables or functions). Many of the compulsory syntax restrictions could have been implemented as built-in restrictions. This would, however, have put severe limits on the way in which a user can construct an edit a CPN diagram. As examples, we could have demanded that each place always has a colour set (and this would mean that the colour set has to be specified at the moment the place is created) and we could have demanded that each arc expression always is of the correct type (and this would mean that a colour set cannot be changed without simultaneously changing all the surrounding arc expressions).

Finally, the CPN editor operates with **optional** syntax restrictions.[86] These are restrictions which the user imposes upon himself – e.g. because he knows that he usually does not use certain facilities of the editor and wants to be warned when he does (in order to check whether this was on purpose or due to an error). As examples, it can be checked whether port assignments are injective, surjective and total, whether all arcs have an explicit arc expression (otherwise they by default evaluate to the empty multi-set) and whether the place/transition names are unique (on each page).

All the type checking is done by the SML compiler and it is the error messages of this compiler which is presented to the user (together with a short heading produced by the CPN editor). The fact that these messages are easy to understand and uses CP-net terminology tells a lot about the generality and quality of SML. To illustrate this, let us imagine that we, in Fig. 1, change the arc expression between A and T1 from (x,i) to x. This will result in an error message which looks as follows:[87]

```
C.11 Arc Expression must be legal
Type clash in: x:((P)ms)
Looking for a: P ms
I have found a: U
«««135»»»
```

To speed up the syntax check we avoid duplicate tests: As an example, the same arc expression may appear at several arcs and it is then only checked once (provided that the places have identical colour sets). We also apply incremental tests: When the user changes part of a CPN diagram as little as possible is rechecked. Changing an arc expression or a guard means that the use of variables in the code segment must be rechecked. Changing a colour set means that the initialization expression and all surrounding arc expressions have to be rechecked.[88] Changing the global declaration node (which contains the declarations of colour sets, functions, operations, and CPN variables), unfortunately means that the entire CPN diagram has to be rechecked. To avoid using too much time for such total rechecks, the CPN editor allows the user to add a temporal declaration node which extends the declarations of the global declaration node.[89]

The CPN editor allows the user to give each page, transition and place a name (i.e. a text string) and a number (which must be non-negative).[90] It should, however, be un-

[86] Optional syntax checks are not implemented in the current version of the CPN editor.

[87] This is how the error message looks when the SML compiler runs on a Macintosh (on a Unix system another SML compiler is used, and thus the error messages looks a little bit different). C11 means that it is the 11th kind of compulsory restriction, while «««135»»» is a hyper text pointer which allows the user to jump to the error position (i.e. to the arc with the erroneous arc expression).

[88] If the place belongs to a fusion set or is an assigned port/socket it also has to be checked whether the restrictions in Def. 3.2 (v) and (vi) still are satisfied.

[89] It is also possible, but not recommended, to use the temporal declaration node to overwrite existing declarations.

[90] In the current version it is not possible to give transitions and places a number.

derstood that these names have no semantic meaning.[91] Names are used in the feedback information from the editor to the user (e.g. in the page hierarchy and in the hierarchy inscriptions). To make this information unambiguous it is recommended to keep names unique,[92] but this is not enforced (unless the user activates an optional syntax restriction). Many users have a large number of transitions and places with an empty name (and this is no problem, as long as these nodes are not used in a way which generates system feedback).

The possibility of performing an automatic syntax check means that the user has a much better chance of getting a consistent and error-free CPN diagram. This is very useful – also in situations where the user isn't interested in making a simulation (or other kinds of machine assisted behavioural analysis).

5.5 CPN simulator

The CPN editor and CPN simulator are two different parts of the same program and they are closely integrated with each other: In the editor it is possible to prepare a simulation (e.g. change the many options which determine how the simulation is performed). In the simulator it is possible to perform simple editing operations (those which change the attributes of objects without changing the semantics of the model).[93]

The CPN simulator is able to work with large CP-nets, i.e. CP-nets with 50-500 page instances, each with 5-25 nodes. Fortunately, it turns out that a CP-net with 100 page instances, typically, simulates nearly as fast as a CP-net with only a single page instance (measured in terms of the number of occurring transitions). This surprising result is due to the fact that the CPN simulator, during the execution of a step, goes through three different phases: First it makes a random selection between enabled transitions, then it removes and adds tokens at the input/output places of the occurring transitions, and finally it calculates the new enabling. The first of these phases is fast (compared to the others), the second is independent of the model size and the third only depends upon the model size to a very limited degree. This is due to the fact that the enabling and occurrence rule of CP-nets are strictly local – and this means that it only is the transitions in the immediate neighbourhood of the occurring transitions that need to have their enabling recalculated.[94] Without a local rule the calculation of the new enabling would grow linearly with the model size and that would make it very cumbersome to deal with large systems. We have not yet tried to work with very large systems (e.g. containing 10.000 page instances) but our present experiences tell us that the upper limit is more likely to be set by the available memory than by the processor speed.

The user must be able to follow the on-going simulation – and it is obvious that no screen (or set of screens) will be able simultaneously to display all page instances of a large model. Like the editor, the CPN simulator uses a window for each page and in

[91] In the current version of the CPN editor the names of fusion sets play a semantic role, and thus they have to be unambiguous. This will be changed in a later version.

[92] For places and transitions it is sufficient to demand the names to be unique on each individual page.

[93] In one of the next versions of the CPN simulator we will also allow the user to make changes that modify the behaviour of the model – as long as these changes cannot make the current marking illegal.

[94] When the neighbourhood of an occurring transition is defined, fusion sets and port/socket assignments must be taken into consideration.

this window the simulator displays the marking of one of the corresponding page instances. The user can see the names of the other page instances and switch to any of these. When a transition occurs the simulator automatically opens the corresponding page window (if necessary), brings it on top of all other windows, switches to the correct page instance, and scrolls the window so that the transition becomes visible. The user can, however, tell that he doesn't want to observe all page instances. In that case the simulator still executes the transitions of the non-observed page instances but this cannot be seen by the user (unless the relevant part of corresponding page instance happens to be visible on the screen without any rearrangements). The user can set breakpoints and in this way ask the simulator to pause before, during, and/or after each simulation step. Breakpoints can be preset or added on the fly, i.e. at any point during a simulation. At each breakpoint the user can investigate the system state (and decide whether he wants to continue or cancel the remaining part of the step).

It is possible to simulate a selected part of a large CPN diagram (without having to copy this part to a separate file, which would give all the usual inconsistency problems). This is achieved by allowing the user to change the multi-set of prime pages and tell that certain page instances should be temporarily ignored. When a page instance is ignored it is no longer generated, and this means that the corresponding direct supernode becomes an ordinary transition with enabling and occurrence calculated in the usual way (i.e. by means of the surrounding arc expressions and guard). As a short hand, it is also possible to ignore a page and this means that all instances of the page are ignored.

When we simulate a CP-net it is sometimes convenient to be able to equip some of the transitions with a **code segment** – i.e. a sequential piece of code which is executed each time a binding of the transition occurs. Each code segment has a code guard, an input pattern, an output pattern and a code action. The code guard replaces the corresponding guard (in a simulation with code segments). A missing code guard means that the ordinary guard is used. The input pattern contains some of the CPN variables of the transition, and this indicates that the code action is allowed to use (but not update) these variables. Analogously, the output pattern contains some of the CPN variables (but only those which do not appear in the input arc expressions and the code guard) and this indicates that the binding of these variables is determined by the code segment. Finally, the action part is an SML expression (with the same type as the output pattern).[95] The action part may declare local variables, share reference variables with other code segments, use the CPN variables from the input pattern and manipulate input/output files. When the transition occurs the action part is evaluated and the resulting value determines the binding of the CPN variables in the output pattern. It should be noticed that the code segment is executed once for each occurring binding, and this means that it may be executed several times in the same step.[96]

[95] In a later version of the CPN simulator it will also be possible to use other programming languages in the code action, e.g. C++, Pascal and Prolog.

[96] The order of these executions is non-deterministic (but it is guaranteed that each execution is indivisible, in the sense that it is finished before the next is started).

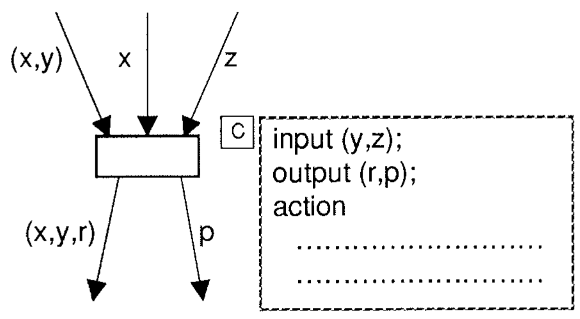


Figure 13. A simple example of a code segment

Code segments can be used for many different purposes: They can be used to gather statistical information about the simulation: It is easy to dump the value of all occurring bindings on a file (which then later can be analysed e.g. by means of a spread sheet program). It is also possible to use the graphic routines of the CPN tools (which can be invoked via predeclared SML functions) and in this way make a visual representation of the simulation results (as an example, it is easy to make a window which has an node for each site in a communication network and a connector for each pair of sites which are engaged in a communication).[97] Code segments also allow interactive user input, and they can be used to communicate with other programs (as an example, it is possible to run different parts of a very large CPN model on separate computers and let the different submodels communicate via input/output statements).

Although code segments are extremely useful for many purposes, they also have severe limitations. This is due to the fact that they allow the occurrence of transitions to have side effects and allow bindings to be determined by input files (and other kinds of user input). This means that it doesn't make sense to talk about occurrence graphs for CP-nets with code segments, and it also becomes more difficult to use the invariant method for such nets (because the relation between the CPN variables surrounding a transition may be determined by the code action, instead of the arc expressions). For this reason it is important to have a well-understood relationship between a CPN diagram executed with code segments and the same CPN diagram executed without code segments. This is one of the main reasons for the introduction of the input and output patterns.

It is possible to perform both **manual** and **automatic** simulations. In a manual simulation the simulator calculates and displays the enabling, the user chooses the occurrence elements (i.e. the transitions and bindings) to be executed and finally the simulator calculates the effect of the chosen step. During the construction of a step, the simulator assists the user in many different ways: First of all, the simulator always shows the current enabling (and updates it each time a new occurrence element is added/removed at the step). Secondly, the user can ask the simulator to find all bindings for a given enabled transition – or he can specify a partial binding and ask the simulator to finish it, if possible. In an automatic simulation the simulator chooses among the enabled occurrence elements by means of a random number generator. It is possible to specify how large each step should be: It may contain a single occurrence element or as

[97] We do not allow code segments to create or delete CPN objects (but the attributes can be changed).

many as possible (and between these two extremes there is a continuum of other possibilities).

It is possible to vary the amount of graphical feedback provided by the CPN simulator. In the most detailed mode the user sees the enabled transitions, the occurring transitions, the tokens which are being moved and the current markings. Each of these feedback mechanisms is, however, controlled by one or more options, and thus they can be fully or partially omitted. In this way it is possible to speed up the simulation. As an extreme a special **super-automatic** mode has been provided. In this mode there is no user interaction (for the selection of bindings) and there is no feedback during the simulation (on the CPN diagram) – and this means that the simulation runs much faster than usual, because the simulation is performed by a SML program alone (while an ordinary simulation is performed by a SML program and a C program, with a heavy intercommunication).[98] At the end of a super-automatic simulation it is possible to inspect the effect of the simulation. This can be done either by means of the usual page windows (in which the marking is updated when the super-automatic simulation finishes) or by means of files manipulated by code segments. Finally, the code segments may, as described above, use the graphic routines of the CPN tools to create a visual representation of the simulation results – and this can be inspected while a super-automatic simulation is going on.

The user can, at any time during a simulation, change between manual, automatic and super-automatic simulation (and there are many other possibilities in between these three extremes).[99] It is usual to apply the more manual simulation modes early in a project (e.g. when a design is being created and investigated) while the more automatic modes are used in the later phases (e.g. when the design is being validated). There are no restrictions on the way in which the different simulation options can be mixed and this means that each of them can be chosen totally independently of the others (as an example manual/automatic/semi-automatic simulation can be with/without code (and with/without time, see section 7.1)).

There are many other facilities in the CPN simulator: An operation that proposes a step (which can be inspected and modified by the user before it is executed). Operations to return to the initial marking of the CPN diagram and to change the current marking of an arbitrary place instance (this means that it often is possible to continue a simulation in the case where a minor modelling error is encountered). Operations to save and load system states. Operations to activate/deactivate a large number of warning and stop options (i.e. different criteria under which a manual simulation issues a warning while an automatic simulation stops).[100] An operation to determine the order in which the different occurrence elements in a step is executed. Moreover, the earlier comments about different skill levels and a consistent and self-explanatory user interface also apply to the CPN simulator.

---

[98] Super-automatic simulation is not available in the released version of the CPN simulator, but a prototype version has been used in several projects (e.g. the one described in section 6.1). One of the next versions of the CPN simulator will contain a super-automatic mode which is fully integrated with the rest of the simulator.

[99] In the current version of the CPN simulator it is, during a simulation, not possible to change to/from super-automatic mode. This will, however, be possible in one of the next versions.

[100] The load/save operations and the warning/stop options are not yet implemented.

Finally, it should be mentioned that many modellers use simulation during the construction of CPN diagrams – in a similar way as a programmer tests selected parts of the program which he is writing. It is thus very important that it is reasonably fast to shift between the editor and the simulator (and that it is possible to simulate selected parts of a large model).

6. Applications of CP-nets

This chapter describes a number of projects which have used hierarchical CP-nets and the CPN tools. All the described projects have worked with reasonably large models and this have been done in an industrial environment – where parameters such as turn-around time and use of man-hours have been important.

6.1 Communication protocol

This project was carried out in cooperation with a large telecommunications company and it involved the modelling and simulation of selected parts of an existing ISDN protocol for digital telephone exchanges.[101] The modelling started from an SDL diagram, and it was straightforward to make a manual translation of the SDL diagram to a hierarchical CP-net.[102] The translation and simulation of the basic part of the protocol was finished in 16 days by a single modeller (which had large experience with the CPN tools, but no prior knowledge of communication protocols). The model was presented to engineers at the participating company. This was done by making a manual simulation of selected occurrence sequences – and by a super-automatic simulation, where code segments were used to update a page containing a visual representation of the travelling messages and the status of the user sites.[103] According to the engineers, who all had large experience with telephone systems, the CPN diagram provided the most detailed behavioural model which they had seen for this kind of system.

Later the modelling of a hold-feature was included in the CPN diagram. This was done in a single day, and it tuned out that it could be done by adding two extra pages, and making a simple modification of the existing pages (a colour set was changed from a triple to quadruple). In SDL the inclusion of the hold-feature made it necessary to duplicate the entire model, i.e. include many new pages – and so did three other features (which were not modelled in the project, but could have been handled in a similar way). Obviously, this makes it easier to maintain the CPN diagram (because it is sufficient to make modifications to one page instead of five).

[101] ISDN stands for Integrated Services Digital Network. The protocol is a BRI protocol (Basic Rate Interface) and it is the network layer which has been modelled.

[102] SDL is one of the standard graphical specification languages used by telecommunications companies. For information about SDL and how it can be translated into high-level Petri nets, see [13], [41] and [53].

[103] The simulation traces a call from the originating user to the terminating user, and to do this it was necessary to include a page which models the underlying protocol layers.

Fig. 14 shows the page hierarchy for the CPN diagram.[104] The subpages of UserTop#2 describe the actions of the user part while the subpages of NetTop#19 describe the actions of the network part. Most of these pages have a supernode which is called Ui (or Ni) and this indicates that the page describes the activity which can happen when the user part is in state Ui (the network part is in state Ni). The bracket in front of the pages U_PROG#41...U_REL_CO#40 indicates that they are subpages of all the pages in Null#3...Release_#17. The five pages describe activities which are carried out in the same way in all user states. If one of these activities is to be changed it is sufficient to modify one page of the CPN diagram (while it in the SDL diagram would be necessary to modify a large number of pages). The hold-feature is modelled by U_HOLD#45 and N_HOLD#44, while ROUTING#24 models the underlying protocol layers.

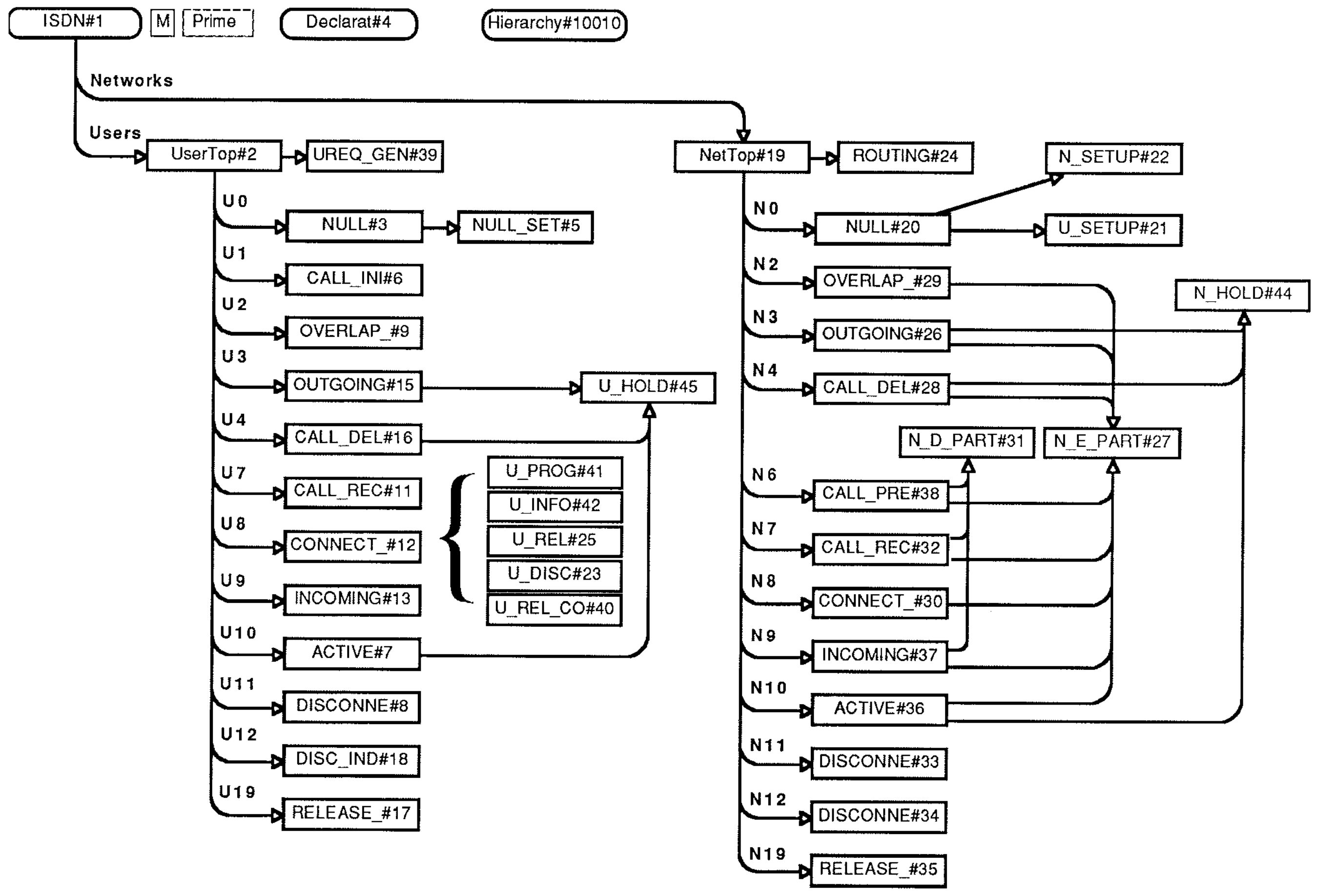


Figure 14. Page hierarchy for the ISDN protocol

A typical representative of the Ui/Ni pages is shown in Fig. 15.[105] It shows that, in the state U8, there are six different possibilities. When there is an internal user request the

[104] Names are truncated to the first eight characters, unless one of these is a format character (such as space, TAB, RETURN, etc). This convention keeps the feedback readable (also in diagrams with very long text strings). One of the next versions of the CPN tools will have a set of name options allowing the user to specify how names are truncated.

[105] The vertical lines and triangular figures inside the transitions are carried over from the SDL diagram, where they have a formal meaning. In the CPN diagram they have no formal meaning but they are retained, because they make the diagram more accessible for people who have experience with SDL.

first transition can occur. It creates a message to the network and the new user state becomes U11. When there is a message from the network one of the last five transitions can occur (the guards determine which one).[106] Two of the transitions create a message to the network, and the new user state becomes either U10, U12, U0, or U8. Three of the transitions are drawn with thick borders, this indicates that they are substitution transitions (having the pages U_DISC#23, U_REL#25 and U_REL_CO#40 as subpages). It should also be noticed that a global fusion set is used to glue all the U0-places together (and analogously for all the other 23 kinds of Ui/Ni-places).[107]

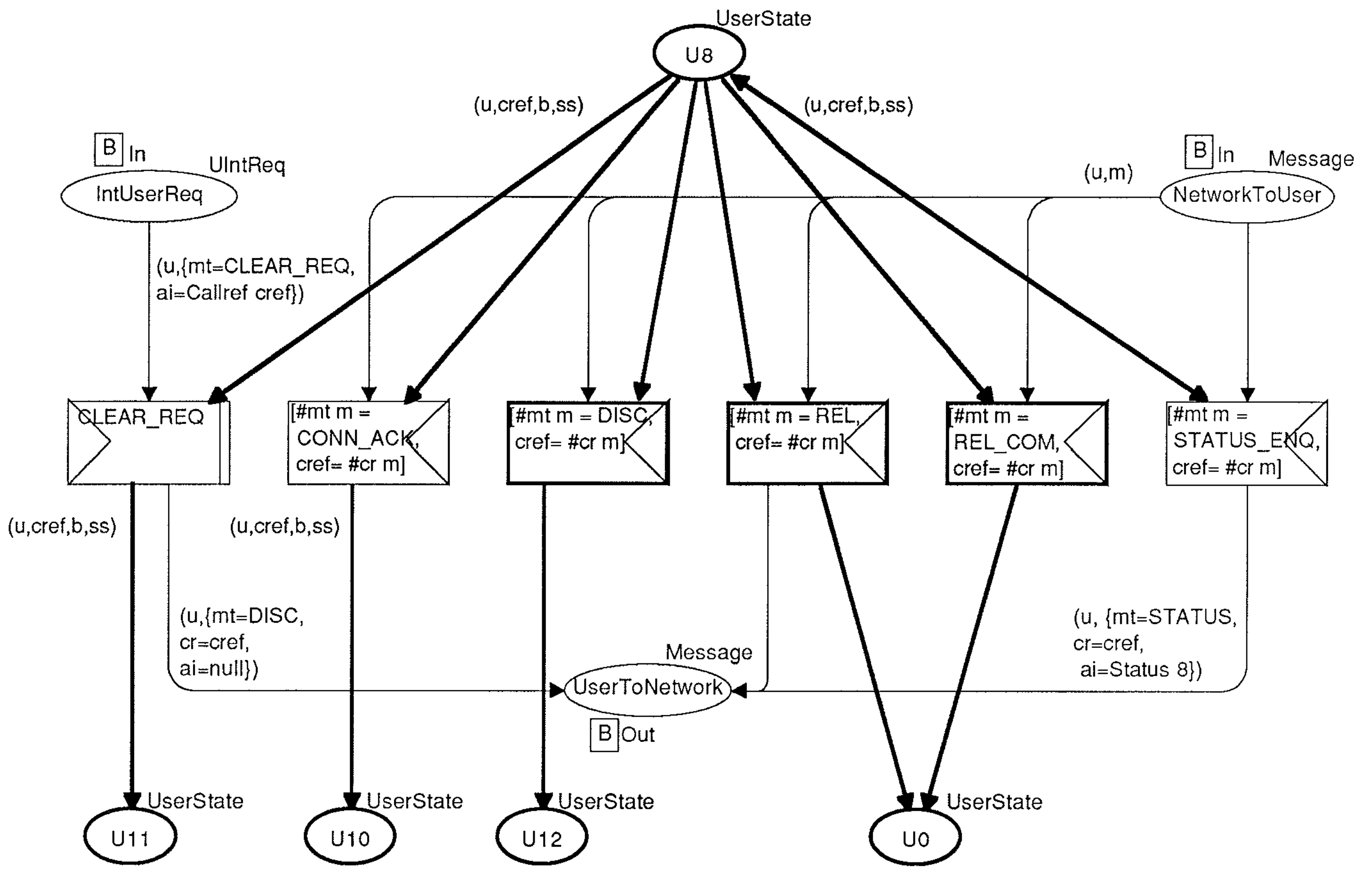


Figure 15. A typical page in the ISDN protocol (CONNECT_#12)

A typical representative of a transition is shown in Fig. 16 – together with the declarations of the appropriate colour sets. The transition is enabled when the user is in state U8 and there is a message with STATUS_ENQ as message type (on NetworkToUser). When the transition occurs the user remains in U8 and a message is created (on UserToNetwork). The new message has the same user and the same CallRef as the received message, it has STATUS as message type and Status 8 as data.

[106] To improve the readability the modeller has made some of the arc expressions invisible. All output arcs of NetworkToUser have identical arc expressions – and only one of these is visible.

[107] To improve the readability the modeller has made all the fusion regions invisible.

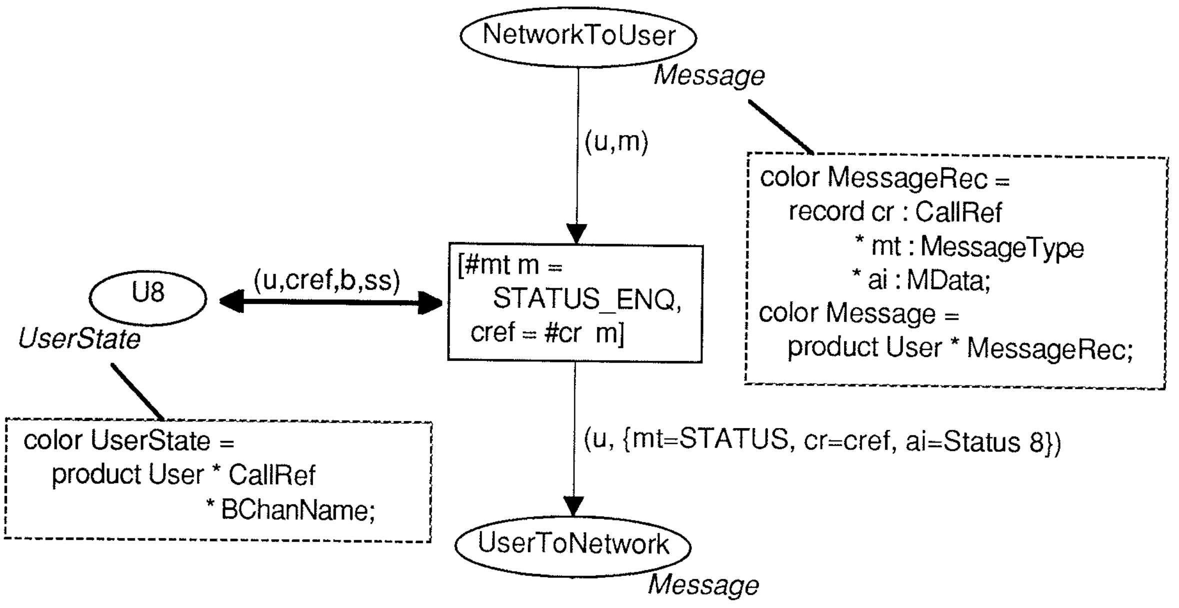


Figure 16. A typical transition in the ISDN protocol (rightmost on CONNECT_#12)

6.2 Hardware chip

This project was carried out in cooperation with a company which, among many other things, is a manufacturer of super-computers. The purpose of the project was to investigate whether the use of CP-nets is able to speed up the design and validation of VLSI chips (at the register transfer level). Below we sketch the main ideas behind the project and the most important conclusions. A much more detailed description of the project, the model, and the conclusions can be found in [54].

Let us first describe the existing design/validation strategy (without CP-nets): The chip designers specify a new chip by means of a set of block diagrams. Each diagram contains an interconnected set of blocks (activities), where each block has a specified input/output behaviour. A complex block may be specified in a separate block diagram, which is related to the block in a similar way as a substitution subpage in a CP-net is related to its supernode. When the designers have finished a new chip, the block diagrams are (by a manual process) translated into a simulation program written in a dialect of C. The simulation program is then executed on a large number of test data and the output is analysed to detect any malfunctions. The design/validation strategy described above has a number deficiencies – and we shall come back to these later (when we compare it to an alternative strategy which involves CP-nets).

Now let us describe the alternative design/validation strategy (involving CP-nets): The basic idea is to replace the manual translation (from the block diagrams into the C program) with an automatic translation into a CP-net. It is important to understand that it is *not* the intention to stop using block diagrams. The designers will still specify the designs by means of block diagrams, and they will during a simulation of the CP-net see the simulation results on the block diagrams. To support the new strategy three things are needed: The existing drawing tool for the block diagrams must be modified (to have a formal syntax and semantics). The set of block diagrams must be translated into CP-nets. Finally, it must be investigated whether the CPN simulator is powerful enough to handle complex VLSI designs.

The project only dealt with the last two issues (which were considered to be the most difficult). It was shown that the block diagrams could be translated into hierarchical CP-nets. This was done manually, but the translation process is rather straightforward and we see no problems in implementing an automatic translation. The obtained CP-net only contained 15 pages, but during a simulation there is nearly 150 page instances (due to the repeated use of substitution subpages representing adders and multipliers). The CP-net was simulated on the CPN simulator.[108]

Fig. 17 shows a subpage from which it can be seen that the VLSI chip has a pipelined design with six different stages. Each stage is modelled on a separate subpage and two of the more complex stages are shown in Fig. 18.[109] The eight transitions in the leftmost part of stage 1 are all substitution transitions (and they have the same subpage). In stage 2 the four transitions SUM1L, SUM1R, SUM2L and SUM2R represent registers. These registers establish the border to stage 3, and the transitions can only occur when they receive a clock pulse from stage 3 (via the two c-transitions in the rightmost part of stage 2). All the remaining transitions in stage 2 are substitution transitions (OR3 and OR4 denote or-gates while "+" denotes 16 bit adders).

Now let us compare the new design/validation strategy with the old: First of all, it is easier to translate the block diagrams into a CP-net than it is to translate them into a C program (the latter takes often several man-months while the construction of the CP-net only took a few man-weeks). The translation is also more transparent – in the sense that it is much easier to recognize those parts of the CP-net which models a given block than it is to find the corresponding parts in the C program (each page in the CP-net has nearly the same graphical layout as the corresponding block diagram). This means that it is relatively easy to change the CP-net to reflect any changes in the design, while this (according to the chip manufacturer) often is rather difficult for the C program. As stated above we think that it will be easy to automate the translation.

Secondly, the new strategy (when it is fully implemented) allows the designer to make simulations during the design process. This means that the knowledge and understanding which is acquired during the simulation of the model can be used to improve the design itself (in a much more direct way than in the old strategy where the validation is performed after the design has been finished).

Thirdly, the validation techniques of the old strategy concentrates on the logic correctness (tested by an inspection of the output data from the C program) and very little concern is given to those design decisions which deal with timing issues (e.g. the division into stages and the clock rate).[110] Using CP-nets it is possible to validate both the logic correctness and the timing issues – inside the same basic model.[111]

---

[108] When maximal graphical feedback was used the simulation was slow (due to the many graphical objects which had to be updated in each step). However, when a more selective feedback was used, the speed became reasonable.

[109] It is our intention to give the reader an idea about the complexity of the model (without explaining it in any detail).

[110] This is surprising, because the timing issues are crucial for the correct behaviour and the effectiveness of the chip (too fast clocking means malfunctioning while too slow means loss of speed).

[111] The timing issues were not modelled in the project – but with the time extensions of the CPN simulator (described in section 7.1) this can easily be done.

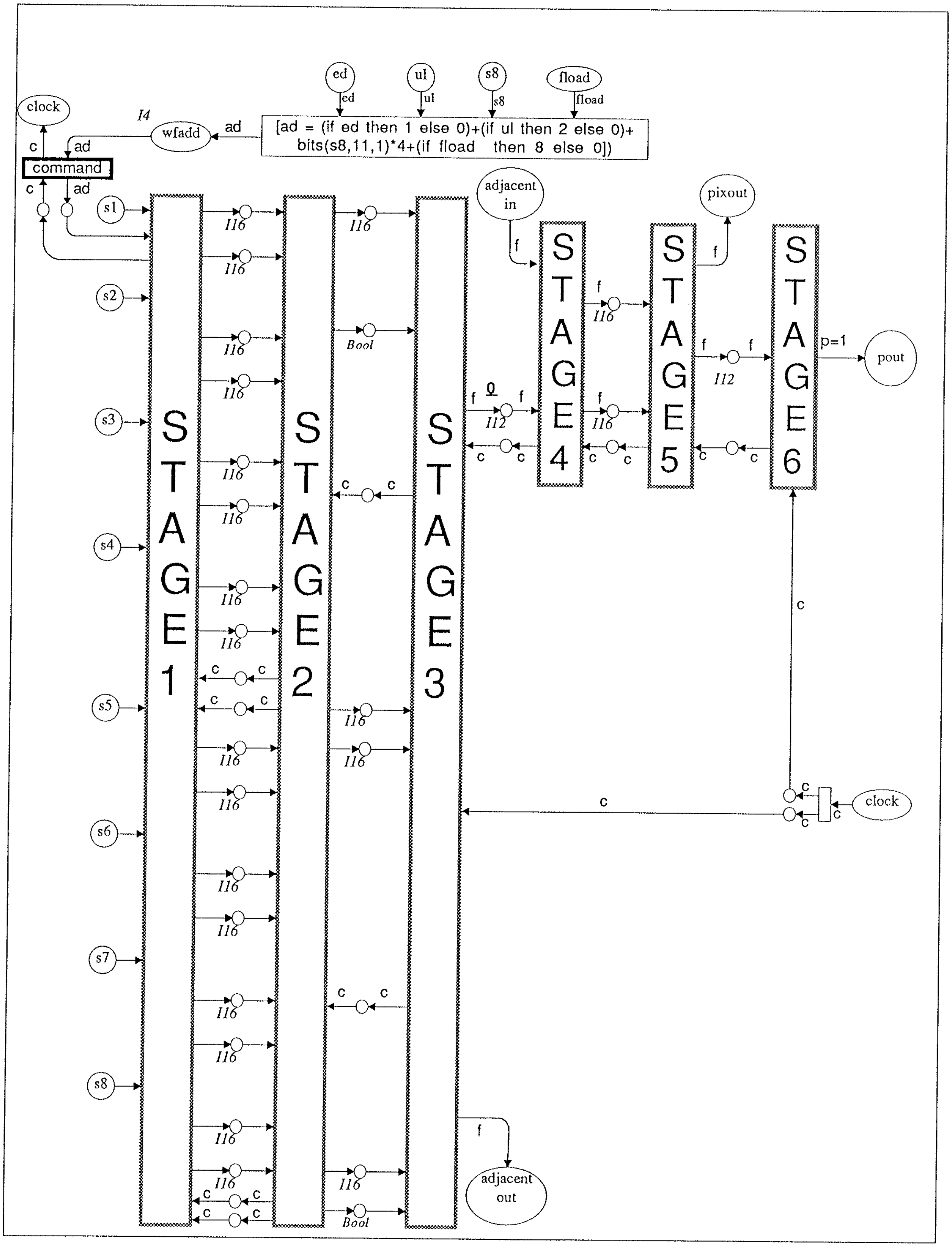


Figure 17. A page from the VLSI chip (showing the division into six pipe-lined stages)

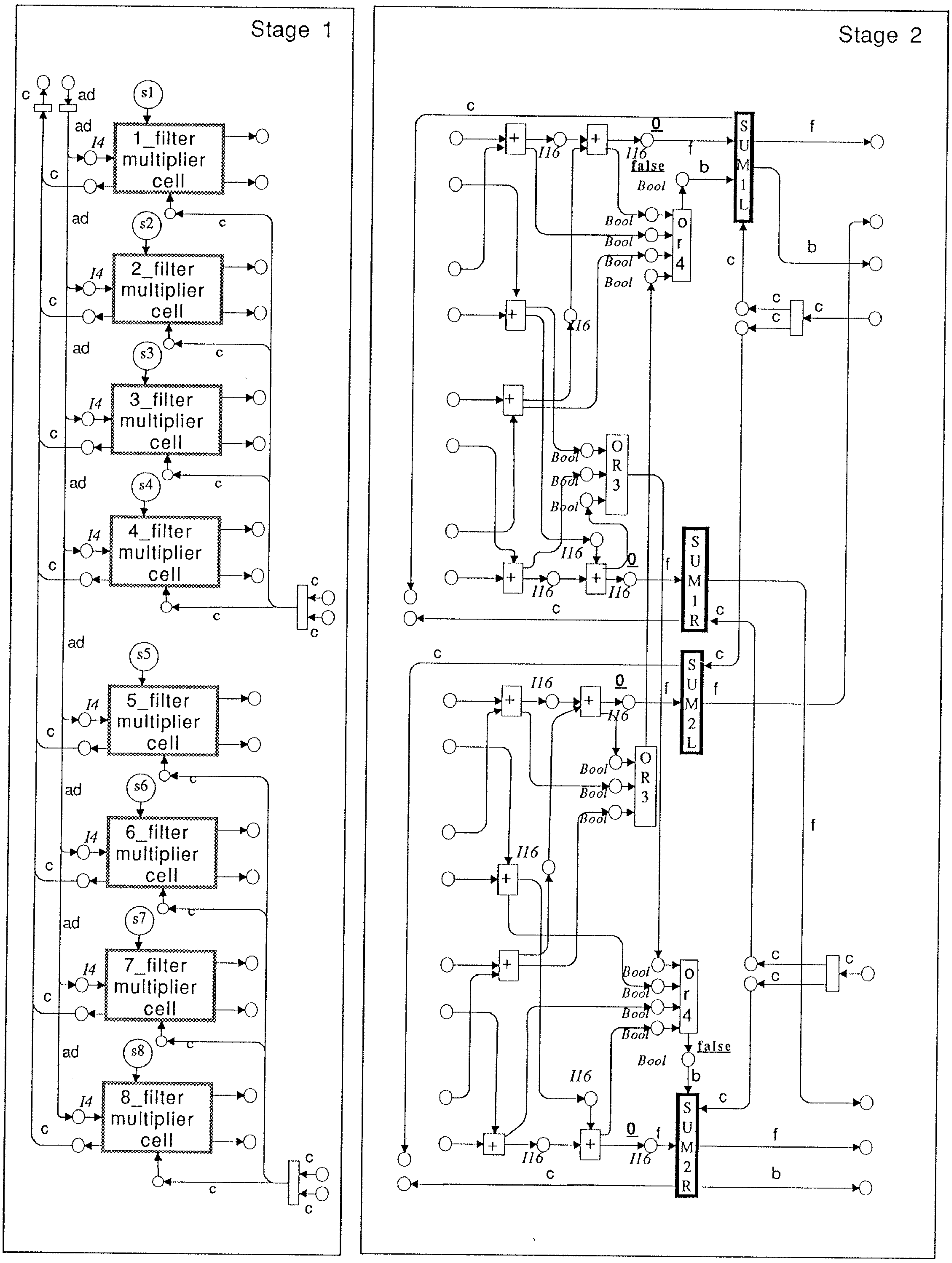


Figure 18. Two subpages of the page in Fig. 17 (modelling stage 1 and 2)

Finally, it was noticed that the execution of the C program was much faster than the CPN simulator – and that it with the latter would be impossible to make the usual amount of test runs (which typically include 10-20.000 sets of test data). It should, however, be noticed that the project was carried out immediately after the first version of the CPN simulator had been released – and that we (based on the experience with this and other large models) now have improved the speed of the CPN simulator with more than a factor 10. Moreover, super-automatic mode has been provided – and this means that we now are in a situation where it makes sense also to deal with large sets of test data.

6.3 Radar surveillance

This project was carried out in cooperation with Armstrong Aerospace Medical Research Laboratory (AAMRL) and it involved the modelling of a command post in the NORAD system.[112] The responsibility of the command post is to recommend different actions – based upon an assessment of the (rapidly changing) status of surveillance networks, defensive weapons and air traffic information. To do this the individual crew member communicates with many different types of equipment, other control posts and other members of the crew, and there is a complex set of detailed rules telling what he must do in the different types of situations. The entire system can be compared to a very complex communication protocol (although a large part of the communication is between human beings and not between computers). The proper design of command posts, including procedures, equipment and staffing, is an on-going problem – typical of the *Command and Control* area.

The purpose of the project was to get an executable model of the command post and use this model to get a better understanding of the command post – in order to improve its effectiveness and robustness. It was never the intention to use the CPN tools directly in the surveillance operations. A team of modellers working at AAMRL created a description of the command post, by means of SADT [43] (which in the United States is known as IDEF). This description was then augmented with more precise behavioural information, and the augmented model was automatically translated into a CP-net and simulated on the CPN simulator (for more details see below). The simulation gave (according to the people at AAMRL) an improved understanding of the command post, and they are now continuing the project modelling other parts of the NORAD system.[113] A much more detailed description of the project, the translation to CP-nets, and the model can be found in [55].

SADT diagrams are in many respects similar to CP-nets: Each SADT model consists of a set of pages,[114] and each of these contains a number of activities (playing a similar role as transitions in CP-nets). The activities are interconnected by arcs (these are called channels and there are three different kinds of them: representing physical flow,

[112] NORAD is the North American Radar Defense system.

[113] It is the plan to model a number of command posts – and run the submodel for each of these on a separate machine (using a separate copy of the CPN simulator). The submodels will then communicate via input/output statements in code segments (and this will be similar to the way in which the real control posts communicate with each other via electronic networks).

[114] In the SADT terminology each page is called a diagram. In this paper we shall, however, use the term diagram for the *set of pages* which constitutes a model.

control flow and availability of resources). SADT has no counterpart to places, but each channel has an attached date type (playing a similar role as the colour sets in CP-nets). Each SADT page (except for the top page) is a refinement of an activity of its parent page (and this works in a way which is totally analogous to transition substitution in CP-nets).

SADT diagrams are often ambiguous. As an example, a branching output channel may mean that the corresponding information/material sometimes is sent in one direction and sometimes in another. It may, however, also mean that the information/material is split in two parts, or that it is copied (and sent in both directions). Although some ambiguity may be tolerable as long as SADT is used to describe the structure of a system,[115] it is obvious that all ambiguity must be removed before the behaviour of a SADT model can be defined (i.e. before simulations can be made) – and this means that SADT must be augmented with better facilities to describe behaviour (e.g. to tell what a branching output channel means).

There are many different ways in which this can be done. One possibility (proposed in several SADT papers) is to attach a table to each activity. Each line in the table describes a possible set of acceptable input values and it specifies the corresponding set of output values. Another, and in our opinion much more attractive possibility, is to describe the input/output relation by a set of channel expressions and a guard – in exactly the same way as the behaviour of a CP-net transition is described by means of a set of arc expressions and a guard. Thus we introduce a new SADT dialect – called IDEF/CPN. In addition to the added channel expressions and guards there is a global declaration node (containing the declarations of types, functions, operations and IDEF variables). Finally, it is possible to use place fusion sets in a similar way as in CPN diagrams.

It is easy to translate an IDEF/CPN diagram into a behavioural equivalent CPN diagram, and this means that the CPN simulator can be used to investigate the behaviour of IDEF/CPN models. For the moment there is a separate IDEF/CPN tool which allows the user to construct, syntax check and modify IDEF/CPN diagrams. This tool works in a similar way as the CPN editor (and many parts of the two user interfaces are identical or very similar). The IDEF/CPN tool can create a file containing a textual representation of the IDEF/CPN diagram, and this file can then be read into the CPN simulator (where it is interpreted as a CPN diagram). The translation from IDEF/CPN to CPN diagrams is thus totally automatic. Later it is the plan to integrate a copy of the CPN simulator into the IDEF/CPN tool itself, and this will mean that the turn-around time will be faster (because it then is possible to edit and simulate in the same tool). Such an integration will also mean that the user will see the simulation results directly on the IDEF/CPN diagram. For the moment he sees the results on the CPN diagram – but this is not a big problem because the two diagrams look nearly identical (except that the former does not have places). Fig. 19 shows an IDEF/CPN page (from the radar surveillance system) and Fig. 20 shows the corresponding CPN page (as it is obtained by the automatic translation).

[115] The designers of SADT argue that it is fine to allow such ambiguities – because SADT should be used to "design" the information/material flow, without having to worry about the detailed behaviour (which in their opinion is an "implementation detail").

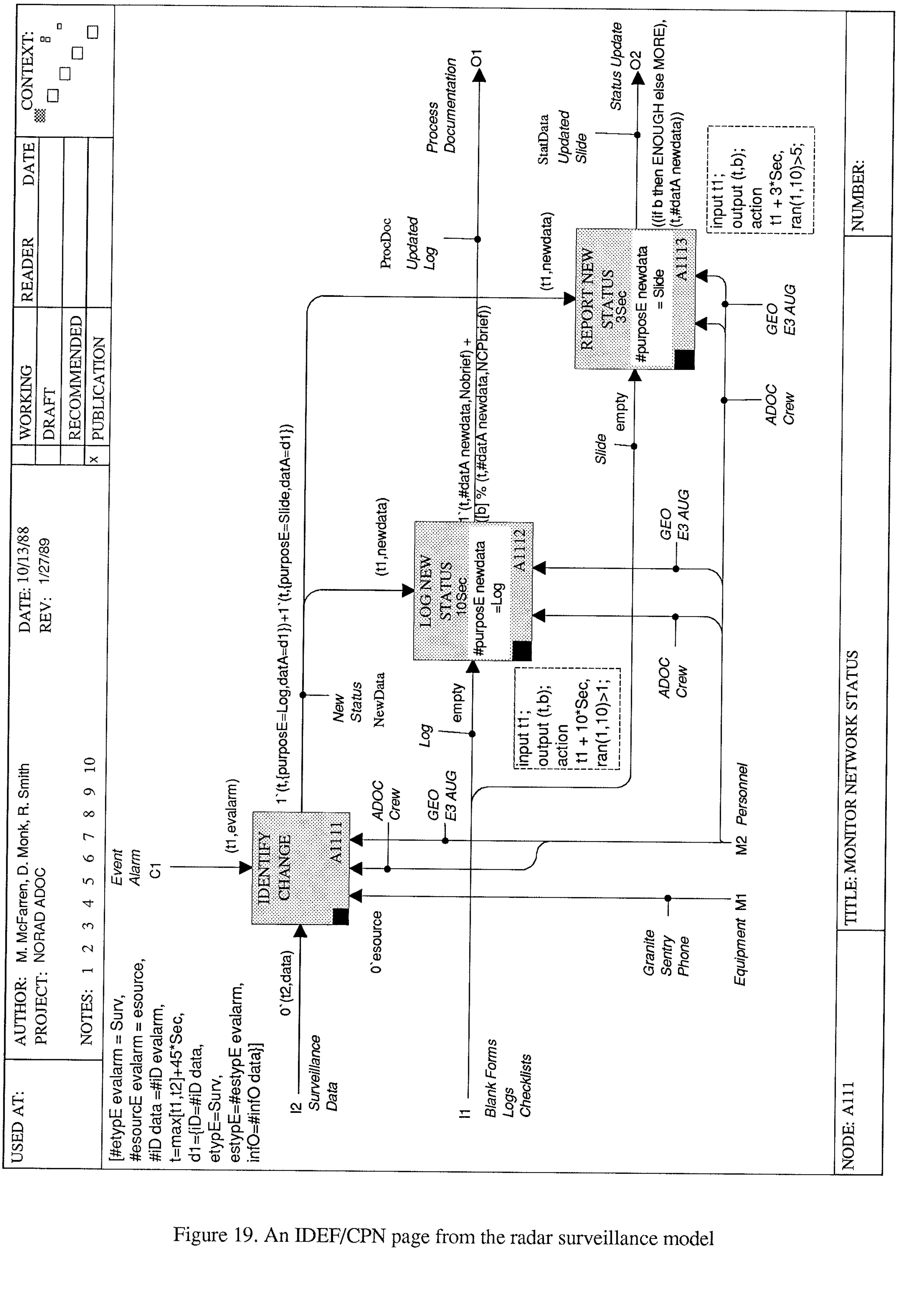

USED AT:
AUTHOR: M. McFarren, D. Monk, R. Smith
PROJECT: NORAD ADOC
NOTES: 1 2 3 4 5 6 7 8 9 10
DATE: 10/13/88
REV: 1/27/89
WORKING
DRAFT
RECOMMENDED
X PUBLICATION
READER
DATE
CONTEXT:
[#etypE evalarm = Surv, #esourcE evalarm = esource, #iD data =#iD evalarm, t=max[t1,t2]+45*Sec, d1={iD=#iD data, etypE=Surv, estypE=#estypE evalarm, infO=#infO data}]
I2 Surveillance Data
0`(t2,data)
0`esource
Event Alarm C1
(t1,evalarm)
IDENTIFY CHANGE
A1111
1`(t,{purposE=Log,dataA=d1})+1`(t,{purposE=Slide,dataA=d1})
New Status
NewData
(t1,newdata)
LOG NEW STATUS
10Sec
#purposE newdata =Log
A1112
input t1; output (t,b); action t1 + 10*Sec, ran(1,10)>1;
Log
empty
1`(t,#dataA newdata,Nobrief) + [b] % (t,#dataA newdata,NCPbrief))
ProcDoc
Updated Log
Process Documentation
O1
(t1,newdata)
REPORT NEW STATUS
3Sec
#purposE newdata = Slide
A1113
Slide
empty
((if b then ENOUGH else MORE), (t,#datA newdata))
input t1; output (t,b); action t1 + 3*Sec, ran(1,10)>5;
StatData
Updated Slide
Status Update
O2
I1 Blank Forms Logs Checklists
ADOC Crew
GEO E3 AUG
Granite Sentry Phone
Equipment M1
M2 Personnel
NODE: A111
TITLE: MONITOR NETWORK STATUS
NUMBER:

Figure 19. An IDEF/CPN page from the radar surveillance model

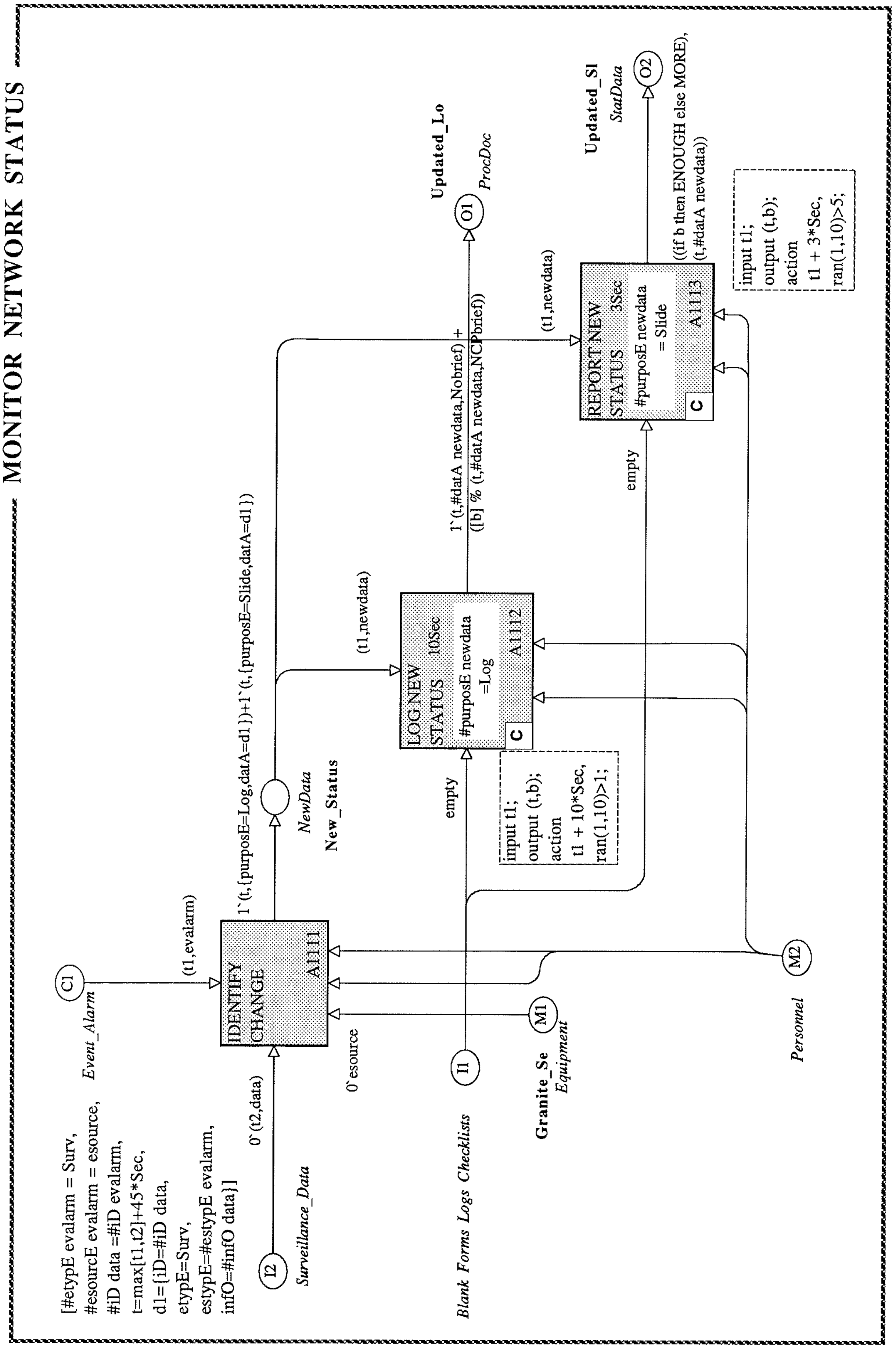


Figure 20. The CPN page obtained from the IDEF/CPN page in Fig. 19

6.4 Electronic funds transfer

This project was carried out in cooperation with two banks (Societé Générale and Marine Midland Bank of New York) and it involved the design and implementation of software to *control* the electronic transfer of money between banks. The speed of modern bank operations means that banks often make commitments which are based on money which they do not have (but expect to receive inside the next few minutes). What happens if these money are delayed – or never arrive? Two managers (at the involved banks) had an idea for a new control strategy – allowing the responsible staff to use computer support to control the electronic funds transfer.[116] The two managers concretized their idea in terms of a relatively small SADT diagram which was created by means of the IDEF/CPN tool (see section 6.3) and contained a rather informal description of the proposed algorithm. The IDEF/CPN diagram was translated to a CP-net and more accurate behavioural information was added, by a CPN modeller.[117] The translation was done in close cooperation with the two bank managers and they participated in the debugging (which also resulted in improvements of the original proposal).

During the project there were several different versions of the CPN model. The first of these was obtained more or less directly from the IDEF/CPN diagram, and it was rather crude (with simple arc expressions and very simple types). This model was primarily used to describe the data flow (while the actual data manipulations were ignored). Later the arc expressions were made more precise, a large number of complex data types were declared (and used as colour sets), complex CPN ML functions were declared (e.g. to search, sort and merge files), and finally most of the behavioural information were moved to code segments. In the final CPN model most transitions have arc expressions which consist of a single variable, and complex code segments determining the values of the output variables from the values of the input variables. It took 5 man-weeks to create the IDEF/CPN diagram, 1 man-week to get the first CPN diagram, and 16 man-weeks to develop this into the final CPN model.

In the first part of the project the graphical interface (in the editor/simulator) was of very large importance– and it was the graphical aspects of IDEF and CP-nets which made it possible for the bank managers to concretize their ideas. Later, however, it turned out that the graphical interface became less important while the output files produced by the simulation became more important – and thus the project started to use a stand-alone SML program (which was generated by the simulator in a similar way as the internal SML code needed for super-automatic simulation).

Now a simulation works with a number of input files (describing transfers which have already been made that day, and transfers which are registered but not yet executed). From these input files (which typically contains 15-50.000 records) a number of output files are produced (in 5-10 minutes) – and it is from these output files the

[116] Today the control of the transfer (i.e. the decisions about acceptance/rejection of the individual transactions) is made totally manual – although the transactions themselves are performed via special computer networks.

[117] The additional behavioural information could just as well have been added before the translation (i.e. by means of the IDEF/CPN tool instead of the CPN editor).

staff determine the transfer strategy to be used for the next 15-20 minutes (at which time a new set of simulation results is ready).

In this project the CPN tools (together with the IDEF/CPN tool) was used as a case tool. When the new strategy had been specified (by means of IDEF/CPN and the CPN editor) and validated (by means of the CPN simulator) the resulting SML code was automatically produced (by the CPN simulator).[118] The new control strategy, proposed by the two bank managers, seems to be working as expected – and it is for the moment being tested on historical bank data (using the SML code produced by the CPN simulator). When these tests are finished it will be determined whether the project will continue. If the project is continued the CPN model (and the IDEF/CPN model) will be extended to reflect additional aspects of funds transfer – and a graphical user interface will be added allowing the staff to interact with the model in a more natural way. The user interface will be created by letting the code segments use the graphical routines of the CPN tools. These routines are also available in the stand-alone ML environment, and thus it will still be possible to obtain the final SML code automatically from the CPN simulator (including the added graphical interface). A much more detailed description of the project, the models, and the conclusions can be found in [49].

6.5 Other application areas

CP-nets and other kinds of high-level Petri nets are used in many other application areas. For more information see e.g. [56] (flexible manufacturing systems); [28] and [52] (distributed algorithms); [58] (computer organization); [61] (data bases); [62] (office automation); [2] (computer architecture); [46] (human-machine interaction); [34] (semantics of programming languages); [15] and [29] (software development methods); [4], [8], [11], [14], [16], [19] and [24] (protocols).

From the applications reported in sections 6.1-6.4 (and some of the applications mentioned above) two interesting observations can be made: First of all, it is often adequate to use CP-net models in connection with different front-end languages (e.g. SDL, SADT and block diagrams). The reason may be that there already exist descriptions in these languages, or that the projects involve people who are familiar with some of the languages and thus prefer to use them (instead of learning a totally new formalism). It will also sometimes be sensible to make a tailored language (with a semantics based on CP-nets, but a syntax adopted to the problem area). This is for instance done by the designers of the Vista language [38], who have defined the semantics of their graphical specification language in terms of CP-nets.

Secondly, it is often the case that the graphical representation (which is very important in the early phases) later becomes less interesting. In this case the modellers may turn to super-automatic simulation and this yields a prototype implementation – or (for certain applications areas) even a final implementation. In this way the CPN tools are used as a case tool – and this will be even more attractive when it becomes possible to write code segments in different languages (such as C++, Pascal and Prolog).

[118] It was necessary to make a few manual operations to create the stand-alone SML code. These operations were trivial, and with the full support of super-automatic simulation they will disappear.

7. Future Plans for CP-nets

This chapter describes our plans for the further development of CP-nets. First we describe a number of extensions which is being made to the existing CPN tools (i.e. the CPN editor and the CPN simulator). Then we describe a number of new CPN tools which are being developed (e.g. to support occurrence graph analysis and invariant analysis). Finally we describe a book project which will provide the necessary introduction and documentation for CP-nets, their analysis methods, and selected examples of industrial applications.

7.1 Extensions of the CPN editor and CPN simulator

The CPN editor/simulator are being extended to handle timed CP-nets, which is an extension of ordinary CP-nets making it easy to describe systems which are time-driven. It will then be possible to use the same net model to analyse both the logic correctness and the time performance of a system. A timed CP-net has a global clock and the value of this is called the current *model time.*[119] The user can specify that certain colour sets are *with time* and this means that the corresponding tokens carry a time stamp (in addition to the ordinary colour information). Intuitively, the time stamp tells when the token is ready to be used (i.e. consumed by a transition). An occurrence element is said to be *colour enabled* if it satisfies the usual enabling criteria (defined by the arc expressions and the guard) and it is then said to have an *enabling time* which is the maximum of all the time stamps in the input tokens and the current model time. A colour enabled occurrence element is *time enabled* iff no other colour enabled occurrence elements have a smaller enabling time.[120] Only time enabled occurrence elements are allowed to occur (and this means that the transitions are executed in the order in which their tokens become ready). The occurrence rule is the same as for CP-nets without time – except that the time stamps of timed output tokens are determined by adding a delay to the current model time. The delays are specified by SML expressions and they may depend upon the colours of the input and output tokens (and via code segments also depend on reference variables and input files).[121] Each time a step has been executed the model time is advanced to match the minimal enabling time in the new system state[122] and this works very much like an event queue in a traditional simulation language. For more information about timed CP-net and the corresponding editor/simulation extensions see [37].

The CPN simulator is also being extended with a set of reporting facilities which will allow much easier visualization of the simulation results (e.g. during a super-automatic simulation). By means of code segments the user will be able to manipulate a

[119] The values of model time may be discrete (integers) or continuous (reals). In both cases each system state exists at a given model time – and the model time is monotonically increased throughout the simulation.

[120] A set of occurrence elements can be concurrently time enabled, but this requires that they all have the same enabling time.

[121] It is also possible to specify that different output tokens get different delays.

[122] The new model time may be identical to the old. This can e.g. happen when some (non conflicting) time enabled occurrence elements do not participate in the step or when some output tokens are created with a time stamp identical to the old model time (or without a time stamp).

large number of different charts (e.g. bar charts, function charts, pie charts and matrix charts). For each chart the code segments update an SML structure (with a predeclared type) while it is the CPN simulator which automatically updates the graphical representation of the chart (based on the value of the SML structure). The frequency by which the chart is updated is specified by the user (either in terms of the number of steps or in terms of model time). The charts are constructed by a special command in the CPN editor and they each consist of a number of auxiliary objects (which can be modified, e.g. resized, recoloured and repatterned, by the same editor operations as the other objects in the CPN diagram). For more information about the reporting facilities and their implementation see [37] and [39].

The implementation of timed CP-nets and the reporting facilities will be finished during the first half of 1991. Later we will also extend the CPN editor to allow the user to construct and modify CP-nets by means of a set of behaviour preserving transformation rules (for more information see [23]). We will also extend the CPN simulator to handle code segments written in other languages[123] and we will extend the CPN editor/simulator to handle the remaining hierarchy constructs and different extensions of CP-nets (e.g. capacities, inhibitor arcs and FIFO places). These projects have, however, lower priority than the creation of the occurrence graph and invariant tools described in section 7.2.

7.2 Additional CPN tools

A CPN tool will be created to support occurrence graph analysis. The tool will construct occurrence graphs for CP-nets (with/without equivalence classes) and it will also assist the user in the analysis of the constructed graphs. As described in section 4.2, a large number of system properties can be automatically determined from the occurrence graph (by an inspection of the individual markings and from the strongly connected components). There is, however, also a need to develop more complex search systems by which the user can perform an interactive inspection of a large occurrence graph. The CPN occurrence graph tool will be able to handle hierarchical CP-nets[124] and it will be tightly integrated into the existing CPN tools. It will e.g. be possible to ask the CPN simulator to execute an occurrence sequence which is found in the occurrence graph – or ask the occurrence graph analyser to search for markings which are identical to or larger than the current marking of the CPN simulator.

To keep the size of occurrence graphs manageable it will be necessary to create occurrence graphs for selected parts of a large model (and this will be done in exactly the same way as in the simulator – i.e. by defining prime pages and being able to ignore specified page instances). It will, moreover, be possible to simplify a model by means of *colour set restrictions*. The basic idea behind this concept is to be able to ignore parts of complex token colours – e.g. one or more components of a record type.[125] As an example, it may during the analysis of a communication protocol be adequate to ig-

[123] With the SML compiler running under Unix it is already today possible to use object code produced by other compilers.

[124] It is straightforward to extend the theory of occurrence graphs with equivalence classes to hierarchical CP-nets with transition substitution and place fusion (and this has already been done).

[125] This is analogous to (and inspired by) the concept of projections defined in [21].

nore the data contents of the messages. The restrictions are specified together with the colour set declarations, and this means that it is unnecessary to change the arc expressions or other net inscriptions. Colour set restrictions are also useful for simulation and it will in the future be possible to simulate a model with/without restrictions. For more information about colour set restrictions see [37].

Occurrence graphs can, for a given model, be constructed with/without time and with/without colour set restrictions. It makes, however, no sense to create occurrence graphs with code segments (at least not when these have side effects). The first version of the occurrence graph tool will be available during 1991. Later we will try to integrate our occurrence graph technique (building upon equivalence classes) with the techniques of other groups (see section 4.2).[126]

Analogously, a CPN tool will be created to support invariant analysis. The tool will calculate and check invariants for CP-nets and it will also assist the user when he applies the invariants to prove properties of the modelled system. The calculation of invariants are done in two steps: The first step is automatic and performs a reduction of the CP-net by a set of transformation rules which are proved to preserve the set of invariants.[127] The second step is interactive and it is performed directly on the CPN diagram (i.e. upon the graphical representation of the CP-net): The user proposes weight functions for a number of places. Typically he will define a small number of non-zero weight functions for places he is interested in (but also tell that certain places have zero weight). Then the invariant tool calculates those weight functions which can be uniquely determined from the weights proposed by the user. In this process the tool may also determine that some weights are inconsistent and high-light those transitions that create problems.[128] To calculate new weights and detect inconsistencies the invariant tool uses the reduced matrix obtained in the first step – but it shows the weights and the inconsistencies on the CPN diagram (i.e. in terms of the original CP-net). The user inspects the calculated weights and the high-lighted transitions – and based on this he may add new weights, modify existing weights, or change the behaviour of transitions (e.g. by modifying arc expressions and guards). The process continues, with a number of iterations, and at the end an invariant will be constructed (with some weights specified by the user and the remaining calculated by the invariant tool). The method described above may seem primitive and cumbersome – but this is *not* the case. On the contrary, it is often possible for the user to obtain useful invariants by defining a few weights.[129] It should, moreover, be remembered that the user often have a good idea about what the invariants will be (and thus e.g. knows that certain weights should be zero).

[126] In particular the technique described in [59] is interesting, because it seems to be orthogonal to our equivalence class technique (in the sense that the former exploits concurrency while the latter exploits symmetries).

[127] There are two different sets of reduction rules. One of them preserves place invariants while the other preserves transition invariants.

[128] To have an invariant each transition must be neutral, in the sense that the input tokens balance the output tokens (when the weights are taken into account).

[129] Each of the five place invariants PI_X from section 4.3 can be determined by specifying the weight of the single place X (and telling that some other places have weight zero).

To check a proposed invariant is even simpler: The user specifies all the weights and the invariant tool checks their consistency. When a set of invariants have been found they can be used to prove system properties, and this is also supported by the tool: As an example, the user may specify the marking of some places. Then the invariant tool calculates upper and lower bounds for other places (by means of the invariants) and in this process the tool may also determine that the specified set of place markings is inconsistent (i.e. impossible in all reachable markings).[130] The invariant tool will be able to handle hierarchical CP-nets[131] and it will, as described above, be tightly integrated into the existing CPN tools. The first version of the invariant tool is planned to be available during 1992. It is, however, obvious that this, among other things, will depend upon the priority given to the improvement of the new occurrence graph tool (and other extensions of existing CPN tools).

Finally we want to develop CPN tools to support reduction methods and the analysis of special subclasses of CP-net – e.g. as described in [9], [12] and [25]. Such tools have, however, lower priority than those described above.

7.3 CPN book

It is our plan to develop a coherent course material for those who want to study the theoretical and practical aspects of CP-nets. This material will be published as a three volume book in EATCS Monographs on Theoretical Computer Science. The book will contain the formal definition of CP-nets and the mathematical theory behind their analysis methods. It is, however, the intention to write the material in such a way that it also becomes attractive to people who are more interested in applications than the underlying mathematics. This means that a large part of the book will be written in a way which is closer to an engineering text book (or a users manual) than it is to a typical textbook in theoretical computer science.

The first volume of the book will introduce and define the net model (i.e. hierarchical CP-nets) and the basic concepts (e.g. the different behavioural properties such as dead-locks, fairness and home markings). It will in detail present a number of small examples and have brief overviews of some industrial applications. It will also contain a description of the CPN editor and the CPN simulator. Most of the material in this volume will be application oriented. The purpose of the volume is to teach the readers how to construct CPN models and how to analyse these by means of simulation.

The second volume will describe the theory behind the formal analysis methods – in particular occurrence graphs with equivalence classes, place/transition invariants and reductions. It will also describe how these analysis methods can be supported by CPN tools, and illustrate this by means of a number of examples. Part of this volume will be rather theoretical while other parts will be application oriented. The purpose of the volume is to teach the readers how to use the formal analysis methods (and this will not necessarily require a deep understanding of the underlying mathematical theory - although such knowledge of course will be a help).

---

130 Performed in this way, the non dead-lock proof in section 4.3 becomes much easier, faster and more reliable.

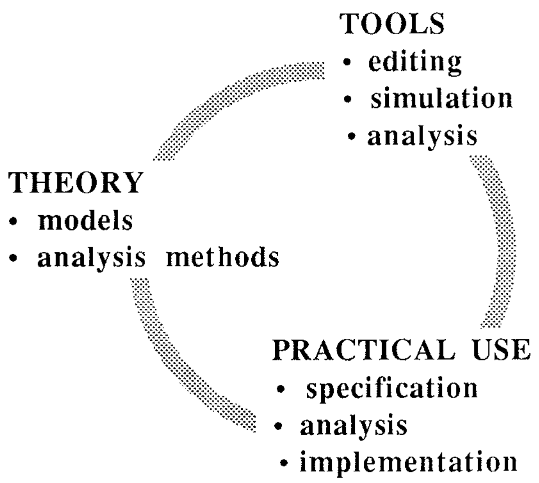
131 It is straightforward to extend the theory of invariants to hierarchical CP-nets with transition substitution and place fusion (and this has already been done).

The third volume will contain a detailed description of approximately ten different industrial applications. The purpose is to document the most important ideas and experiences from the projects – in a way which is useful for people who do not yet have personal experiences with the construction and analysis of large CPN diagrams. Another purpose is, of course, to document the feasibility of using CP-nets and the CPN tools for such projects.

For the moment approximately 400 pages have been written. Volume 1 will be available at the end of 1991 and we hope that volume 2 will be available during 1992/93. This depends, among other things, upon the speed by which the additional CPN tools are implemented.

8. Conclusions

This paper has presented the theory behind CP-nets, the supporting CPN tools and some of the practical experiences with them. In our opinion it is extremely important to develop these three research ares simultaneously. The three areas influence each other and none of them can be adequately developed without the other two. As an example we think it would have been totally impossible to develop the hierarchy concepts of CP-nets without simultaneously having a solid background in the theory of CP-nets, a good idea about a tool to support the hierarchy concepts and a thorough knowledge of the typical application areas.



Acknowledgments

Many different persons have contributed to the development of CP-nets and the CPN tools. Below some of the most important contributions are listed:

- CP-nets were derived from Predicate/Transition Nets which were developed by *Hartmann Genrich & Kurt Lautenbach.*
- The first version of occurrence graphs with equivalence classes was developed together with *Peter Huber, Arne Møller Jensen & Leif Obel Jepsen.*
- Many students and colleagues – in particular at *Aarhus University* – have influenced the development of CP-nets.
- *Grzegorz Rozenberg* has been a great support and inspiration for my book project (and for many other of my Petri net activities).
- The hierarchy constructs and the basic structure of the CPN tools were developed together with *Peter Huber & Robert M. Shapiro.*
- The idea to use an extension of Standard ML for the inscriptions of CP-nets is due to *Jawahar Malhotra.*
- The idea of a super-automatic simulation mode and a stand-alone SML program is due to *Valerio Pinci & Robert M. Shapiro..*
- The user interface of the CPN tools was designed together with *Søren Christensen* and it was implemented by *Ole Bach Andersen.* Valuable critique and suggestions were provided by *Michel Beaudouin-Lafon.*
- The abstract date base of the CPN tools was designed by *Peter Huber* and it was implemented by *Vino Gupta.*
- The ML functions to calculate enabling and bindings were designed and implemented together with *Søren Christensen & Peter Huber.*
- IDEF/CPN was designed and implemented by *Robert M. Shapiro.*
- The Unix + X-Windows version of the CPN tools was implemented by *Jane Eisenstein, Ivan Hajadi & Greg Alonso.*
- The reporting facilities was implemented by *Alain Karsenty.*
- Some of the first hierarchical CPN models were made by *Vino Gupta, Peter Huber, Robert Mameli, Valerio Pinci & Robert M. Shapiro.*
- *Hartmann Genrich* has participated in many parts of the development of the CPN tools.
- *Bob Seltzer* has been a continuous supporter of the CPN tool project (and the daily chats with him have been of great importance for the mood of the project group).
- *Meta Software* has provided the financial support for the CPN tool project. So far more than 25 man years have been used. The project is also supported by the Danish National Science Research Council, the Human Engineering Division of the Armstrong Aerospace Medical Research Laboratory at Wright-Patterson Air Force Base, and the Basic Research Group of the Technical Panel C3 of the US Department of Defense Joint Directors of Laboratories at the Navel Ocean Systems Center.

Finally I thank the anonymous referees for their contributions to this paper.

References

[1] K. Albert, K. Jensen and R.M. Shapiro: **Design/CPN. A tool package supporting the use of Coloured Petri Nets.** Petri Net Newsletter 32 (April 1989), 22-36.

[2] J.L. Baer: **Modelling architectural features with Petri nets.** In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986 Part II, Lecture Notes in Computer Science vol. 255, Springer-Verlag 1987, 258-277.

[3] E. Best: **Structure theory of Petri nets: the free choice hiatus.** In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I, Lecture Notes in Computer Science vol. 254, Springer-Verlag 1987, 168-205.

[4] J. Billington, G. Wheeler and M. Wilbur-Ham: **Protean: a high-level Petri net tool for the specification and verification of communication protocols.** IEEE Transactions on.

Software Engineering, Special Issue on Tools for Computer Communication Systems, SE-14(3), 1988, 301-316.

[5] W. Brauer (ed.): **Net theory and applications**. Proceedings of the Advanced Course on General Net Theory of Processes and Systems, Hamburg 1979, Lecture Notes in Computer Science vol. 84, Springer-Verlag 1980, 213-223.

[6] W. Brauer, W. Reisig and G. Rozenberg (eds.): **Petri nets: Central models and their properties**. Advances in Petri Nets 1986 Part I, Lecture Notes in Computer Science vol. 254, Springer-Verlag 1987

[7] W. Brauer, W. Reisig and G. Rozenberg (eds.): **Petri nets: Applications and relationships to other models of concurrency**. Advances in Petri Nets 1986 Part II, Lecture Notes in Computer Science vol. 255, Springer-Verlag 1987

[8] G. Chehaibar: **Validation of phase-executed protocols modelled with coloured Petri nets.** Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris 1990, 84-103.

[9] G. Chiola, C. Dutheillet, G. Franceschinis and S. Haddad: **On well-formed coloured nets and their symbolic reachability graph.** Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris 1990, 387-411.

[10] C. Choppy and C. Johnen: **Petrireve: proving Petri net properties with rewriting systems.** J.P. Jouannaud (ed.): Rewriting Techniques and Applications, Lecture Notes in Computer Science vol. 202, Springer-Verlag 1985, 271-286.

[11] B. Cousin et. al.: **Validation of a protocol managing a multi-token ring architecture.** Proceedings of the 9th European Workshop on Applications and Theory of Petri Nets, Vol. II, Venice 1988.

[12] J.M. Couvreur: **The general computation of flows for coloured Petri nets.** Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris 1990, 204-223.

[13] F. De Cindio, G. Lanzarone and A. Torgano: **A Petri net model of SDL.** Proceedings of the 5th European Workshop on Applications and Theory of Petri Nets, Aarhus 1984, 272-289.

[14] M. Diaz: **Petri net based models in the specification and verification of protocols.** In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986 Part II, Lecture Notes in Computer Science vol. 255, Springer-Verlag 1987, 135-170.

[15] R. Di Giovanni: **Putting Petri nets into use: the Columbus programme**. Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris 1990, 123-138.

[16] P. Estraillier and C. Girault: **Petri nets specification of virtual ring protocols.** In: A. Pagnoni and G. Rozenberg (eds.): Applications and Theory of Petri Nets, Informatik-Fachberichte vol. 66, Springer-Verlag 1983, 74-85.

[17] F. Feldbrugge: **Petri net tool overview 1989.** In: G. Rozenberg (ed.): Advances in Petri Nets 1989. Lecture Notes in Computer Science vol. 424, Springer-Verlag 1990, 151-178.

[18] A. Finkel: **A minimal coverability graph for Petri nets.** Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris 1990, 1-21.

[19] G. Florin, C. Kaiser, S. Natkin: **Petri net models of a distributed election protocol on undirectional ring**. Proceedings of the 10th International Conference on Application and Theory of Petri Nets, Bonn 1989, 154-173.

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

[21] H.J. Genrich: **Projections of C/E-systems.** In: G. Rozenberg (ed.): Advances in Petri Nets 1985. Lecture Notes in Computer Science vol. 222, Springer-Verlag 1986, 224-232.

[22] H.J. Genrich: **Predicate/Transition nets.** In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I, Lecture Notes in Computer Science vol. 254, Springer-Verlag 1987, 207-247.

[23] H.J. Genrich: **Equivalence transformations of PrT-nets.** In: G. Rozenberg (ed.): Advances in Petri Nets 1989, Lecture Notes in Computer Science, vol. 424, Springer-Verlag 1990, 179-208.

[24] C. Girault, C. Chatelain and S. Haddad: **Specification and properties of a cache coherence protocol model.** In: G. Rozenberg (ed.): Advances in Petri Nets 1987, Lecture Notes in Computer Science, vol. 266, Springer-Verlag 1987, 1-20.

[25] S. Haddad: **A reduction theory for coloured nets.** In: G. Rozenberg (ed.): Advances in Petri Nets 1989, Lecture Notes in Computer Science, vol. 424, Springer-Verlag 1990, 209-235.

[26] R. Harper: **Introduction to Standard ML**. University of Edinburgh, Department of Computer Science, The King's Buildings, Edinburgh EH9 3JZ, Technical Report ECS-LFCS-86-14, 1986.

[27] R. Harper, D. MacQueen and R. Milner: **Standard ML**. University of Edinburgh, Department of Computer Science, The King's Buildings, Edinburgh EH9 3JZ, Technical Report ECS-LFCS-86-2, 1986.

[28] G. Hartung: **Programming a closely coupled multiprocessor system with high level Petri nets.** In: G. Rozenberg (ed.): Advances in Petri Nets 1988, Lecture Notes in Computer Science vol. 340, Springer-Verlag 1988, 154-174.

[29] T. Hildebrand, H. Nieters, and N Trèves: **The suitability of net-based Graspin tools for monetics applications.** Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris 1990,139-160.

[30] P. Huber, A.M. Jensen, L.O. Jepsen and K. Jensen: **Reachability trees for high-level Petri nets**. Theoretical Computer Science 45 (1986), 261-292.

[31] P. Huber, K. Jensen and R.M. Shapiro: **Hierarchies in coloured Petri nets**. In: G. Rozenberg (ed.): Advances in Petri Nets 1990, Lecture Notes in Computer Science, Springer-Verlag.

[32] K. Jensen: **Coloured Petri nets and the invariant method**. Theoretical Computer Science 14 (1981), 317-336.

[33] K. Jensen: **High-level Petri nets**. In: A. Pagnoni and G. Rozenberg (eds.): Applications and Theory of Petri Nets, Informatik-Fachberichte vol. 66, Springer-Verlag 1983, 166-180.

[34] K. Jensen and E.M. Schmidt: **Pascal semantics by a combination of denotational semantics and high-level Petri nets.** In: G. Rozenberg (ed.): Advances in Petri Nets 1985. Lecture Notes in Computer Science vol. 222, Springer-Verlag 1986, 297-329.

[35] K. Jensen: **Coloured Petri nets.** In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I, Lecture Notes in Computer Science vol. 254, Springer-Verlag 1987, 248-299.

[36] K. Jensen et. al.: **Design/CPN: A tool supporting coloured Petri nets.** User's manual, vol 1-2. Meta Software Corporation, 150 Cambridge Park Drive, Cambridge MA 02140, USA, 1988.

[37] K. Jensen et. al.: **Design/CPN extensions.** Meta Software Corporation, 150 Cambridge Park Drive, Cambridge MA 02140, USA, 1990.

[38] E. de Jong and M.R. van Steen: **Vista: a specification language for parallel software design**. Proceedings of the 3rd International Workshop on Software Engineering and its Applications, Toulouse, 1990.

[39] A. Karsenty: **Interactive graphical reporting facilities for Design/CPN.** Master Thesis, University of Paris Sud, Computer Science Department, 1990.

[40] R.M. Karp and R.E. Miller: **Parallel program schemata**. Journal of Computer and System Sciences, vol. 3, 1969, 147-195.

[41] M. Lindqvist: **Translation of the specification language SDL into predicate/transition nets.** Licentiate's Thesis, Helsinki University of Technology, Digital Systems Laboratory, 1987.

[42] M. Lindqvist: **Parameterized reachability trees for predicate/transition nets.** Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris 1990, 22-42.

[43] D.A. Marca and C.L. McGowan: **SADT**. McGraw-Hill, New York, 1988.

[44] G. Memmi and J. Vautherin: **Analysing nets by the invariant method.** In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I, Lecture Notes in Computer Science vol. 254, Springer-Verlag 1987, 300-336.

[45] Y. Narahari: **On the invariants of coloured Petri nets.** In: G. Rozenberg (ed.): Advances in Petri Nets 1985. Lecture Notes in Computer Science vol. 222, Springer-Verlag 1986, 330-345.

[46] H. Oberquelle: **Human-machine interaction and role/function/action-nets.** In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986 Part II, Lecture Notes in Computer Science vol. 255, Springer-Verlag 1987, 171-190.

[47] **Petri nets and performance models.** Proceedings of the third international workshop, Kyoto Japan 1989, IEEE computer society press, order number 2001, ISBN 0-8186-20001-3.

[48] C.A. Petri: **Kommunikation mit automaten**. Schriften des IIM Nr. 2, Institut für Instrumentelle Mathematik, Bonn, 1962. *English translation:* Technical Report RADC-TR-65-377, Griffiss Air Force Bas, New York, Vol. 1, Suppl. 1, 1966.

[49] V.O. Pinci and R.M. Shapiro: **Development and implementation of a strategy for electronic funds transfer by means of hierarchical coloured Petri nets**. Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris 1990, 161-180.

[50] C. Reade: **Elements of functional programming.** Addison Wesly, International Computer Science Series, ISBN 0-201-12915-9, 1989.

[51] G. Rozenberg: **Behaviour of elementary net systems**. In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I, Lecture Notes in Computer Science vol. 254, Springer-Verlag 1987, 60-94.

[52] M. Rukoz and R. Sandoval.: **Specification and correctness of distributed algorithms by coloured Petri nets.** Proceedings of the 9th European Workshop on Applications and Theory of Petri Nets, Vol. II, Venice 1988.

[53] **Functional specification and description language SDL**. In: CCITT Yellow Book, Vol. VI, recommendations Z.101 - Z.104, CCITT, Geneva, 1981.

[54] R.M. Shapiro: **Validation of a VLSI chip using hierarchical coloured Petri nets**. Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris 1990, 224-243.

[55] R.M. Shapiro, V.O. Pinci and R. Mameli: **Modelling a NORAD command post using SADT and coloured Petri nets.** Proceedings of the IDEF Users Group, Washington DC, May 1990.

[56] M. Silva and R. Valette: **Petri nets and flexible manufacturing.** In: G. Rozenberg (ed.): Advances in Petri Nets 1989, Lecture Notes in Computer Science, vol. 424, Springer-Verlag 1990, 374-417.

[57] P.S. Thiagarajan: **Elementary net systems**. In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I, Lecture Notes in Computer Science vol. 254, Springer-Verlag 1987, 26-59.

[58] R. Valk: **Nets in computer organization.** In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986 Part II, Lecture Notes in Computer Science vol. 255, Springer-Verlag 1987, 218-233.

[59] A. Valmari: **Stubborn sets for reduced state space generation.** Proceedings of the 10th International Conference on Application and Theory of Petri Nets, Bonn 1989, Vol II.

[60] A. Valmari: **Compositional state space generation.** Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris 1990, 43-62.

[61] K. Voss: **Nets in data bases.** In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986 Part II, Lecture Notes in Computer Science vol. 255, Springer-Verlag 1987, 97-134.

[62] K. Voss: **Nets in office automation.** In: W. Brauer, W. Reisig and G. Rozenberg (eds.): Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986 Part II, Lecture Notes in Computer Science vol. 255, Springer-Verlag 1987, 234-257.

[63] Å. Wikström: **Functional programming using Standard ML.** Prentice Hall International Series in Computer Science, ISBN 0-13-331968-7, ISBN 0-13-331661-0 Pbk, 1987

[64] E. Yourdon: **Managing the system life cycle**. Yourdon Press, 1982.