An Introduction to the Theoretical Aspects of Coloured Petri Nets

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

Phone: +45 89 42 32 34
Telefax: +45 89 42 32 55
E-mail: kjensen@daimi.aau.dk

Abstract: This paper presents the basic theoretical aspects of Coloured Petri Nets (CP-nets or CPN). 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). The paper contains the formal definition of CP-nets and their basic concepts (e.g., the different dynamic properties such as liveness and fairness). The paper also contains a short introduction to the analysis methods, in particular occurrence graphs and place invariants.

The development of CP-nets has been driven by the desire to develop a modelling language – at the same time theoretically well-founded and versatile enough to be used in practice for systems of the size and complexity that we find in typical industrial projects. To achieve this, we have combined the strength of Petri nets with the strength of programming languages. Petri nets provide the primitives for the description of the synchronisation of concurrent processes, while programming languages provide the primitives for the definition of data types and the manipulation of their data values.

The paper does not assume that the reader has any prior knowledge of Petri nets – although such knowledge will, of course, be a help.

Keywords: Petri Nets, High-level Petri Nets, Coloured Petri Nets.

Table of Contents

This paper is also published in: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.): A Decade of Concurrency, Lecture Notes in Computer Science vol. 803, Springer-Verlag 1994, 230-272.

1 Informal Introduction to CP-nets

This sections contains an informal introduction to CP-nets. This is done by means of an example that models a small distributed data base system, cf. Fig. 1. The example is far too small to illustrate the typical practical use of CP-nets, but it is large enough to illustrate the theoretical definition of CP-nets, their basic concepts and their analysis methods. We shall use the data base system throughout this paper. As a curiosity, it can be mentioned that the annual International Petri Net Conference uses the net structure of the data base system as its logo.

The data base system has n different geographical sites ($n \geq 3$). Each site contains a copy of the entire data base and this copy is handled by a local data base manager. When a manager d_i makes an update to his own copy of the data base, he must send a message to all the other managers – to ensure consistency between the n copies of the data base. We are not interested in the content of the message, but only in the header information. Hence, we represent each message as a pair (s,r) where s identifies the sender and r identifies the receiver. This means that the data base manager d_i sends the following messages:

$$Mes(d_i) = \{(d_i,d_1), (d_i,d_2), \ldots, (d_i,d_{i-1}), (d_i,d_{i+1}), \ldots, (d_i,d_{n-1}), (d_i,d_n)\}.$$

In contrast to most specification languages, Petri nets are state and action oriented at the same time – providing an explicit description of both the states and the actions. This means that the modeller can determine freely whether – at a given moment of time – he wants to concentrate on states or on actions.

The states of a CP-net are represented by means of **places** (which are drawn as ellipses). In the data base system there are nine different places. Three of the places represent the three possible states of the data base managers: *Inactive*, *Waiting* and *Performing*. Four of the places represent the possible states of the messages: *Unused*, *Sent*, *Received* and *Acknowledged*. The two remaining places indicate whether an update is going on or not: *Active* and *Passive*. By convention we write the names of the places inside the ellipses. The names have no formal meaning – but they have large practical importance for the readability of a CP-net (just like the use of mnemonic names in traditional programming). A similar remark applies to the graphical appearance of the places, i.e., the line thickness, size, colour, font, position, etc. A good graphical representation corresponds to a good indentation strategy in a program – it has an immense importance for the readability of the net. A set of guidelines for the graphical representation of CP-nets can be found in Sect. 1.6 of [27].

Each place has an associated data **type** determining the kind of data which the place may contain (by convention the type information is written in italics, next to the place). The box at the top of Fig. 1 contains declarations. The first six lines specify the possible values of four different types: DBM, PR, MES and E. The declarations also, implicitly, specify the operations which can be performed on the values of the types. In Fig. 1, we specify the types by means of a language based on Standard ML, see [34], [35] and [37]. In this paper, we give only an informal explanation of the declarations. For more details, see Sects. 1.3 and 1.4 of [27].

For CP-nets, we use the terms: type, value, operation, expression, variable, binding and evaluation in exactly the same way as these concepts are used in functional programming languages. It is possible to specify the types in many other ways, e.g., by means of the kind of specifications which are used in abstract data types. In Sect. 3 we discuss the requirements demanded for the language in which declarations and arc expressions are written.

A state of a CP-net is called a **marking**. It consists of a number of **tokens** positioned on the individual places. Each token carries a data value which belongs to the type of the corresponding place. *Inactive*, *Performing* and *Waiting* have the type:

$$\text{DBM} = \{d_1, d_2, \ldots, d_n\}.$$

Intuitively, this means that each token (on one of these places) represents a data base manager. In the initial marking all data base managers are inactive, hence we have n tokens on *Inactive* while *Performing* and *Waiting* have none. In general, a place may contain two or more tokens with the same data value. This means that we have multi-sets of tokens, and not just sets. A marking of a CP-net

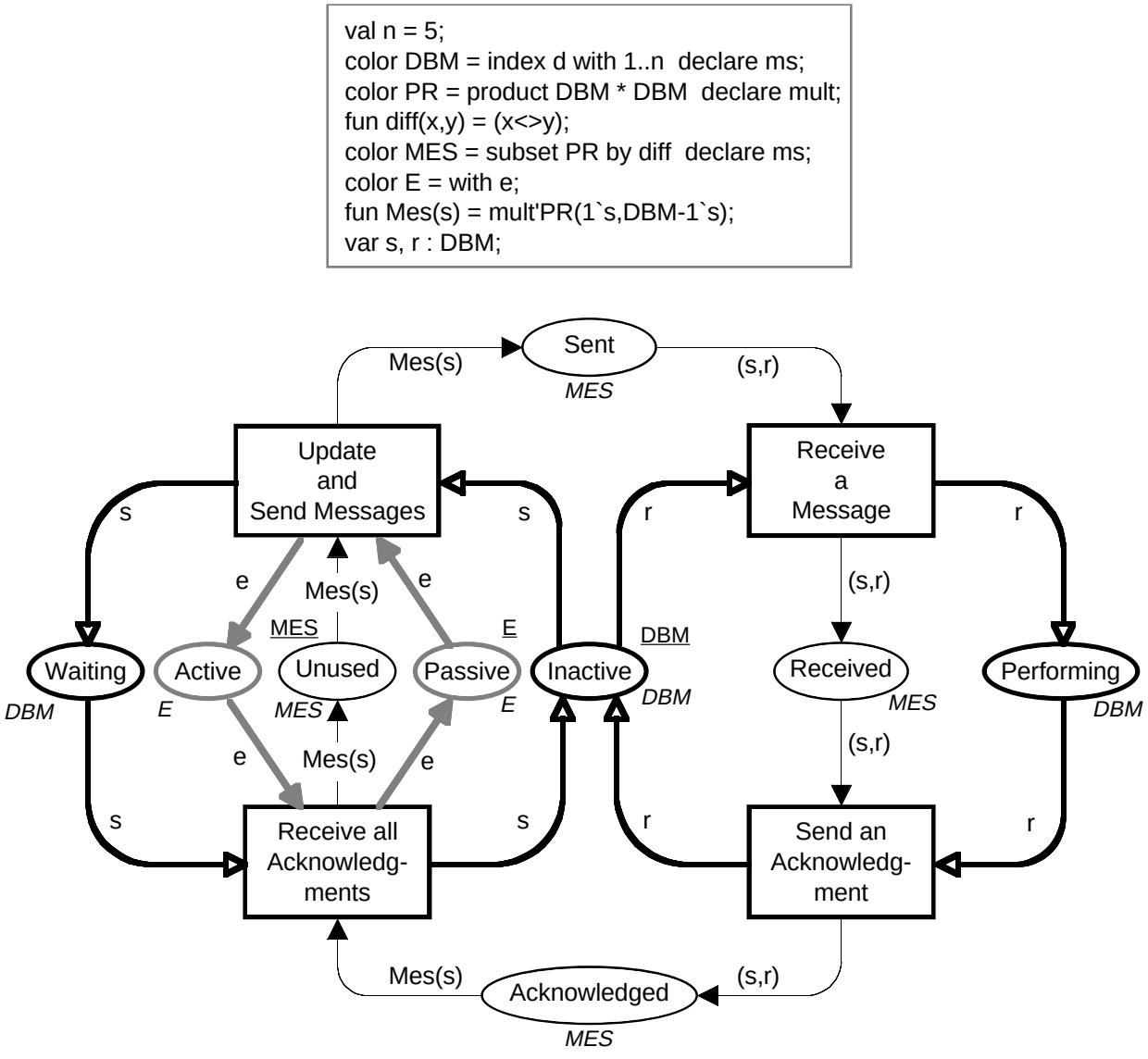

Fig. 1. CP-net describing a distributed data base

is a function which maps each place into a multi-set of tokens of the correct type. The initial marking M_0 is specified by means of the initialisation expressions (which by convention are written with an underline, next to the place). A missing initialisation expression implies that the initial marking of the corresponding place is empty, i.e., contains no tokens. For the places of type DBM we have the following initial marking:

$$M_0(\text{Inactive}) = \text{DBM}$$

$$M_0(\text{Performing}) = M_0(\text{Waiting}) = \emptyset$$

where Ø denotes the empty multi-set. By convention, we use DBM to denote the type, but we also use it to denote the set which contains all data values from the type and the multi-set which contains exactly one appearance of each data value from the type. Hence we have:

$$M_0(\text{Inactive}) = 1`d_1 + 1`d_2 + \ldots + 1`d_n$$

where the integer coefficients (in front of `) indicate that *Inactive* has one token for each data value d_i in the type DBM.

Unused, *Sent*, *Received* and *Acknowledged* have the type MES which is a subtype of the cartesian product DBM×DBM. Intuitively, this means that each token on one of these places represents a message (s,r). In the initial marking all messages are unused. Hence we have:

$$M_0(\text{Unused}) = \text{MES} = \{(s,r) \in \text{DBM} \times \text{DBM} \mid s \neq r\}$$

$$M_0(\text{Sent}) = M_0(\text{Received}) = M_0(\text{Acknowledged}) = \emptyset$$

where the requirement $s \neq r$ indicates that we do not have messages in which the sender and receiver are identical. The declaration of MES is a bit complex and specific to the declaration language which we have chosen. An explanation can be found in Sect. 1.3 of [27].

Active and *Passive* have the type E = {e} which has only one possible value. The initial marking looks as follows:

$$M_0(\text{Passive}) = \text{E} = 1`e$$

$$M_0(\text{Active}) = \emptyset.$$

For historical reasons we often refer to the token values as token **colours** and we also refer to the data types as **colour sets**. This is a metaphoric picture where we consider the tokens of a CP-net to be distinguishable from each other and hence “coloured” – in contrast to ordinary low-level Petri nets (PT-nets) which have “black” indistinguishable tokens. The types of a CP-net can be arbitrarily complex, e.g., a record where one field is a real, another a text string and a third a list of integers. Hence, it is much more adequate to imagine a continuum of colours (like in physics) instead of a few discrete colour values (like red, green and blue). Intuitively, we consider tokens of type E to be black tokens, like in PT-nets, and hence without any data. However, mathematically these tokens carry a data value, e, like all the other tokens of a CP-net.

The actions of a CP-net are represented by means of **transitions** (which are drawn as rectangles). In the data base system there are four different transitions.

An incoming arc indicates that the transition may remove tokens from the corresponding place while an outgoing arc indicates that the transition may add tokens. The exact number of tokens and their data values are determined by the **arc expressions** (which are positioned next to the arcs). The transition *Update and Send Messages (SM)* has six arcs with three different arc expressions: e, s and Mes(s). The first of these is a constant. The two other expressions contain the free variable s of type DBM, declared in the last line of the declarations. To talk about an **occurrence** of the transition *SM* we need to bind s to a value from DBM. Otherwise, we cannot evaluate the expressions s and Mes(s). As explained at the beginning of the section, the function Mes(s) maps each data base manager s into the messages which s sends. The declaration of Mes(s) is a bit complex and specific to the declaration language which we have chosen. An explanation can be found in Sect. 1.3 of [27].

Now let us assume that we bind the variable s (of the transition *SM*) to the value d_2. This gives us a binding $<s = d_2>$ for *SM*. Together with *SM* the binding forms a pair which we refer to as a **binding element**:

$(SM, <s = d_2>)$.

For this binding element we evaluate the arc expressions as follows (in the rest of this section we assume that there are n = 5 data base managers):

e	$\rightarrow$	e
s	$\rightarrow$	d_2
Mes(s)	$\rightarrow$	$1`(d_2,d_1) + 1`(d_2,d_3) + 1`(d_2,d_4) + 1`(d_2,d_5)$.

This tells us that an occurrence of the transition *SM* with the binding $<s = d_2>$ will remove a token with value e from *Passive* and add a token with value e to *Active*. The occurrence will also remove a token with value d_2 from *Inactive* and add a token with this value to *Waiting*. Finally, it will remove four tokens from *Unused* (with the values specified by the evaluation of Mes(s)) and add four tokens with these values to *Sent*. Intuitively, this means that the manager d_2 changes from being inactive to being waiting, and simultaneously sends a message to each of the other managers.

The occurrence of the binding element $(SM, <s = d_2>)$ is possible, if the six tokens to be removed exist, i.e., if *Passive* has an e token, *Inactive* a d_2 token and *Unused* have the four tokens specified by $Mes(d_2)$. In the initial marking M_0 this is the case, and hence we say that the binding element $(SM, <s = d_2>)$ is **enabled** in M_0. It is easy to verify that all the five possible bindings for *SM* yield binding elements which are enabled in the initial marking.

If the binding element $(SM, <s = d_2>)$ occurs, it removes tokens from its input places and adds tokens to its output places. When we interpret a net we often think of the tokens as being moved from one place to another, possibly with some change of their data values. However, in the mathematical formulation of a CP-net model there is no connection between particular input tokens and particular output tokens. The number of output tokens may differ from the number of input tokens and they may have data values which are of different types.

The other three transitions work in a similar way. The two transitions to the right use two different variables, s and r. Each of these must be bound to a value in DBM. It is easy to verify that all the possible bindings for the transitions *Receive a Message (RM)*, *Send an Acknowledgement (SA)* and *Receive all Acknowledgements (RA)* are **disabled** in the initial marking. For *RM* there is no token on *Sent*. For *SA* and *RA* there is no token on any of the input places. Hence, we conclude that the only thing that can happen in the initial marking is an occurrence of one of the five binding elements of *SM*. Each of these is enabled and the choice between them is non-deterministic. As soon as one of the binding elements has occurred, the others become disabled – because there is no longer a token on *Passive*. Hence we say that the binding elements are in **conflict** with each other.

Let us assume that (SM, $<s = d_2>$) occurs. We then reach a marking M_1 with the following tokens (we only list those places which have a non-empty marking):

$$M_1(\text{Inactive}) = \text{DBM} - 1`d_2 = 1`d_1 + 1`d_3 + 1`d_4 + 1`d_5$$

$$M_1(\text{Waiting}) = 1`d_2$$

$$M_1(\text{Sent}) = \text{Mes}(d_2) = 1`(d_2,d_1) + 1`(d_2,d_3) + 1`(d_2,d_4) + 1`(d_2,d_5)$$

$$M_1(\text{Unused}) = \text{MES} - \text{Mes}(d_2)$$

$$M_1(\text{Active}) = 1`e.$$

In the marking M_1 the transition *RM* is enabled – for all the four bindings in which s is bound to d_2 while r is bound to a value that differs from d_2. Intuitively, this means that all the other managers are ready to receive the message which d_2 has sent to them. The binding element (RM, $<s = d_2, r = d_3>$) moves a token with value (d_2, d_3) from *Sent* to *Received* and it moves a token with value d_3 from *Inactive* to *Performing*. Intuitively, this means that the manager d_3 changes from being inactive to being performing. Simultaneously the message (d_2, d_3) changes from being sent to being received. Analogously, the binding element (RM, $<s = d_2, r = d_4>$) moves a token with value (d_2, d_4) from *Sent* to *Received* and it moves a token with value d_4 from *Inactive* to *Performing*. The two binding elements deal with different tokens and hence they do not influence each other. This means that they can occur **concurrently** with each other, i.e., in the same **step**. It is easy to verify that all the four enabled binding elements are concurrent with each other. This means that the next step may contain all of them or any non-empty subset of them. The choice is again non-deterministic.

The conflict and concurrency relations (between binding elements) are marking dependent. If we add tokens, a conflict situation may be turned into a concurrency situation. As an example, we could change the initial marking of *Passive* to 3`e. Then we can have steps where up to three of the binding elements of *SM* occur concurrently. It is also allowed that a binding element may be concurrent with itself, one or more times. This means that, in general, a step is a multi-set of binding elements (which is demanded to be finite and non-empty). Let us assume that the step:

$1`(RM, <s = d_2, r = d_3>) + 1`(RM, <s = d_2, r = d_4>)$

occurs in the marking M_1. We then reach a marking M_2 with the following tokens:

$M_2(Inactive) = M_1(Inactive) - (1`d_3 + 1`d_4) = 1`d_1 + 1`d_5$

$M_2(Waiting) = 1`d_2$

$M_2(Performing) = 1`d_3 + 1`d_4$

$M_2(Sent) = M_1(Sent) - (1`(d_2,d_3) + 1`(d_2,d_4)) = 1`(d_2,d_1) + 1`(d_2,d_5)$

$M_2(Received) = 1`(d_2,d_3) + 1`(d_2,d_4)$

$M_2(Unused) = MES - Mes(d_2)$

$M_2(Active) = 1`e.$

In M_2 there are four enabled binding elements:

$(RM, <s = d_2, r = d_1>)$

$(RM, <s = d_2, r = d_5>)$

$(SA, <s = d_2, r = d_3>)$

$(SA, <s = d_2, r = d_4>).$

The four binding elements are concurrent with each other – because they use different tokens. Let us assume that the first three of them occur, i.e., that we have a step which looks as follows:

$1`(RM, <s = d_2, r = d_1>) + 1`(RM, <s = d_2, r = d_5>) + 1`(SA, <s = d_2, r = d_3>).$

We then reach a marking M_3 with the following tokens:

$M_3(Inactive) = (M_2(Inactive) - (1`d_1 + 1`d_5)) + 1`d_3 = 1`d_3$

$M_3(Waiting) = 1`d_2$

$M_3(Performing) = (M_2(Performing) - 1`d_3) + (1`d_1 + 1`d_5)$

$= 1`d_1 + 1`d_4 + 1`d_5$

$M_3(Sent) = M_2(Sent) - (1`(d_2,d_1) + 1`(d_2,d_5)) = Ø$

$M_3(Received) = (M_2(Received) - 1`(d_2,d_3)) + (1`(d_2,d_1) + 1`(d_2,d_5))$

$= 1`(d_2,d_1) + 1`(d_2,d_4) + 1`(d_2,d_5)$

$M_3(Acknowledged) = 1`(d_2,d_3)$

$M_3(Unused) = MES - Mes(d_2)$

$M_3(Active) = 1`e.$

In M_3 there are three enabled binding elements:

$(SA, <s = d_2, r = d_1>)$

$(SA, <s = d_2, r = d_4>)$

$(SA, <s = d_2, r = d_5>).$

A concurrent occurrence of the three binding elements leads to a marking M_4 with the following tokens:

$M_4(\text{Inactive}) = 1`d_1 + 1`d_3 + 1`d_4 + 1`d_5$

$M_4(\text{Waiting}) = 1`d_2$

$M_4(\text{Performing}) = M_4(\text{Sent}) = M_4(\text{Received}) = \emptyset$

$M_4(\text{Acknowledged}) = 1`(d_2,d_1) + 1`(d_2,d_3) + 1`(d_2,d_4) + 1`(d_2,d_5) = \text{Mes}(d_2)$

$M_4(\text{Unused}) = \text{MES} - \text{Mes}(d_2)$

$M_4(\text{Active}) = 1`e.$

In M_4 there is only one enabled binding element:

$(\text{RA}, <s = d_2>)$.

An occurrence of this binding element leads to the initial marking M_0. The markings and steps considered above form an **occurrence sequence**:

$$M_0[Y_0\rangle M_1[Y_1\rangle M_2[Y_2\rangle M_3[Y_3\rangle M_4[Y_4\rangle M_0.$$

The occurrence sequence which we have considered is only one out of infinitely many. Our occurrence sequence happens to be cyclic, i.e., it starts and ends in the same marking.

To ensure consistency between the different copies of the data base, this simple model only allows one update at a time. When a manager has initiated an update, this has to be performed by all the managers before another update can be initiated. The mutual exclusion is guaranteed by the place *Passive*. Our description of the data base system is very high-level (and unrealistic) – in the sense that the mutual exclusion is described by means of a global mechanism (the place *Passive*). To implement the data base system on distributed hardware, the mutual exclusion must be handled by means of a “distributed mechanism”. Such an implementation could be described by a more detailed CP-net – which could also model how to handle “loss of messages” and “disabled sites”.

The CP-net for the data base system has several redundant places – which could be omitted without changing the behaviour (i.e., the possible occurrence sequences). As an example, we can omit *Unused*. Then there will only be an explicit representation of those messages which currently are in use (i.e., in one of the states *Sent, Received* or *Acknowledged)*. We can also omit *Active* and *Performing*. It is very common to have redundant places in a CP-net, and this often makes the description easier to understand – because it gives a more detailed and more comprehensive description of the different states of the system.

Attaching a data value to each token allows us to use much fewer places than would be needed in a PT-net. For the data base system, a typical PT-net would have n places for each of the states *Inactive*, *Waiting* and *Performing* – because this is the only way in which a PT-net can distinguish between the n different managers. Analogously, there would be $|\text{MES}| = n^2 - n$ different places for the message states *Unused*, *Sent*, *Received* and *Acknowledged*. With 5 managers the PT-net would have 97 places while the CP-net only has 9. An even more dramatic reduction is obtained when we use more complex types.

The use of variables in arc expressions means that each CP-net transition can occur in many slightly different ways – in a similar way as a procedure can be executed with different input parameters. Hence, we can use a single transition to describe a class of related activities, while in a PT-net we need a transition for each instance of such an activity. In the data base system there is only one *SM* transition and one *RA* transition. In a typical PT-net there would be one for each manager. Analogously, there would be an *RM* and *SA* transition for each element of MES. With 5 managers the PT-net would have 50 transitions while the CP-net only has 4.

In this paper we only consider the data base system, which has rather simple arc expressions. However, it is possible to use much more complex arc expressions such as:

case x of p=>1`(x,i) | q=>empty.

When x is bound to p the arc expression evaluates to a multi-set with one token. When x is bound to q the arc expression evaluates to the empty multi-set. This example also illustrates the fact that different bindings for the same transition may remove and add a different number of tokens. For more details about the syntax and semantics of arc expressions, see Sect. 1.4 of [27].

In addition to the arc expressions, it is possible to attach a boolean expression (with variables) to each transition. The boolean expression is called a **guard**. It specifies that we only accept binding elements for which the boolean expression evaluates to true. In the data base system we could add the guard:

$[s \neq d_3]$

to the transition *SM* (by convention guards are written in square brackets, next to the transition). Such a guard would mean that we exclude (SM, $<s = d_3>$) from the set of binding elements, and hence it would no longer be possible for the data base manager d_3 to initiate an update.

The above informal explanation of the enabling and occurrence rules tells us how to understand the behaviour of a CP-net, and it explains the intuition on which CP-nets build. However, it is 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 Sect. 3). The formal definition forms the foundation for the analysis methods presented in Sects. 5–7.

It can be shown that each CP-net can be translated into a PT-net and vice versa – if the CP-net has infinite types, such as the integers, text strings or reals, the equivalent PT-net may become infinite. Since the expressive power of the two formalisms are the same, there is no theoretical gain by using CP-nets. However, in practice, CP-nets constitute a more compact, and much more convenient, modelling language than PT-nets – in a similar way as high-level programming languages are much more adequate for practical programming than assembly code and Turing machines.

CP-nets form a language in its own right, and this means that systems are modelled and analysed directly in terms of CP-nets – without thinking of PT-nets and without translating them into PT-nets. The benefits which we achieve by us-

ing CP-nets, instead of PT-nets, are very much the same as those achieved by using high-level programming languages instead of assembly languages:

- Description and analysis become more compact and manageable (because the complexity is divided between the net structure, the declarations and the net inscriptions).
- It becomes possible to describe data manipulations in a much more direct way (by using the arc expressions instead of a complex set of places, transitions and arcs).
- It becomes easier to see the similarities and differences between similar system parts (because they are represented by the same place, transition or subnet).
- The description is more redundant and this means that there will be less errors (because errors can be found by noticing inconsistencies, e.g., between the type of an arc expression and the type of the corresponding place). This is useful, in particular when we have a computer tool to perform the consistency checks.
- It is possible to create hierarchical descriptions, i.e., structure a large description as a set of smaller CP-nets with well-defined interfaces and relationships to each other. This is similar to the use of modules in a programming language. In this paper, we only deal with non-hierarchical CP-nets. An introduction and formal definition of hierarchical CP-nets can be found in Chap. 3 of [27]. It is easy to modify all the concepts and results, presented in the present paper, so that they work for hierarchical nets also. All the details can be found in [27] and [28].

2 Why use CP-nets?

There are three different – but closely related – reasons to make CPN models (and other kinds of behavioural models). First of all, a CPN model is a *description* of the modelled system, and it can be used as a specification (of a system which we want to build) or as a presentation (of a system which we want to explain to other people, or ourselves). By creating a model we can investigate a new system before we construct it. This is an obvious advantage, in particular for systems where design errors may jeopardise security or be expensive to correct. Secondly, the behaviour of a CPN model can be *analysed*, either by means of simulation (which is equivalent to program execution and program debugging) or by means of more formal analysis methods (to be presented in Sects. 6 and 7). Finally, the process of creating the description and performing the analysis usually gives the modeller a dramatically improved *understanding* of the modelled system – and it is often the case that this is more valid than the description and the analysis results themselves.

There exist so many different modelling languages that it would be very difficult and time consuming to make an explicit comparison with all of them (or even the most important of them). Instead we shall, in this section, make an implicit comparison, by listing some of those properties which make CP-nets a valuable language for the design, specification and analysis of many different

types of systems. It should be understood that many of the other modelling languages also fulfil some of the properties listed below, and it should also be understood that some of these languages have nice properties which are not found in CP-nets. We do *not* claim that CP-nets are superior to all the other languages. Such claims are, in our opinion, made far too often – and they nearly always turn out to be ridiculous. However, we do think that for some purposes CP-nets are extremely useful, and that, together with some of the other languages, they should be a standard part of the repertoire of advanced system designers and system analysts.

1. CP-nets have a graphical representation. The graphical form is intuitively very appealing. It is extremely easy to understand and grasp – even for people who are not very familiar with the details of CP-nets. This is due to the fact that CPN diagrams resemble many of the informal drawings which designers and engineers make while they construct and analyse a system. Just think about how often you have illustrated an algorithm or a communication protocol by drawing a directed graph, where the nodes represent states and actions, while the arcs describe how to go from one state to another, by executing some of the actions. The notions of states, actions and flow are basic to many kinds of system and these concepts are – in a very vivid and straightforward way – represented by the places, transitions and arcs of CP-nets.

2. CP-nets have a well-defined semantics which unambiguously defines the behaviour of each CP-net. It is the presence of the semantics which makes it possible to implement simulators for CP-nets, and it is also the semantics which forms the foundation for the formal analysis methods described in Sects. 6 and 7.

3. CP-nets are very general and can be used to describe a large variety of different systems. The applications of CP-nets range from informal systems (such as the description of work processes) to formal systems (such as communication protocols). They also range from software systems (such as distributed algorithms) to hardware systems (such as VLSI chips). Finally, they range from systems with a lot of concurrent processes (such as flexible manufacturing) to systems with no concurrency (such as sequential algorithms).

4. CP-nets have very few, but powerful, primitives. The definition of CP-nets is rather short and it builds upon standard concepts which many system modellers already know from mathematics and programming languages. This means that it is relatively easy to learn to use CP-nets. However, the small number of primitives also means that it is much easier to develop strong analysis methods.

5. CP-nets have an explicit description of both states and actions. This is in contrast to most system description languages which describe either the states or the actions – but not both. Using CP-nets, the reader may easily change the point of focus during the work. At some instances of time it may be convenient to concentrate on the states (and almost forget about the actions) while at other instances it may be more convenient to concentrate on the actions (and almost forget about the states).

6. CP-nets have a semantics which builds upon true concurrency, instead of interleaving. This means that the notions of conflict and concurrency can be defined in a very natural and straightforward way (as we have seen in Sect. 1). In

an interleaving semantics it is impossible to have two actions in the same step, and thus concurrency only means that the actions can occur after each other, in any order. In our opinion, a true-concurrency semantics is easier to work with – because it is closer to the way human beings usually think about concurrent actions.

7. CP-nets offer hierarchical descriptions. This means that we can construct a large CP-net by relating smaller CP-nets to each other, in a well-defined way. The hierarchy constructs of CP-nets play a role similar to that of subroutines, procedures and modules of programming languages. The existence of hierarchical CP-nets makes it possible to model very large systems in a manageable and modular way.

8. CP-nets integrate the description of control and synchronisation with the description of data manipulation. This means that on a single sheet of paper it can be seen what the environment, enabling conditions and effects of an action are. Many other graphical description languages work with graphs which only describe the environment of an action – while the detailed behaviour is specified separately (often by means of unstructured prose).

9. CP-nets can be extended with a time concept. This means that it is possible to use the same modelling language for the specification/validation of functional/ logical properties (such as absence of deadlocks) and performance properties (such as average waiting times). The basic idea behind the time extension is to introduce a global clock and to allow each token to carry a time stamp – in addition to the data value which it already has. Intuitively, the time stamp specifies the time at which the token is ready to be used, i.e., consumed by a transition. For more details about timed CP-nets, see [27], [28] and [29].

10. CP-nets are stable towards minor changes of the modelled system. This is proved by many practical experiences and it means that small modifications of the modelled system do not completely change the structure of the CP-net. In particular, it should be observed that this is also true when a number of subnets describing different sequential processes are combined into a larger CP-net. In many other description languages, e.g., finite automata, such a combination often yields a description which is difficult to relate to the original sub-descriptions.

11. CP-nets offer interactive simulations where the results are presented directly on the CPN diagram. The simulation makes it possible to debug a large model while it is being constructed – analogously to a good programmer debugging the individual parts of a program as he finishes them. The data values of the moving tokens can be inspected.

12. CP-nets have a large number of formal analysis methods by which properties of CP-nets can be proved. There are four basic classes of formal analysis methods: construction of occurrence graphs (representing all reachable markings), calculation and interpretation of system invariants (called place and transition invariants), reductions (which shrink the net without changing a certain selected set of properties) and checking of structural properties (which guarantee certain behavioural properties). In this paper we only deal with the first two classes of formal analysis methods.

13. CP-nets have computer tools supporting their drawing, simulation and formal analysis. This makes it possible to handle even large nets without drowning in details and without making trivial calculation errors. The existence of such computer tools is extremely important for the practical use of CP-nets.

In this section we have listed a number of advantages of CP-nets. Many of these are also valid for other kinds of high-level nets, PT-nets, and other kinds of modelling languages. Once more, we want to stress that we do not view CP-nets as "the superior" system description language. In contrast, we consider the world of computer science to be far too complicated and versatile to be handled by a single language. Thus we think CP-nets must be used together with many other kinds of modelling languages. It is often valuable to use different languages to describe different aspects of the system. The resulting set of descriptions should be consider as complementary, not as alternatives.

3 Formal Definition of CP-nets

This chapter contains the formal definition of (non-hierarchical) CP-nets and their behaviour. A CP-net is defined as a many-tuple. However, it should 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 CPN diagram (i.e., a diagram similar to Fig. 1). It is in principle (but not in practice) easy to translate a CPN diagram into a CP-net tuple, and vice versa. The tuple form is adequate when we want to formulate general definitions and prove theorems which apply to all (or a large class) of CP-nets. The graph form is adequate when we want to construct a particular CP-net modelling a specific system.

First we define multi-sets. $\mathbb{N}$ denotes the set of all non-negative integers and iff means "if and only if".

Definition 3.1: A **multi-set** m, over a non-empty set S, is a function $m \in [S \rightarrow \mathbb{N}]$ which we represent as 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 the **coefficients** of the multi-set. $s \in m$ iff $m(s) \neq 0$.

We use Ø to denote the empty multi-set (there is an empty multi-set for each element set S; we ignore this and speak about *the* empty multi-set – in a similar way that we speak about *the* empty set).

There is a one-to-one correspondence between sets over S and those multi-sets for which all coefficients are zero or one. Thus we shall, without any further comments, use a set $A \subseteq S$ to denote the multi-set which consists of one appearance of each element in A. Analogously, we use an element $s \in S$ to denote the multi-set 1`s.

For multi-sets we have a number of standard operations (with the exception of $|m|$ all of them are derived from standard operations for functions).

Definition 3.2: Addition, **scalar multiplication**, **comparison**, and **size** of multi-sets are defined in the following way, for all m, m_1, $m_2 \in S_{MS}$ and all $n \in \mathbb{N}$:

(i) $m_1 + m_2 = \sum_{s \in S} (m_1(s) + m_2(s)) `s$ (addition).

(ii) $n * m = \sum_{s \in S} (n * m(s)) `s$ (scalar multiplication).

(iii) $m_1 \neq m_2 = \exists s \in S: m_1(s) \neq m_2(s)$ (comparison; $\geq$ and = are defined analogously to $\leq$).
$m_1 \leq m_2 = \forall s \in S: m_1(s) \leq m_2(s)$

(iv) $|m| = \sum_{s \in S} m(s)$ (size).

When $|m| = \infty$ we say that m is **infinite**. Otherwise m is **finite**. 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).

The multi-set operations have a large number of the standard algebraic properties. As an example $(S_{MS}, +)$ is a **commutative monoid**. More details can be found in Sect. 2.1 of [27].

To give the abstract definition of CP-nets it is not necessary to fix the concrete syntax in which the modeller specifies the net 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 *elements of a type*, T. The set of all elements in T is denoted by the type name T itself.
- 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).
- A *binding of a set of variables*, V – associating with each variable $v \in V$ an element $b(v) \in Type(v)$.
- The *value obtained by evaluating an expression*, expr, *in a binding*, b – denoted by expr<b>. Var(expr) is required to be a subset of the variables of b, and the evaluation is performed by substituting for each variable $v \in Var(expr)$ the value $b(v) \in Type(v)$ determined by the binding.

An expression *without* variables is said to be a **closed** expression. It can be evaluated in all bindings, and all evaluations give the same value – which we often shall denote by the expression itself. This means that we simply write “expr” instead of the more pedantic “expr<b>”.

We use $\mathbb{B}$ to denote the boolean type (containing the elements {false, true} and having the standard operations). Moreover, we extend the notation Type(v)

to $Type(A) = \{Type(v) \mid v \in A\}$ where A is a set of variables. In the rest of this paper, we shall make such extensions without explicit notice.

Now we are ready to define CP-nets. Some motivations and explanations of the individual parts of the definition are given immediately below the definition, and it is recommended that these are read in parallel with the definition. More comments can be found in Sect. 2.2 of [27].

Definition 3.3: A **CP-net** is a tuple $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ where:

(i) Σ is a finite set of non-empty **types**, also called colour sets.
(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:
- $\forall t \in T: [Type(G(t)) = \mathbb{B} \wedge Type(Var(G(t))) \subseteq \Sigma]$.

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

where p is the place of N(a).
(ix) I is an **initialisation** function. It is defined from P into closed expressions such that:
- $\forall p \in P: [Type(I(p)) = C(p)_{MS}]$.

(i) The set of **types** determines the data values and the operations and functions that can be used in the net expressions (i.e., arc expressions, guards and initialisation expressions). If desired, the types (and the corresponding operations and functions) can be defined by means of a many-sorted sigma algebra (as in the theory of abstract data types). We assume that each type has at least one element.

(ii) + (iii) + (iv) The **places**, **transitions** and **arcs** are described by three sets P, T and A which are required to be finite and pairwise disjoint. By requiring the sets of places, transitions and arcs to be finite, we avoid a number of technical problems such as the possibility of having an infinite number of arcs between two nodes.

(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). 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 $P \times T \cup T \times P$). Multiple arcs is a modelling convenience. For theory, they do not add or change anything.

(vi) The **colour** function C maps each place, p, to a type C(p). Intuitively, this means that each token on p must have a data value that belongs to C(p).

(vii) The **guard** function G maps each transition, t, into a boolean expression where all variables have types that belong to Σ. When we draw a CP-net we omit guard expressions which always evaluate to true.

(viii) The **arc expression** function E maps each arc, a, into an expression of type $C(p)_{MS}$. This means that each arc expression must evaluate to multi-sets over the type of the adjacent place, p. We allow a CPN diagram to have an arc expression expr of type C(p), and consider this to be a shorthand for 1`(expr).

(ix) The **initialisation** function I maps each place, p, into a closed expression which must be of type $C(p)_{MS}$. When we draw a CP-net we omit initialisation expressions which evaluate to Ø.

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.

Having defined the structure of CP-nets, we are now ready to consider their behaviour – but first we introduce the following notation for all $t \in T$ and for all pairs of nodes $(x_1,x_2) \in (P \times T \cup T \times P)$:

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

The summation indicates addition of expressions (and it is well-defined because all the participating expressions have a common multi-set type). From the argument(s) it will always be clear whether we deal with the function $E \in [A \rightarrow Expr]$ or the function $E \in [(P \times T \cup T \times P) \rightarrow Expr]$. A similar remark applies to A, A(t) and $A(x_1,x_2)$. Notice that $A(x_1,x_2) = Ø$ implies that $E(x_1,x_2) = Ø$ (where the latter Ø denotes the closed expression which evaluates to the empty multi-set).

Next we define bindings. It should be noted that (ii) implies that all bindings satisfy the corresponding guard. As defined below Def. 3.2, G(t)<b> denotes the evaluation of the guard expression G(t) in the binding b:

Definition 3.4: A **binding** of a transition t is a function b defined on Var(t), such that:

(i) $\forall v \in Var(t): b(v) \in Type(v)$.
(ii) G(t)<b>.

By B(t) we denote the set of all bindings for t.

As shown in Sect. 1 we often write bindings in the form $<v_1=c_1,v_2=c_2,\ldots,v_n=c_n>$ where $Var(t) = \{v_1,v_2,\ldots,v_n\}$. The order of the variables has no importance.

Definition 3.5: A **token element** is a pair (p,c) where $p \in P$ and $c \in C(p)$, while a **binding element** is a pair (t,b) where $t \in T$ and $b \in B(t)$. The set of all token elements is denoted by TE while the set of all binding elements is denoted by BE.

A **marking** is a multi-set over TE while a **step** is a *non-empty* and *finite* multi-set over BE. The **initial marking** M_0 is the marking which is obtained by evaluating the intitialisation expressions:

$$\forall (p,c) \in TE: M_0(p,c) = (I(p))(c).$$

The sets of all markings and steps are denoted by $\mathbb{M}$ and $\mathbb{Y}$, respectively.

Each marking $M \in TE_{MS}$ determines a unique function M* defined on P such that $M^*(p) \in C(p)_{MS}$:

$$\forall p \in P\ \forall c \in C(p): (M^*(p))(c) = M(p,c).$$

On the other hand, each function M*, defined on P such that $M^*(p) \in C(p)_{MS}$ for all $p \in P$, determines a unique marking M:

$$\forall (p,c) \in TE: M(p,c) = (M^*(p))(c).$$

Thus we shall often represent markings as functions defined on P (and we shall use the same name for the function and the multi-set representation of a marking).

Now we are ready to give the formal definition of enabling and occurrence (in which we represent steps by multi-sets while we represent markings by functions). Some explanations follow below.

Definition 3.6: A step Y is **enabled** in a marking M iff the following property is satisfied:

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

We then say that (t,b) is enabled and we also say that t is enabled. The elements of Y are concurrently enabled (when $|Y| \geq 1$).

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:

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

M_2 is **directly reachable** from M_1. This is written: $M_1[Y\rangle M_2$.

The expression evaluation $E(p,t)\langle b \rangle$ gives us the tokens, which are removed from p when t occurs with the binding b. By taking the sum over all binding elements $(t,b) \in Y$ we get all the tokens that are removed from p when Y occurs. This multi-set is required to be less than or equal to the marking of p. It means that each binding element $(t,b) \in Y$ must be able to get the tokens specified by $E(p,t)\langle b \rangle$, without having to share these tokens with other binding elements of

Y. It should be remembered that all bindings of a step, according to Def. 3.4, automatically satisfy the corresponding guards. Moreover, it should be noted that the summations in Def. 3.6 are summations over a multi-set Y. When a binding element appears more than once in Y, we get a contribution for each appearance.

The occurrence of a step is an indivisible event. Although the formula above requires the subtraction to be performed before the addition we do *not* recognise the existence of an intermediate marking, where the tokens in the first sum have been removed while those in the second have not yet been added. It should also be noted that a step does *not* need to be maximal. When a number of binding elements are concurrently enabled, it is possible to have an occurring step which only contains some of them.

Definition 3.7: A **finite occurrence sequence** is a sequence of markings and steps:

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

such that $n \in \mathbb{N}$, and $M_i[Y_i\rangle M_{i+1}$ for all $i \in \{1,2,\ldots,n\}$ M_1 is the **start marking**, M_{n+1} is the **end marking** and n is the **length**.

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

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

such that $M_i[Y_i\rangle M_{i+1}$ for all $i \geq 1$.

A marking M'' is **reachable from** a marking M' iff there exists a finite occurrence sequence starting in M' and ending in M''. The *set* of markings which are reachable from M' is denoted by $[M'\rangle$. A marking is **reachable** iff it belongs to $[M_0\rangle$.

We allow occurrence sequences of length zero. This means that $M \in [M\rangle$ for all markings M. Often we omit some parts of an occurrence sequence, e.g., all the intermediate markings. In particular, we use:

$$M[t,b\rangle \quad \text{and} \quad M[t\rangle$$

to denote that the binding element (t,b) and the transition t is enabled in the marking M.

4 Dynamic Properties of CP-nets

Dynamic properties characterise the behaviour of individual CP-nets, e.g., whether it is possible to reach a marking in which no step is enabled. It is often rather difficult to verify dynamic properties – in particular when relying only on informal arguments. However, in Sects. 6 and 7 we shall introduce a number of formal analysis methods which can be used to prove dynamic properties. In this paper we only introduce some of the most important dynamic properties.

A much more complete set of dynamic properties can be found in Chap. 4 of [27].

Boundedness properties tell us how many tokens we may have at a particular place:

Definition 4.1: Let a place $p \in P$, a non-negative integer $n \in \mathbb{N}$ and a multi-set $m \in C(p)_{MS}$ be given.

(i) n is an **integer bound** for p iff:
$\forall M \in [M_0\rangle: |M(p)| \leq n$.

(ii) m is a **multi-set bound** for p iff:
$\forall M \in [M_0\rangle: M(p) \leq m$.

For the data base system (with n managers) we have the following bounds. All of them are optimal, i.e., the smallest possible bounds:

	Multi-set	Integer
Inactive	DBM	n
Waiting	DBM	1
Performing	DBM	n−1
Unused	MES	n^2-n
Sent, Received, Acknowledged	MES	n−1
Passive, Active	E	1

Notice that multi-set bounds and integer bounds supplement each other. From one of them it is often possible to deduce information which cannot be deduced from the other, and vice versa. This is, e.g., the case for the places *Waiting* and *Inactive*. For *Waiting* the integer bound gives us much more precise information than the multi-set bound. For *Inactive* it is the other way round.

Home properties tell us about markings (or sets of markings) to which it is always possible to return:

Definition 4.2: Let a marking $M \in \mathbb{M}$ and a set of markings $X \subseteq \mathbb{M}$ be given:

(i) M is a **home marking** iff:
$\forall M' \in [M_0\rangle: M \in [M'\rangle$.

(ii) X is a **home space** iff:
$\forall M' \in [M_0\rangle: X \cap [M'\rangle \neq \emptyset$.

It is easy to see that M is a home marking iff {M} is a home space. Notice that the existence of a home marking tells us that it is possible to reach the home marking. However, it is not guaranteed that we ever do this. In other words, there may exist infinite occurrence sequences which do not contain the home marking. A similar remark applies to home spaces.

For the data base system it can be shown that the initial marking is a home marking. From this it follows that any reachable marking is a home marking.

Liveness properties tell us that a set of binding elements X remains active. This means that it is possible, for each reachable marking M', to find an occurrence sequence starting in M' and containing an element from X.

Definition 4.3: Let a marking $M \in \mathbb{M}$ and a set of binding elements $X \subseteq BE$ be given.

(i) M is **dead** iff no binding element is enabled, i.e., iff:
$\forall x \in BE: \neg M[x\rangle$.

(ii) X is **dead** in M iff no element of X can become enabled, i.e., iff:
$\forall M' \in [M\rangle\ \forall x \in X: \neg M'[x\rangle$.

(iii) X is **live** iff there is no reachable marking in which X is dead, i.e., iff:
$\forall M' \in [M_0\rangle\ \exists M'' \in [M'\rangle\ \exists x \in X: M''[x\rangle$.

Liveness only demands that elements of X can become enabled. Thus there may be infinite occurrence sequences starting in M' and containing no elements of X. It should be noted that live is *not* the negation of dead. Each live set of binding elements is non-dead – but the opposite is not true. For a transition $t \in T$, we use $BE(t) \subseteq BE$ to denote the set of all those binding elements which contain t. We say that t is dead or live iff BE(t) possesses the corresponding property. We also say that t is **strictly live** iff $\{x\}$ is live for all $x \in BE(t)$.

For the data base system, we have that all four transitions are strictly live. This can be verified by proving that the initial marking M_0 is a home marking and that there exists an occurrence sequence which starts in M_0 and contains all binding elements of BE.

Fairness properties tell us how often the different binding elements occur. For a set of binding elements $X \subseteq BE$ and an infinite occurrence sequence σ of the form:

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

we use $EN_{X,i}(\sigma)$ to denote the number of elements from X which are enabled in the marking M_i (when an element is concurrently enabled with itself this is reflected in the count). Analogously, we use $OC_{X,i}(\sigma)$ to denote the number of elements from X which occur in the step Y_i (when an element occurs concurrently with itself this is reflected in the count).

We use $EN_X(\sigma)$ and $OC_X(\sigma)$ to denote the total number of enablings/ occurrences in σ, i.e.:

$$EN_X(\sigma) = \sum_{i=1}^{\infty} EN_{X,i}(\sigma) \quad \text{and} \quad OC_X(\sigma) = \sum_{i=1}^{\infty} OC_{X,i}(\sigma).$$

Since all elements in the two sums are non-negative integers, it is easy to see that the sums must be convergent – either to an element of $\mathbb{N}$ or to ∞.

Definition 4.4: Let $X \subseteq BE$ be a set of binding elements and σ be an infinite occurrence sequence.

(i) X is **impartial** for σ iff it has infinitely many occurrences, i.e., iff: $OC_X(\sigma) = \infty$.

(ii) X is **fair** for σ iff an infinite number of enablings implies an infinite number of occurrences, i.e., iff: $EN_X(\sigma) = \infty \Rightarrow OC_X(\sigma) = \infty$.

(iii) X is **just** for σ iff a persistent enabling implies an occurrence, i.e., iff: $\forall i \geq 1: [EN_{X,i}(\sigma) \neq 0 \Rightarrow \exists k \geq i: [EN_{X,k}(\sigma) = 0 \vee OC_{X,k}(\sigma) \neq 0]]$.

When X is impartial for all infinite occurrence sequences of the given CP-net (starting in a reachable marking), we say that X is impartial. Analogous definitions are made for fair and just.

We say that a transition $t \in T$ is impartial, fair or just iff BE(t) possesses the corresponding property. We also say that t is **strictly impartial** (**strictly fair** / **strictly just**) iff {x} is impartial (fair / just) for all $x \in BE(t)$. It is reasonably easy to prove that impartiality implies fairness, which in turn implies justice.

For the data base system, we have that all four transitions are impartial. *SM* is strictly just, while the other three transitions are strictly fair.

5 Simulation

Simulation of CP-nets can be supported by a computer tool or it can be totally manual, for example, performed on a blackboard or in the head of a modeller. Simulation is similar to the debugging of a program, in the sense that it can reveal errors, but never be sufficient to give a full proof for the correctness of a system. Some people argue that this makes simulation uninteresting and that the modeller should concentrate on the more formal analysis methods presented in Sects. 6 and 7. We do not agree with this conclusion. On the contrary, we consider simulation to be just as important and necessary as the 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 possible 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 the simulation activity. With this rephrasing the answer becomes trivial. Of course, we want computer support. This means that the simulations 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 during the early investigation of a system design, while the more formal analysis methods are used for validation.

The CPN simulator described in [13] represents the ongoing simulation directly on the CPN diagram – by high-lighting the enabled and occurring transitions and by showing how the markings of the individual places change. In an **interactive** simulation the steps are chosen by the user. This is done under strict supervision and guidance by the CPN simulator, which, e.g., checks the legality and the enabling of all the proposed binding elements. In an **automatic** simulation the steps are chosen by the CPN simulator. This is done by means of a random number generator. In both cases it is the CPN simulator that performs the really complex work: the calculation of the enablings and the calculation of the effects of the occurring steps.

It is possible to vary the amount of graphical feedback which is provided by the CPN simulator. In the most detailed mode the user watches all the occurring transitions. He sees the input tokens, the output tokens, and the current marking. There is a trade-off between information and speed. With the most detailed feedback it is impossible to execute and observe more than a few steps per minute. With less feedback, the speed can be increased by a factor 100–1000.

The CPN simulator is designed to work with complex CP-nets. Some industrial applications use CPN models with more than 500 transitions. Fortunately, it turns out that a large CP-net simulates almost as fast as a small CP-net (when speed is measured in terms of the number of occurring binding elements). The reason for this is the locality of the enabling and occurrence rule. When a transition has occurred, it is only necessary to recalculate the enabling of the nearest neighbours (and the number of these are often independent of the size of the model). The calculation of the new enabling is the most expensive part of the simulation. Without local rules for enabling and occurrence, the calculation would grow linearly with the model size and that would make it very cumbersome to simulate large systems.

Above, we have only mentioned a small fraction of the facilities in the CPN simulator. More thorough descriptions can be found in [13] and in [27].

6 Occurrence Graphs

In this section we deal with occurrence graphs (which are also called reachability graphs or state spaces). The basic idea behind occurrence graphs is to construct a graph with a node for each reachable marking and an arc for each occurring binding element. Obviously such a graph may become very large, even for small CP-nets. Fig. 2 shows the occurrence graph for the data base system with 3 managers.

Each node represents a reachable marking. To save space (in our drawing) we represent the marking by listing those managers which have a message addressed to them – on *Sent*, *Received* or *Acknowledged*, respectively. This means that (2,3,–) denotes a marking in which d_1 is *Waiting*, while d_2 is *Inactive* and d_3 *Performing*. Analogously (23,–,–) denotes a marking in which d_1 is *Waiting*, while d_2 and d_3 are *Inactive*. The initial marking is represented by (–,–,–). This

node is drawn twice – to avoid long arcs. The second copy has a dashed border line.

Each arc represents an occurrence $M_1[b\rangle M_2$ where M_1 and M_2 are reachable markings while b is a binding element enabled in M_1. We write SM, i and RM, i, k instead of $(SM, <s = d_i>)$ and $(RM, <s = d_i, r = d_k>)$, and analogously for *SA* and *RA*.

Definition 6.1: A **directed graph** is a tuple DG = (V, A, N) such that:

(i) V is a set of **nodes** (or **vertices**).
(ii) A is a set of **arcs** (or **edges**) such that:
- $V \cap A = \emptyset$.

(iii) N is a **node** function. It is defined from A into $V \times V$.

DG is **finite** iff V and A are finite.

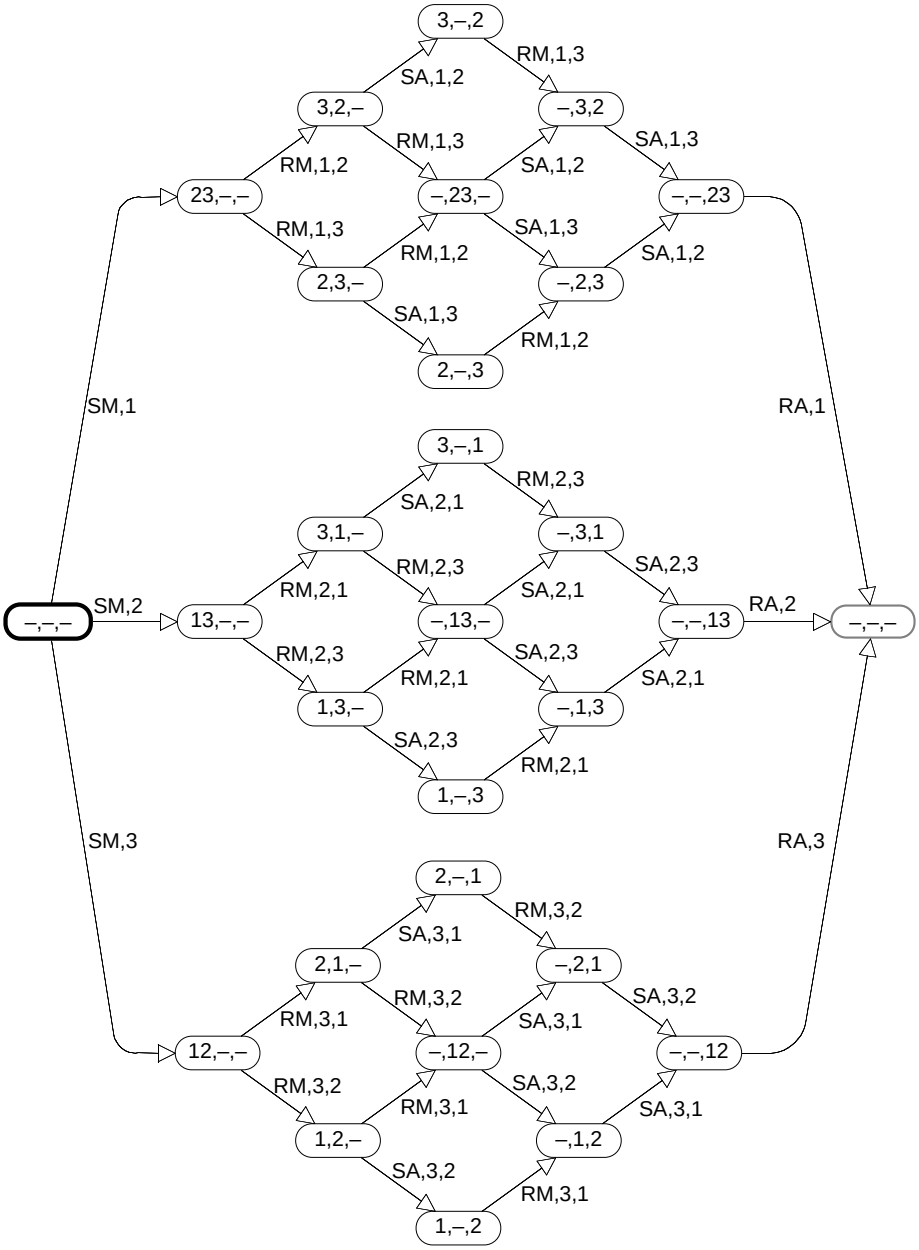

Fig. 2. Occurrence graph for data base system with 3 managers

It should be noted that, in contrast to classical graph theory, we allow a directed graph 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 $V \times V$).

An arc a with $N(a) = (v_1,v_2)$ is said to go from the **source node** v_1 to the **destination node** v_2, and we define two functions $s, d \in [A \rightarrow V]$. The first function maps each arc into its source node, while the second maps each arc into its destination node.

Definition 6.2: The **full occurrence graph** of a CP-net, also called the **O-graph**, is the directed graph OG = (V, A, N) where:

(i) $V = [M_0\rangle$.
(ii) $A = \{(M_1,b,M_2) \in V \times BE \times V \mid M_1[b\rangle M_2\}$.
(iii) $\forall a=(M_1,b,M_2) \in A$: $N(a) = (M_1,M_2)$.

When we have a CP-net where all variables (in the arc expressions and guards) have finite types, it is straightforward to prove that the O-graph is finite iff all places are bounded. Notice that an occurrence graph only contains arcs that correspond to steps with a single binding element. Otherwise, we would have had, e.g., an arc from node (23,–,–) to node (–,23,–), with the inscription $1`(RM,1,2)+1`(RM,1,3)$. Such arcs would give us information about the concurrency between binding elements, but they are not necessary for the verification of the kind of dynamic properties defined in Sect. 4.

When we draw O-graphs, like the one in Fig. 2, we usually inscribe each node with a text string describing the marking which the node represents. To save space, we sometimes use a condensed representation of the marking. Analogously, we inscribe each arc with the binding element which it represents. For an arc (M_1,b,M_2), it would be redundant to include the two markings M_1 and M_2 in the arc inscription – because these two markings are already described via the node inscriptions of the source node and the destination node.

Below we give an abstract algorithm to construct the O-graph. *W* is a set of nodes. It contains those nodes for which we have not yet found the successors, i.e., the nodes that wait to be processed. *Node(M)* is a procedure which creates a new node M, and adds M to *W*. If M is already a node, the procedure has no effect. Analogously, *Arc(M_1,b,M_2)* is a procedure which creates a new arc (M_1,b,M_2) with source M_1 and destination M_2. If (M_1,b,M_2) is already an arc, the procedure has no effect (this never happens for O-graphs but it may happen for OE-graphs and OS-graphs which we introduce later in this section). For a marking $M_1 \in \mathbb{M}$ we use *Next(M_1)* to denote the set of all possible "next moves":

$$Next(M_1) = \{(b,M_2) \in BE \times \mathbb{M} \mid M_1[b\rangle M_2\}.$$

Proposition 6.3: The following algorithm constructs the O-graph. The algorithm halts iff the O-graph is finite. Otherwise the algorithm continues forever, producing a larger and larger subgraph of the O-graph.

```
W := Ø
Node(M0)
repeat
    select a node M1 ∈ W
    for all (b,M2) ∈ Next(M1) do
    begin
        Node(M2)
        Arc(M1,b,M2)
    end
    remove M1 from W
until W = Ø.
```

Proof: Straightforward consequence of Def. 6.2. □

When the O-graph is infinite or too big to be constructed, by the available computing power, it may still be very useful to construct a partial O-graph, i.e., a subgraph of the O-graph. A discussion of this can be found in [28].

We define finite directed paths and strongly connected components in the usual way (a strongly connected component is a subgraph in which there exists a directed path from any node to any node). The detailed definitions can be found in Chap. 1 of [28] and in most text books on graph theory.

We use SCC to denote the set of all strongly connected components (of a given directed graph), and we use v^c to denote the strongly connected component to which a node v belongs. A similar notation is used for arcs.

Definition 6.4: The directed graph DG* = (V*, A*, N*) is the **SCC-graph** of a directed graph DG = (V, A, N) iff the following properties are satisfied:

(i) $V^* = SCC$.
(ii) $A^* = \{a \in A \mid s(a)^c \neq d(a)^c\}$.
(iii) $\forall a \in A^*: N^*(a) = (s(a)^c, d(a)^c)$.

The SCC-graph contains a node for each strongly connected component of DG. The SCC-graph contains those arcs (among the arcs of DG) which connect two different strongly connected components. Intuitively, we can obtain the SCC-graph by folding the original graph. We position all nodes of each strongly connected component "on top of each other" and we lump them into a single node which has all the arcs of the original nodes – with exception of those arcs which start and end in the same component.

For CP-nets with a cyclic behaviour, we often have O-graphs with a single strongly connected component, and hence the SCC-graph has a single node and no arcs. This is the case for the data base system.

By means of the occurrence graph in Fig. 2 (and the corresponding SCC-graph) it is possible to investigate the dynamic properties of the data base system. This is done by using a set of propositions, called **proof rules**. The proof rules relate properties of the occurrence graph and the SCC-graph to dynamic properties of the CP-net. In Prop. 6.5 we use SCC_T to denote those nodes of the SCC-graph which have no outgoing arcs. We use $(t,b) \in A$ to denote that a binding element (t,b) can be found on one of the arcs of the O-graph.

Proposition 6.5: For O-graphs we have the following **proof rules** which are valid for all $p \in P$ and all $t \in T$:

(i) $\text{BestIntegerBound}(p) = \max\{|M(p)| \mid M \in V\}$.

(ii) $\text{BestMulti-setBound}(p) = \sum_{c \in C(p)} \max\{M(p,c) \mid M \in V\} \grave{} c$.

(iii) $|SCC| = 1 \Rightarrow M_0$ is a home marking.

(iv) $|SCC| = 1 \wedge \forall b \in B(t): (t,b) \in A \Rightarrow t$ is strictly live.

The proof rules in Prop 6.5 are specialisations of rules given in Chap. 1 of [28] (which also contains a large number of other proof rules, e.g., for the fairness properties). The proofs of the proof rules are rather straightforward. They can be found in [28].

It is easy to see that the proof rules in Prop. 6.5 allow us to verify the boundedness, home and liveness properties which we have postulated for the data base system in Sect. 4. To use (i) and (ii) in Prop. 6.5, it is convenient to expand the condensed marking representation used in Fig. 2, so that we can see the marking of those places p we are interested in. In [28] it is shown how to verify fairness properties by means of occurrence graphs.

Even for a small occurrence graph, like the one in Fig. 2, the construction and investigation are tedious and error-prone. In practice it is not unusual to handle CP-nets which have occurrence graphs containing more than 100,000 nodes (and many CP-nets have millions of markings). Thus it is obvious that we have to construct and investigate the occurrence graphs by means of a computer. A detailed description of an occurrence graph tool can be found in [14], [27] and [28]. We also want to develop techniques by which we can construct reduced occurrence graphs without losing too much information. Below we sketch one way to obtain such a reduction. All the formal details can be found in [28].

Many systems contain some kind of symmetry. In the data base system we treat all managers in a similar way. Hence we may interchange them – without modifying the behaviour of the system. As an example, there is a lot of similarities between the markings (1,–,3) and (2,–,3). They are symmetrical, in the sense that one of them can be obtained from the other by interchanging d_1 with d_2 in all tokens, i.e., by means of a permutation of DBM.

We use Φ_{DBM} to denote all permutations of DBM. We say that $\phi \in \Phi_{DBM}$ is a **symmetry** and that it maps the marking M into a symmetrical marking $\phi(M)$, which we obtain from M by replacing each $d \in$ DBM by $\phi(d)$. We also say that M and $\phi(M)$ are **equivalent** and write $M \approx_M \phi(M)$. Φ_{DBM} is an algebraic group (with functional composition as the law of composition and identity function as

neutral element). This implies that $\approx_M$ becomes an equivalence relation. Similar notation and terminology are used for binding elements and for multi-sets of token values. All the arc expressions of the data base system satisfy the following property for all $\phi \in \Phi_{DBM}$ and all $b \in B(t)$ where t is the transition of the arc a:

$(*)$ $\quad E(a)<\phi(b)> = \phi(E(a)<b>).$

Intuitively, this means that symmetrical bindings have symmetrical effects. We can obtain the tokens used for the binding $\phi(b)$ by applying ϕ to the tokens used for b. Property $(*)$ is local and static. It can be determined from the individual transitions without considering the set of all reachable markings. From $(*)$ we can prove the following dynamic property which is satisfied for all markings $M', M'' \in [M_0\rangle$, all binding elements $b \in BE$ and all $\phi \in \Phi_{DBM}$:

$(**)$ $\quad M'[b\rangle M'' \Leftrightarrow \phi(M')[\phi(b)\rangle \phi(M'').$

Intuitively, this means that symmetrical markings have symmetrical enabled binding elements which will lead to symmetrical successor markings. When $(**)$ is satisfied, it makes sense to construct occurrence graphs where we only have a node for each equivalence class of markings and an arc for each equivalence class of occurring binding elements. Such a graph is called an **occurrence graph with symmetries** (or an OS-graph). The OS-graph for the data base system with three managers is shown in Fig. 3. As usual we have represented each equivalence class by means of one of its members.

An OS-graph is often much smaller than the corresponding O-graph. For the data base system the O-graph grows with exponential speed while the OS-graph only grows quadratic (the detailed calculations can be found in [28]). Fig. 4 illustrates the space complexity of O-graphs and OS-graphs. However, it is equally important how much time it takes to generate the two kinds of occurrence graphs. The time complexity of the O-graph construction is of order $O(n^2 * 3^n)$, while the time complexity of the OS-graph construction is of order $O(n^3)$.

Although OS-graphs are often much smaller than the corresponding O-graphs, they are – to a very large degree – as powerful with respect to the verification of dynamic properties. The proof rules in Prop. 6.6 are closely related to those in Prop. 6.5. We use $*$ to denote multiplication of the multi-set DBM by an integer. As before, we use $(t,b) \in A$ to denote that a binding element

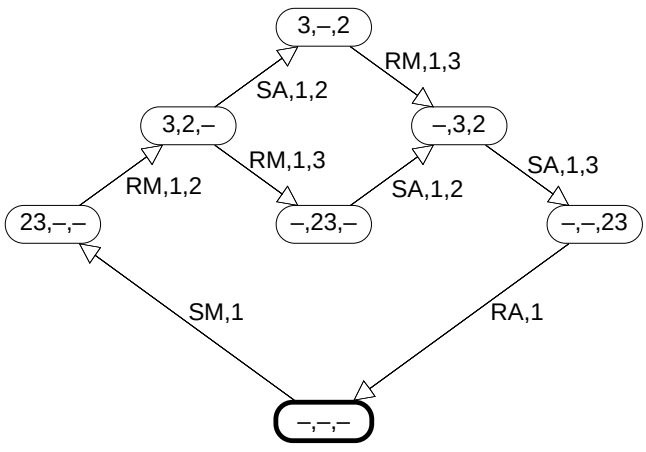

Fig. 3. OS-graph for data base system with 3 managers

(t,b) can be found on one of the arcs of the graph. However, it should be noticed that each arc now represents an equivalence classes of binding elements. By $(t,b) \in A$ we demand (t,b) to belong to one of these equivalence classes.

Proposition 6.6: For OS-graphs we have the following **proof rules** which are valid for all $p \in P$ and all $t \in T$:

(i) $\text{Best Integer Bound}(p) = \max\{|M(p)| \mid M \in V\}$.
(ii) $\text{Best Multi-set Bound}(p) = \max\{M(p,c) \mid M \in V \wedge c \in C(p)\} * C(p)$.
(iii) $|SCC| = 1 \Rightarrow M_0$ is a home marking.
(iv) $|SCC| = 1 \wedge \forall b \in B(t): (t,b) \in A \Rightarrow t$ is strictly live.

Rule (ii) is only valid when all the values of type C(p) can be mapped into each other by means of the allowed set of permutations. Otherwise we get a slightly more complex rule, because we have to make a separate investigation for each equivalence class of C(p).

Above, we have sketched the main ideas behind OS-graphs. All the formal definitions can be found in [28]. In general, we allow the use of subgroups of permutations (e.g., all rotations) and we also allow cartesian products to be supplemented by other structuring mechanisms, e.g., lists, records and discrete unions.

Permutations is only one way to obtain symmetries. In general, it is not necessary to impose any restrictions on the way in which a symmetry maps markings into markings and binding elements into binding elements – as long as we satisfy the consistency requirement in (**). It is also possible to start directly with equivalence relations for markings and binding elements (and replace (**) by a weaker consistency requirement). Then we obtain **occurrence graphs with equivalence classes** (also called OE-graphs). OE-graphs can be used for a wide variety of purposes – because they allow many different kinds of equivalence relations to be used. The theory of OE-graphs and OS-graphs (with general symmetries) can be found in [28].

	O-graph		OS-graph	
$\lvert DBM \rvert$ $O(n)$	Nodes $O(n * 3^n)$	Arcs $O(n^2 * 3^n)$	Nodes $O(n^2)$	Arcs $O(n^2)$
2	7	8	4	4
3	28	42	7	8
4	109	224	11	14
5	406	1,090	16	22
6	1,459	4,872	22	32
7	5,104	20,426	29	44
8	17,497	81,664	37	58
9	59,050	314,946	46	74
10	196,831	1,181,000	56	92
15	71,744,536	669,615,690	121	212
20	23,245,229,341	294,439,571,680	211	382

Fig. 4. The size of the O-graphs and OS-graphs for the data base system

The original ideas behind occurrence graphs with symmetries were developed in [21]. Similar reduction methods are described in [6], [7], [10] and [15]. The first two of these papers deal with a subclass of CP-nets, called Well-Formed CP-nets, while the last two deal with more general transition systems.

There are several other ways to reduce occurrence graphs. **Stubborn sets** discard some of the many orderings in which concurrent binding elements may occur (see [45] and [46]). **Covering markings** take care of the situation in which some places become unbounded (see [16] and [31]). It is widely believed that it is possible to combine the use of symmetries, stubborn sets and covering markings (see [21] and [39]). Intuitively, this is rather obvious. However, the mathematics behind the combined methods become rather hairy.

Analysis by means of occurrence graphs has several attractive properties. First of all, it is extremely easy to use occurrence graphs. The construction and the analysis (of standard properties) can be fully automated. This means that the modeller does not need any knowledge of the underlying mathematics. As an example, it is not necessary to know how it is checked whether two markings are symmetrical or not, or how the different proof rules work. Secondly, the occurrence graph contains all details about the behaviour of the CP-net – since it represents all possible occurrence sequences. Hence, it is possible to investigate all kinds of dynamic properties by means of occurrence graphs (with the exception of those properties which deal with concurrency). The main drawback of occurrence graph analysis is the fact that the occurrence graphs become very large. Even a small CP-net may have an occurrence graph which is intractable. The use of symmetries (and other reduction methods) improves the situation, but it does not remove the problem. With our present knowledge (and technology) we cannot hope to verify large systems by means of occurrence graphs. However, we can use occurrence graphs on selected subnets. This is a very effective way to find errors. A small mistake will often imply that we do not get, e.g., an expected marking bound or an expected home marking. It is also possible to investigate more complex CP-nets – by means of partial occurrence graphs, where we only develop, e.g., a fixed number of outgoing arcs for each node. Such a method will very often catch errors – although it cannot count as a full proof of the desired system properties. A partial occurrence graph corresponds to making a large number of simulation runs – the graph represents the results in a systematic way.

One problem with occurrence graphs is the fact that it is necessary to fix all system parameters (e.g., the number of managers in the data base system) before an occurrence graph can be constructed. This means that we always find properties which are specific to the chosen values of the system parameters. In practice the problem isn't that big. When we understand how a data base system behaves for a few managers, we also know a lot about how it behaves when there are more managers. This is of course only true when we talk about the logical correctness of a system, and not when we speak about the performance.

7 Place Invariants

The basic idea behind place invariants is to construct equations which are satisfied for all reachable markings. In the data base system we expect each manager to be either *Inactive*, *Waiting* or *Performing*. This is expressed by the following equation satisfied for all reachable markings M:

M(Inactive) + M(Waiting) + M(Performing) = DBM.

Analogously, we expect each message to be either *Unused*, *Sent*, *Received* or *Acknowledged* and we expect the system to be either *Active* or *Passive:*

M(Unused) + M(Sent) + M(Received) + M(Acknowledged) = MES

M(Active) + M(Passive) = 1`e.

These three equations are examples of place invariants. Each of them states that a certain set of places has – together – an invariant multi-set of tokens, i.e., a multi-set of tokens which is the same for all reachable markings.

It is possible to modify the tokens of some of the involved places, before we make the multi-set addition. This is illustrated by the following place invariant, where we use the function *Rec* to map each message (s,r) into the receiver r:

M(Inactive) + M(Waiting) + Rec(M(Received)) = DBM.

Without the function *Rec* it would have been impossible to add the three multi-sets – because two of them are over DBM while the last one is over MES. Each of the above equations can be written on the form:

$$W_{p_1}(M(p_1)) + W_{p_2}(M(p_2)) + \ldots + W_{p_n}(M(p_n)) = m_{inv}$$

where $\{p_1, p_2, \ldots, p_n\} \subseteq P$. Each **weight** W_p is a function mapping from the type of p into some common type $A \in \Sigma$ shared by all weights. Finally m_{inv} is a multi-set. It can be determined by evaluating the left-hand side of the equation in the initial marking (or in any other reachable marking).

As illustrated by the following invariant there are situations in which we want some of the weights to be negative:

M(Performing) – Rec(M(Received)) = Ø.

There are also situations in which we want weights that map each token of p into several elements of type A – instead of just one. This is illustrated by the following invariant where the function *Mes* maps each data base manager into n – 1 tokens of type MES:

M(Sent) + M(Received) + M(Acknowledged) – Mes(M(Waiting)) = Ø.

To capture the two extensions of weights, described above, we introduce **weighted sets**. A weighted set is defined in exactly the same way as a multi-set – except that we replace $\mathbb{N}$ by $\mathbb{Z}$, i.e., allow coefficients to be negative. The operations on weighted sets are similar to the operations on multi-sets. For weighted sets it is always possible to perform subtraction and we can make scalar-multiplication with negative integers. However, it no longer makes sense

to talk about the size (because the sum of the coefficients may be divergent). The set of all weighted sets over A is denoted by A_{WS}. A formal definition of weighted sets and their operations can be found in [28]. It is a straightforward modification of Defs. 3.1 and 3.2.

Having introduced weighted sets, we demand each place $p \in P$ to have a weight $W_p \in [C(p) \rightarrow A_{WS}]$. To apply W_p to a weighted set $m \in C(p)_{WS}$ (or a multi-set $m \in C(p)_{MS}$), we apply W_p to each individual element. This gives us an extended function $\underline{W_p} \in [C(p)_{WS} \rightarrow A_{WS}]$ defined by:

$$\underline{W_p}(m) = \sum_{c \in C(p)} m(c) * W_p(c).$$

It is easy to show that $\underline{W_p}$ is **linear**, i.e., that it satisfies:

$$\underline{W_p}(m_1 + m_2) = \underline{W_p}(m_1) + \underline{W_p}(m_2)$$

for all weighted-sets $m_1, m_2 \in C(p)_{WS}$. Hence we say that $\underline{W_p}$ is the linear extension of W_p. It can be proved that there is a one-to-one correspondence between $[C(p) \rightarrow A_{WS}]$ and the linear functions in $[C(p)_{WS} \rightarrow A_{WS}]$. Hence, we do not need to distinguish between W_p and $\underline{W_p}$.

The intuition behind an invariant is the following. For each marking M, we use a set of weights $W = \{W_p\}_{p \in P}$ to calculate a weighted set called a **weighted sum**:

$$W(M) = \sum_{p \in P} W_p(M(p)).$$

The weighted sum is demanded to be independent of the marking, and hence we have $W(M) = W(M_0)$ for all $M \in [M_0\rangle$. It is usually the case that many of the weights are zero functions, i.e., map each weighted-set into Ø.

How do we check that a set of weights really determines an invariant? Unless the system is trivial, we do not want to calculate W(M) for all reachable markings. Hence we introduce **place flows**. A set of weights $W = \{W_p\}_{p \in P}$ is a **place flow** iff the following property is satisfied for all $(t,b) \in BE$:

$$\sum_{p \in P} W_p(E(p,t)<b>) = \sum_{p \in P} W_p(E(t,p)<b>).$$

The intuition behind a place flow is to check that each binding element (t,b) removes – when the weights are taken into account – a set of tokens that is identical to the set of tokens which is added. For each occurring step $M_1[t,b\rangle M_2$ we then have that the weighted sum $W(M_2)$ becomes identical to the weighted sum $W(M_1)$, because the removed tokens are counterbalanced by the added tokens.

Definition 7.1: Let $A \in \Sigma$ be a type and let $W = \{W_p\}_{p \in P}$ be a set of linear functions such that $W_p \in [C(p)_{WS} \rightarrow A_{WS}]$ for all $p \in P$.

(i) W is a **place flow** iff:

$$\forall (t,b) \in BE: \sum_{p \in P} W_p(E(p,t)<b>) = \sum_{p \in P} W_p(E(t,p)<b>).$$

(ii) W determines a **place invariant** iff:

$$\forall M \in [M_0\rangle: \sum_{p \in P} W_p(M(p)) = \sum_{p \in P} W_p(M_0(p)).$$

The following theorem is the heart of invariant analysis. It tells us that the static property in Def. 7.1 (i) is sufficient and necessary to guarantee the dynamic property in Def. 7.1 (ii).

Theorem 7.2: W is a place flow $\Leftrightarrow$ W determines a place invariant.

$\Rightarrow$ is satisfied for all CP-nets.
$\Leftarrow$ is only satisfied when the CP-net does not have dead binding elements.

Proof: Assume that W is a place flow. We first prove that $M_1[Y\rangle M_2$ implies $W(M_1) = W(M_2)$. From the occurrence rule in Def. 3.6 we have:

$$\forall p \in P: M_2(p) + \sum_{(t,b) \in Y} E(p,t)<b> = M_1(p) + \sum_{(t,b) \in Y} E(t,p)<b>$$

which implies:

$$\sum_{p \in P} W_p\Big(M_2(p) + \sum_{(t,b) \in Y} E(p,t)<b>\Big) = \sum_{p \in P} W_p\Big(M_1(p) + \sum_{(t,b) \in Y} E(t,p)<b>\Big).$$

From the linearity of the weight functions we get:

$$\sum_{p \in P} W_p(M_2(p)) + \sum_{p \in P} \sum_{(t,b) \in Y} W_p(E(p,t)<b>)$$

$$= \sum_{p \in P} W_p(M_1(p)) + \sum_{p \in P} \sum_{(t,b) \in Y} W_p(E(t,p)<b>).$$

From the flow property we have:

$$\forall (t,b) \in BE: \sum_{p \in P} W_p(E(p,t)<b>) = \sum_{p \in P} W_p(E(t,p)<b>)$$

which implies:

$$\sum_{(t,b) \in Y} \sum_{p \in P} W_p(E(p,t)<b>) = \sum_{(t,b) \in Y} \sum_{p \in P} W_p(E(t,p)<b>)$$

which we can rewrite to:

$$\sum_{p \in P} \sum_{(t,b) \in Y} W_p(E(p,t)<b>) = \sum_{p \in P} \sum_{(t,b) \in Y} W_p(E(t,p)<b>).$$

The two double sums in this equation are identical to those which we had above. Hence we conclude that:

$$\sum_{p \in P} W_p(M_2(p)) = \sum_{p \in P} W_p(M_1(p)),$$

i.e., that $W(M_2) = W(M_1)$.

Next let $M \in [M_0\rangle$ be a reachable marking and let σ be an occurrence sequence which starts in M_0 and ends in M. By applying our result above to each step $M_i[Y_i\rangle M_{i+1}$ of σ, we conclude that $W(M) = W(M_0)$. Hence we have proved $\Rightarrow$.

Next let us assume that W determines an invariant and that the CP-net has no dead binding elements. This means that each binding element (t,b) has at least one reachable marking M_1 in which it becomes enabled. Let M_2 be the marking determined by $M_1[t,b\rangle M_2$. By a sequence of arguments which is very similar to the ones we have made above, it can be seen that $W(M_2) = W(M_1)$ implies that (t,b) satisfies the property in Def. 7.1 (i). Hence, we have shown $\Leftarrow$. □

Above we have discussed how we can use flows to check whether a set of weights determines an invariant or not. Later in this section, we discuss how to find suitable weights, i.e., how to construct invariants. However, first we illustrate how we can use invariants – to prove dynamic properties of the CP-net.

The data base system has the invariants shown below (plus many more). For brevity, and improved readability, we omit M(). This means that we write $W_p(p)$ instead of $W_p(M(p))$. The function *Ign* (for ignore) maps each token (of any type) into $e \in E$.

PI_{DBM}	Inactive + Waiting + Performing = DBM
PI_{MES}	Unused + Sent + Received + Acknowledged = MES
PI_E	Active + Passive = E
PI_{PER}	Performing = Rec(Received)
PI_{WA}	Mes(Waiting) = Sent + Received + Acknowledged
PI_{AC}	Active = Ign(Waiting)

Let us first show that the invariants can be used to prove the integer and multi-set bounds postulated in Sect. 4.

All the multi-bounds follow from PI_{DBM}, PI_{MES} and PI_E, respectively. The same is true for the integer bounds for *Inactive*, *Unused*, *Passive* and *Active*. The integer bound for *Waiting* follows from PI_{AC} since we have already shown that $|M(\text{Active})| \leq 1$. The integer bound for *Sent*, *Received* and *Acknowledged* follows from PI_{WA}, since we know that $|M(\text{Waiting})| \leq 1$ and hence that $|\text{Mes}(M(\text{Waiting}))| \leq n-1$. The integer bound for *Performing* follows from PI_{PER}, since we know that $|M(\text{Received})| \leq n-1$. It is straightforward to construct occurrence sequences which show that all the bounds are optimal, i.e., the smallest possible. We omit this part of the proof.

Next let us show that the data base system cannot reach a dead marking. *The proof is by contradiction:* Let us assume that we have a reachable marking M which is dead, i.e., has no enabled transitions. From PI_{DBM} we know that M has n tokens of type DBM, distributed on the places *Inactive*, *Performing* and *Waiting*. Now let us investigate where these tokens can be positioned.

Let us first assume that at least one data base manager, s, is *Waiting*. From PI_{AC} it follows that there is exactly one *Waiting* manager and it also follows that the system is *Active*. From PI_{DBM}, we know that the remaining $n-1$ managers are *Inactive* or *Performing*. From PI_{WA} we know that the messages *Mes(s)* are either *Sent*, *Received* or *Acknowledged*, and we also know that no other messages are in these states. From PI_{PER} it then follows that each manager $r \in DBM - \{s\}$ is *Performing* iff the message (s,r) is *Received* and that r is *Inactive* iff the message (s,r) is either *Sent* or *Acknowledged*. A message (s,r) on *Sent* would imply that *RM* is enabled (since s is *Inactive*). A message (s,r) on *Received* would imply that *SA* is enabled (since s is *Performing*). Hence, we conclude that all messages in *Mes(s)* must be *Acknowledged*. However, this implies that *RA* is enabled (since s is *Waiting* and the system is *Active*). Hence, we have a contradiction (with the assumption that M is dead).

Next, let us assume that no data base manager is *Waiting*. From the invariants it is then easy to prove that M is identical to the initial marking in which *SM* is enabled. Hence, we have a contradiction (with the assumption that M is dead).

From the proof above, we know that any message on *Sent* can be moved to *Received*, and that any message on *Received* can be moved to *Acknowledged*. This means that, from any marking $M \in [M_0\rangle$, we can reach a marking in which *Sent*, *Received* and *Performing* have no tokens. From the invariants it then follows that we are either in the initial marking M_0 or in a marking from which M_0 can be reached by an occurrence of *RA*. Hence, we have shown that M_0 is a home marking.

It is easy to construct an occurrence sequence which starts in M_0 and contains all binding elements. Since M_0 is a home marking, we then conclude that all four transitions are strictly live.

Now let us discuss how we can find flows. There are two possible approaches. The first approach tries to make a fully automatic calculation of the flows – starting from scratch, i.e., without using any of the knowledge which the modeller has built up during the construction of the system. It is possible to make such a calculation. A CP-net can be represented as a matrix, I, with a row for each place and a column for each transition. Each matrix element I(p,t) is a function. It maps each binding $b \in B(t)$ into the weighted set of tokens E(t,p)<b> – E(p,t)<b> (which describe how the marking of p is changed when t occurs with the binding b). For each occurring step $M_1[Y\rangle M_2$, we then have the following matrix equation:

$$M_2 = M_1 + I * Y$$

where Y is the column vector with the elements $\{Y(t) \mid t \in T\}$, while M_1 and M_2 are the column vectors with the elements $\{M_1(p) \mid p \in P\}$ and $\{M_2(p) \mid p \in P\}$, respectively. The matrix product is defined in the usual way, i.e.:

$$(I * Y)(p) = \sum_{t \in T} I(p,t) * Y(t).$$

However, we no longer multiply integers, instead we apply the function I(p,t) to the multi-set Y(t). From mathematics, it is well-known that matrices and matrix multiplication can be generalised in this way, and it can be proved that all the

usual rules for matrices still apply (provided that the new multiplication operation satisfies certain basic requirements, e.g., associativity). It can also be proved that a set of weights is a flow iff the row vector W with the elements $\{W_p \mid p \in P\}$ is a solution to the following homogeneous matrix equation – where multiplication is defined as composition of the functions W(p) and I(p,t) while 0 denotes a matrix vector in which all elements are zero functions:

$$W * I = 0.$$

The equation can be solved by means of Gauss-elimination. However, there are a couple of problems. The first problem is the fact that we may have matrix elements for which the inverse function does not exist (intuitively this means that we cannot divide matrix elements by each other). This problem can be circumvented, but the algorithms become rather complex. The second problem is more profound. A CP-net usually has infinitely many flows – because any linear combination of flows is itself a flow. It is possible to find a basis for the flows, but it is not at all easy to do this in such a way that the flows in the basis determine the invariants which are useful to prove dynamic properties. Hence, we are left with the problem of finding those linear combinations which are useful to us, and this problem is often as difficult as finding the basis. The representation of CP-nets as matrices and the calculation of flows via matrix equations are described in [12], [23], [25] and [33].

The second approach to calculation of flows does not try to find the flows from scratch. Instead it builds on the idea that the modeller constructs some sets of weights which he expects to determine invariants. Such potential invariants may be derived, e.g., from the system specification and from the modeller's knowledge of the expected system properties. The invariants may be specified during the analysis of the CP-net. However, it is much more useful (and also easier) to specify the invariants while the CP-net is being created. This means that we construct the invariants as an integrated part of the design (in a similar way as a good programmer specifies a loop invariant at the moment he creates the loop). In programming, it would be most unusual (and unrealistic), first to write a large program (without even thinking about invariants) and then expect an automatic construction of useful invariants which can be easily interpreted by the programmer.

The proposed weights are checked by means of the property in Def. 7.1 (i). This means that the check is constructive, in the sense that it indicates exactly where the problems are – by identifying the transitions (or even the binding elements) that violate the flow property. Hence, it is usually rather straightforward to figure out how to modify the CP-net or the weights so that we obtain valid flows.

It is not trivial to check flows – although it is much easier than to find them from scratch. When the set of bindings is small we can simply test each binding in turn. However, when a transition t has a large set of possible bindings, or even an infinite set, we need a more subtle method. We want to show that the function which maps each binding $b \in B(t)$ into the weighted set:

$$\sum_{p \in P} W_p(E(t,p)<b>) - \sum_{p \in P} W_p(E(p,t)<b>)$$

is the zero-function, i.e., that it maps all b into Ø. The invariant analysis tool described in [9] verifies this by using the lambda expression of the function. The lambda expression is available because the arc expressions are specified in the functional programming language Standard ML (which compiles into lambda expressions). By means of lambda reductions it is possible to verify that the function is the zero function. Arc expressions may be arbitrarily complex, and hence there may exist transitions where the reduction is impossible. However, the tool in [9] is able to cope with most of the arc expressions which are found in "standard" CP-nets.

Above, we have advocated that the modeller should be responsible for the construction of potential invariants. However, there are several ways in which this process can be supported by tools. Hence, it is more adequate to think about the construction as semi-automatic than to think of it as purely manual. We have already discussed how the construction of flows can be supported by an automatic check. In addition to this it is also possible, under certain circumstances, to deduce some weights – when other weights are known. As an example, let us assume that the modeller is looking for an invariant which relates the marking of *Inactive*, *Waiting* and *Performing*. If the modeller specifies that all other places have weights which are zero-functions, it is possible to prove that *Inactive*, *Waiting* and *Performing* must have identical weights. The proof is by contradiction: Let us assume that there exists, e.g., a manager d such that $W_{Inac}(d) \neq W_{Perf}(d)$. It is then easy to see that the binding elements of the form (RM,<s = d, r = …>) violate the flow property in Def. 7.1 (i). More information about a method to relate weights to each other can be found in [9] and [25]. The basic idea behind the method is to perform a reduction of the CP-net – without changing the set of place invariants.

In general, we envision an interactive system in which the user may specify the weights of certain places, e.g., to be zero functions. Based on this information the system calculates a number of derived weights – or tells us that the specified weights are inconsistent with each other. The situation can be compared to a spreadsheet model. Whenever the user changes a number in the spreadsheet, all the corresponding equations are recalculated, and the new results are shown. Whenever the user changes a weight in the CP-net, all the corresponding dependencies are recalculated, and the new derived weights are shown.

It is also desirable to develop tool support for the use of invariants, i.e., to help the modeller to prove dynamic properties. As an example, it is possible to implement algorithms which use the invariants to derive bounds for the marking of certain places – from a specified marking of other places. With such a system it would be much easier and less error prone to perform the kind of arguments which we made to prove that the data base system is deadlock free. The main development problem for such a system is the design of a suitable user interface.

Transition invariants and transition flows are the duals of place invariants and place flows. This means that we attach a weight to each transition. Intuitively, a transition flow determines a set of occurrence sequences that have no total effect,

i.e., have the same start and end marking. Transition flows can be calculated in a way which is similar to that of place flows – but it is also in this case easier and more useful to construct the invariants during the creation of the CP-net. Transition invariants are used for similar purposes as place invariants, i.e., to investigate the dynamic properties of CP-nets.

Analysis by means of place and transition invariants has several attractive properties. First of all, it is possible to obtain an invariant for a hierarchical CP-net by composing invariants of the individual pages. This means that it is possible to use invariants for large systems – without encountering the same kind of complexity problems as we have for occurrence graphs. Secondly, we can find invariants without fixing the system parameters, such as the number of data base managers. Hence we can obtain general properties which are independent of the system parameters. Thirdly, we can construct the invariants during the design of a system and this will usually lead to an improved design and also an improved understanding of the system. The main drawback of invariant analysis is the fact that it requires skills which are considerably higher (and more mathematical) than those required by the other analysis methods. This means that it is more difficult to use invariants in industrial system development.

8 Historical Remarks

The foundation of Petri nets was presented by Carl Adam Petri in his doctoral thesis [38]. The first nets were called Condition/Event Nets (CE-nets). This net model allows each place to contain at most one token. The place is considered to represent a boolean condition, which can be either true or false. In the following years a large number of people 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) allowing a place to contain several tokens.

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) was proposed in [41] and [44]. The basic ideas of this net model are very close to those of CE-nets – but EN-nets avoid some of the technical problems which 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 real-world applications in a manageable way. 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 almost 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 [18]. PrT-nets were the first kind of high-level nets which were constructed without any particular application area in mind. PrT-nets form a nice generalisation 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 generalise 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 was presented in [19]. This definition draws heavily on sigma algebras (as known from the theory of abstract data types).

However, it soon became apparent that PrT-nets present some technical problems when the analysis methods of place invariants and transition invariants are generalised. 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. To overcome this problem the first version of Coloured Petri Nets (CP^{81}-nets) was defined in [23]. The main ideas of this net model are directly inspired by PrT-nets, but the relation between a binding element and the data values of the tokens involved in the occurrence is now defined by functions and not by expressions as in PrT-nets. This means that invariants can 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 relationship 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 [24] where the nets were 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 change the name from HL-nets to Coloured Petri Nets (CP-nets).

CP-nets have two different representations. The *expression representation* uses arc expressions and guards, while the *function representation* uses linear functions between multi-sets. There are formal translations between the two representations (in both directions). The expression representation is nearly identical to PrT-nets (as presented in [18]), while the function representation is nearly identical to CP^{81}-nets. The first coherent presentations of CP-nets and their analysis methods were given in [25] and [26].

Today most of the practical applications of Petri nets (reported in the literature) use either PrT-nets or CP-nets. 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 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.

Several other classes of high-level nets have been defined. Most of these are quite similar to CP-nets, but use different inscription languages (e.g., building on algebraic specifications or object oriented languages). For more information, see the bibliographical remarks to Chap. 1 in [27]. Some of the most important papers on high-level nets, their analysis methods and applications have been reprinted in [30].

9 Conclusion

This paper has presented an overview of the theoretical aspects of CP-nets. After an informal introduction, we presented the formal definition of CP-net models and the formal definition of their behaviour. Then we defined a number of dynamic properties and finally we discussed how CP-nets can be analysed, to investigate their dynamic properties. Due to space limitations for this paper, we have only been able to give the most important definitions and to state the most important propositions and theorems (without proofs). The proofs and a large number of additional details can be found in [27] and [28].

The development of CP-nets has been driven by the desire to develop a modelling language – at the same time theoretically well-founded and versatile enough to be used in practice for systems of the size and complexity that we find in typical industrial projects. To be successful it has been mandatory to support the modelling and analysis activities by a powerful set of computer tools available on different hardware and software platforms. It has also been necessary to develop hierarchical CP-nets, i.e., techniques which allow a model to be composed of a set of submodels with well-defined interfaces and well-defined interaction. Hierarchical CP-nets are defined in Chap. 3 of [27]. The dynamic properties and the analysis methods presented in Sects. 5–7 (of the present paper) can easily be extended to cope with hierarchical nets. Details can be found in [27] and [28] which define all concepts directly for hierarchical nets and treat non-hierarchical nets as a special case (where the model consists of only one module).

The space limitations for this paper have not made it possible to present examples of the industrial use of CP-nets. Neither has it been possible to give a comprehensive description of the tool support. Reports on a number of case studies can be found in [1], [2], [3], [4], [5], [8], [11], [17], [20], [22], [27], [32], [36], [40], [42] and [43]. Nearly all the case studies have been conducted in an in-

dustrial setting. Many of them have used the Design/CPN tool described in [13] and [27].

Acknowledgements

Many students and colleagues – in particular at Aarhus University and Meta Software – have influenced the development of CP-nets, their analysis methods and their tool support. The development has been supported by several grants from the Danish Natural Science Research Council. A more detailed description of individual contributions can be found in the preface of [27].

References

[1] G. Balbo, S.C. Bruell, P. Chen, G. Chiola: *An Example of Modelling and Evaluation of a Concurrent Program Using Colored Stochastic Petri Nets: Lamport's Fast Mutual Exclusion Algorithm.* IEEE Transactions on Parallel and Distributed Systems, 3 (1992). Also in [30], 533–559.

[2] J. Berger, L. Lamontagne: *A Colored Petri Net Model for a Naval Command and Control System.* In: M. Ajmone-Marsan (ed.): Application and Theory of Petri Nets 1993. Proceedings of the 14th International Petri Net Conference, Chicago 1993, Lecture Notes in Computer Science Vol. 691, Springer-Verlag 1993, 532–541.

[3] C. Capellmann, H. Dibold: *Petri Net Based Specifications of Services in an Intelligent Network. Experiences Gained from a Test Case Application.* In: M. Ajmone-Marsan (ed.): Application and Theory of Petri Nets 1993. Proceedings of the 14th International Petri Net Conference, Chicago 1993, Lecture Notes in Computer Science Vol. 691, Springer-Verlag 1993, 542–551.

[4] L. Cherkasova, V. Kotov, T. Rokicki: *On Net Modelling of Industrial Size Concurrent Systems*. In: M. Ajmone-Marsan (ed.): Application and Theory of Petri Nets 1993. Proceedings of the 14th International Petri Net Conference, Chicago 1993, Lecture Notes in Computer Science Vol. 691, Springer-Verlag 1993, 552–561.

[5] L. Cherkasova, V. Kotov, T. Rokicki: *On Scalable Net Modeling of OLTP*. In *PNPM93: Petri Nets and Performance Models.* Proceedings of the 5th International Workshop, Toulouse, France 1993, IEEE Computer Society Press, 270-279.

[6] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad: *On Well-Formed Coloured Nets and Their Symbolic Reachability Graph.* In [30], 373–396.

[7] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad: *A Symbolic Reachability Graph for Coloured Petri Nets.* Submitted to Theoretical Computer Science.

[8] S. Christensen, L.O. Jepsen: *Modelling and Simulation of a Network Management System Using Hierarchical Coloured Petri Nets.* In: E. Mosekilde (ed.): Modelling and Simulation 1991. Proceedings of the 1991 European Simulation Multiconference, Copenhagen, 1991, Society for Computer Simulation 1991, 47–52.

[9] S. Christensen, J. Toksvig: *Tool Support for Place Flow Analysis of Hierarchical CP-nets.* Computer Science Department, Aarhus University, Denmark, Version 2.0.1, 1993.

[10] E.M. Clarke, T. Filkorn, S. Jha: *Exploiting Symmetry in Temporal Logic Model Checking.* In: C. Courcoubetis (ed.): Computer Aided Verification. Proceedings of the 5th International Conference on Computer Aided Verification, Elounda, Greece, 1993, Lecture Notes in Computer Science Vol. 697, Springer-Verlag 1993, 450–462.

[11] H. Clausen, P.R. Jensen: *Validation and Performance Analysis of Network Algorithms by Coloured Petri Nets.* In *PNPM93: Petri Nets and Performance Models.* Proceedings of the 5th International Workshop, Toulouse, France 1993, IEEE Computer Society Press, 280-289.

[12] J.M. Couvreur, J. Martínez: *Linear Invariants in Commutative High Level Nets.* In: G. Rozenberg (ed.): Advances in Petri Nets 1990, Lecture Notes in Computer Science Vol. 483, Springer-Verlag 1991, 146–165. Also in [30], 284 –302.

[13] *Design/CPN. Reference Manual.* Meta Software Corporation, 125 Cambridge Park Drive, Cambridge MA 02140, USA, Version 2.0, 1993.

[14] *Design/CPN Occurrence Graph Analyzer.* Meta Software Corporation, 125 Cambridge Park Drive, Cambridge MA 02140, USA, Version 0.35, 1993.

[15] E.A. Emerson, A.P. Sistla: *Symmetry and Model Checking.* In: C. Courcoubetis (ed.): Computer Aided Verification. Proceedings of the 5th International Conference on Computer Aided Verification, Elounda, Greece, 1993, Lecture Notes in Computer Science Vol. 697, Springer-Verlag 1993, 463–477.

[16] 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.

[17] 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.

[18] H.J. Genrich, K. Lautenbach: *System Modelling with High-level Petri Nets.* Theoretical Computer Science 13 (1981), North-Holland, 109–136.

[19] H.J. Genrich: *Predicate/Transition Nets.* In: W. Brauer, W. Reisig, 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. Also in [30], 3– 43.

[20] H.J. Genrich, R.M. Shapiro: *Formal Verification of an Arbiter Cascade.* In: K. Jensen (ed.): Application and Theory of Petri Nets 1992. Proceedings of the 13th International Petri Net Conference, Sheffield 1992, Lecture Notes in Computer Science Vol. 616, Springer-Verlag 1992, 205–223.

[21] P. Huber, A.M. Jensen, L.O. Jepsen, K. Jensen: *Reachability Trees for High-level Petri Nets.* Theoretical Computer Science 45 (1986), North-Holland, 261–292. Also in [30], 319–350.

[22] P. Huber, V.O. Pinci: *A Formal Executable Specification of the ISDN Basic Rate Interface.* Proceedings of the 12th International Conference on Application and Theory of Petri Nets, Aarhus 1991, 1–21.

[23] K. Jensen: *Coloured Petri Nets and the Invariant Method.* Theoretical Computer Science 14 (1981), North-Holland, 317–336.

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

[25] K. Jensen: *Coloured Petri Nets.* In: W. Brauer, W. Reisig, 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.

[26] K. Jensen: *Coloured Petri Nets: A High-level Language for System Design and Analysis.* In: G. Rozenberg (ed.): Advances in Petri Nets 1990, Lecture Notes in Computer Science Vol. 483, Springer-Verlag 1991, 342–416. Also in [30], 44–122.

[27] K. Jensen: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* Vol. 1: Basic Concepts. EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1992.

[28] K. Jensen: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* Vol. 2: Analysis Methods. EATCS Monographs on Theoretical Computer Science, Springer-Verlag. To appear in 1994.

[29] K. Jensen: *Coloured Petri Nets with Time Stamps.* Computer Science Department, Aarhus University, Denmark, 1993.

[30] K. Jensen, G. Rozenberg (eds.): *High-level Petri Nets. Theory and Application.* Springer-Verlag, 1991.

[31] R.M. Karp, R.E. Miller: *Parallel Program Schemata.* Journal of Computer and System Sciences, Vol. 3, 1969, 147–195.

[32] W.W. McLendon, R.F. Vidale: *Analysis of an Ada System Using Coloured Petri Nets and Occurrence Graphs.* In: K. Jensen (ed.): Application and Theory of Petri Nets 1992. Proceedings of the 13th International Petri Net Conference, Sheffield 1992, Lecture Notes in Computer Science Vol. 616, Springer-Verlag 1992, 384–388.

[33] G. Memmi, J. Vautherin: *Analysing Nets by the Invariant Method.* In: W. Brauer, W. Reisig, 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. Also in [30], 247–283.

[34] R. Milner, R. Harper, M. Tofte: *The Definition of Standard ML*. MIT Press, 1990.

[35] R. Milner, M. Tofte: *Commentary on Standard ML*. MIT Press, 1991.

[36] K.H. Mortensen, V. Pinci: *Modelling the Work Flow of a Nuclear Waste Management Program*. Computer Science Department, Aarhus University, Denmark, 1993.

[37] L. Paulson: *ML for the Working Programmer*. Cambridge University Press, 1991.

[38] 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, Griffiths Air Force Base, New York, Vol. 1, Suppl. 1, 1966.

[39] L. Petrucci: *Combining Finkel's and Jensen's Reduction Techniques to Build Covering Trees for Coloured Nets*. Petri Net Newsletter no. 36 (August 1990), Special Interest Group on Petri Nets and Related System Models, Gesellschaft für Informatik (GI), Germany, 1990, 32–36.

[40] V.O. Pinci, R.M. Shapiro: *An Integrated Software Development Methodology Based on Hierarchical Colored Petri Nets*. In: G. Rozenberg (ed.): Advances in Petri Nets 1991, Lecture Notes in Computer Science Vol. 524, Springer-Verlag 1991, 227–252. Also in [30], 649–667.

[41] G. Rozenberg: *Behaviour of Elementary Net Systems*. In: W. Brauer, W. Reisig, 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.

[42] R.M. Shapiro: *Validation of a VLSI Chip Using Hierarchical Coloured Petri Nets*. Journal of Microelectronics and Reliability, Special Issue on Petri Nets, Pergamon Press, 1991. Also in [30], 667–687.

[43] R.M. Shapiro, V.O. Pinci, R. Mameli: *Modelling a NORAD Command Post Using SADT and Coloured Petri Nets*. Proceedings of the IDEF Users Group, Washington DC, May 1990.

[44] P.S. Thiagarajan: *Elementary Net Systems*. In: W. Brauer, W. Reisig, 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.

[45] A. Valmari: *Stubborn Sets for Reduced State Space Generation*. In: G. Rozenberg (ed.): Advances in Petri Nets 1990, Lecture Notes in Computer Science Vol. 483, Springer-Verlag 1991, 491–515.

[46] A. Valmari: *Stubborn Sets of Coloured Petri Nets*. Proceedings of the 12th International Conference on Application and Theory of Petri Nets, Aarhus 1991, 102–121.