Interval timed coloured Petri nets and their analysis

Eindhoven University of Technology

Department of Mathematics and Computing Science

Interval timed coloured Petri nets and their analysis

by

W.M.P. van der Aalst

92/17

Computing Science Note 92/17
Eindhoven, September 1992

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. F. van Neerven
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

Interval timed coloured Petri nets and their analysis

W.M.P. van der Aalst*
Eindhoven University of Technology
Dept. of Mathematics and Computing Science

Abstract

Practical experiences show that only *timed* and *coloured* Petri nets are capable of modelling large and complex real-time systems. This is the reason we present the *Interval Timed Coloured Petri Net* (ITCPN) model. An interval timed coloured Petri net is a coloured Petri net extended with time; time is in tokens and transitions determine a delay for each produced token. This delay is specified by an upper and lower bound, i.e. an interval. The ITCPN model allows the modelling of the dynamic behaviour of large and complex systems, without losing the possibility of formal analysis. In addition to the existing analysis techniques for coloured Petri nets, we propose a new analysis method to analyse the temporal behaviour of the net. This method constructs a reduced reachability graph and exploits the fact that delays are described by an interval.

1 Introduction

Petri nets have been widely used for the modelling and analysis of concurrent systems (Reisig [22]). There are several factors which contribute to their success: the graphical nature, the ability to model parallel and distributed processes in a natural manner, the simplicity of the model and the firm mathematical foundation. Nevertheless, the basic Petri net model is not suitable for the modelling of many systems encountered in logistics, production, communication, flexible manufacturing and information processing. Petri nets describing real systems tend to be complex and extremely large. Sometimes it is even impossible to model the behaviour of the system accurately. To solve these problems many authors propose extensions of the basic Petri net model.

Several authors have extended the basic Petri net model with *coloured* or *typed tokens* ([8], [12], [13], [9], [10]). In these models tokens have a value, often referred to as 'colour'. There are several reasons for such an extension. One of these reasons is the

*This research is supported by IPL-TUE/TNO

fact that (uncoloured) Petri nets tend to become too large to handle. Another reason is the fact that tokens often represent objects or resources in the modelled system. As such, these objects may have attributes, which are not easily represented by a simple Petri net token. These 'coloured' Petri nets allow the modeller to make much more succinct and manageable descriptions, therefore they are called *high-level nets.*

Other authors have proposed a Petri net model with explicit quantitative time (e.g. [25], [21], [17], [16], [9], [23]). We call these models *timed Petri net* models.

In our opinion, only timed *and* coloured Petri nets are suitable for the modelling of large and complex real-time systems. Although there seems to be a consensus of opinion on this matter, only a few timed coloured Petri net models have been proposed in literature (e.g. Van Hee et al. [9], Morasca [19]). Moreover, even fewer methods have been developed for the analysis of the temporal behaviour of these nets. This is one of the reasons we propose the *Interval Timed Coloured Petri Net* (ITCPN) model and an analysis method, called *MTSRT*, based on this model.
The ITCPN model uses a rather new timing mechanism where time is associated with tokens. This timing concept has been adopted from Van Hee et al. ([9]). In the ITCPN model we attach a *timestamp* to every token. This timestamp indicates the time a token becomes available. Associating time with tokens seems to be the natural choice for high-level Petri nets, since the colour is also associated with tokens. The *enabling time* of a transition is the maximum timestamp of the tokens to be consumed. Transitions are *eager* to fire (i.e. they fire as soon as possible), therefore the transition with the smallest enabling time will fire first. Firing is an atomic action, thereby producing tokens with a timestamp of at least the firing time. The difference between the firing time and the timestamp of such a produced token is called the *firing delay.* The (firing) delay of a produced token is specified by an *upper* and *lower bound*, i.e. an *interval.*

Instead of using 'interval timing', we could have used a Petri net model with fixed delays or stochastic delays.
Petri nets with fixed (deterministic) delays have been proposed in [25], [21], [23] and [9]. They allow for simple analysis methods but are not very expressive, because in a real system the durations of most activities are variable.
One way to model this variability, is to assume certain delay distributions, i.e. to use a timed Petri net model with delays described by probability distributions. These nets are called *stochastic Petri nets* ([7], [16], [15]). Analysis of stochastic Petri nets is possible (in theory), since the reachability graph can be regarded, under certain conditions, as a Markov chain or a semi-Markov process. However, these conditions are severe: all firing delays have to be sampled from an exponential distribution or the topology of the net has to be of a special form (Ajmone Marsan et al. [15]). Since there are no general applicable analysis methods, several authors resorted to using simulation to study the behaviour of the net (see section 4).
To avoid these problems, we propose delays described by an *interval* specifying an

upper and lower bound for the duration of the corresponding activity. On the one hand, interval delays allow for the modelling of variable delays, on the other hand, it is not necessary to determine some artificial delay distribution (as opposed to stochastic delays). Instead, we have to specify bounds. These bounds can be used to verify time constraints. This is very important when modelling time-critical systems, i.e. *real-time* systems with 'hard' deadlines. These hard (real-time) deadlines have to be met for a safe operation of the system. An acceptable behaviour of the system depends not only on the logical correctness of the results, but also on the time at which the results are produced. Examples of such systems are: real-time computer systems, process controllers, communication systems, flexible manufacturing systems and just-in-time manufacturing systems.

To our knowledge, only one other model has been presented in literature which also uses delays specified by an interval. This model was presented by Merlin in [17, 18]. In this model the enabling time of a transition is specified by a minimal and a maximal time. Another difference with our model is the fact that Merlin's model is not a high-level Petri net model because of the absence of typed (coloured) tokens. Compared to our model, Merlin's model has a rather complex formal semantics, which was presented in [5] by Berthomieu and Diaz. This is caused by a redundant state space (marking and enabled transitions are represented separately) and the fact that they use a relative time scale and allow for multiple enabledness of transitions. An additional advantage of our approach is the fact that our semantics closely correspond to the intuitive interpretation of the dynamical behaviour of a timed Petri net.

The main purpose of this paper is to present a high-level Petri net model extended with interval timing which allows for new methods of analysis. In section 2 we introduce the ITCPN model. The formal definition and semantics are given in section 3. Section 4 deals with the analysis of interval timed coloured Petri nets. In this section, we introduce a new and powerful analysis method.

2 Interval timed coloured Petri nets

We use an example to introduce the notion of interval timed coloured Petri nets. Figure 1 shows an ITCPN composed of four **places** ($p_{in}, p_{busy}, p_{free}$ and p_{out}) and two **transitions** (t_1 and t_2). At any moment, a place contains zero or more **tokens**, drawn as black dots. In the ITCPN model, a token has three attributes: a position, a value and a timestamp, i.e. we can use the tuple $\langle\langle p, v\rangle, x\rangle$ to denote a token in place p with value v and timestamp x. The value of a token is often referred to as the **token colour**. Each place has a **colour set** attached to it which specifies the set of allowed values, i.e. each token residing in place p must have a colour (value) which is a member of the colour set of p.
The ITCPN shown in figure 1 represents a jobshop, jobs arrive via place p_{in} and leave the system via place p_{out}. The jobshop is composed of a number of machines. Each machine is represented by a token which is either in place p_{free} or in place p_{busy}. There

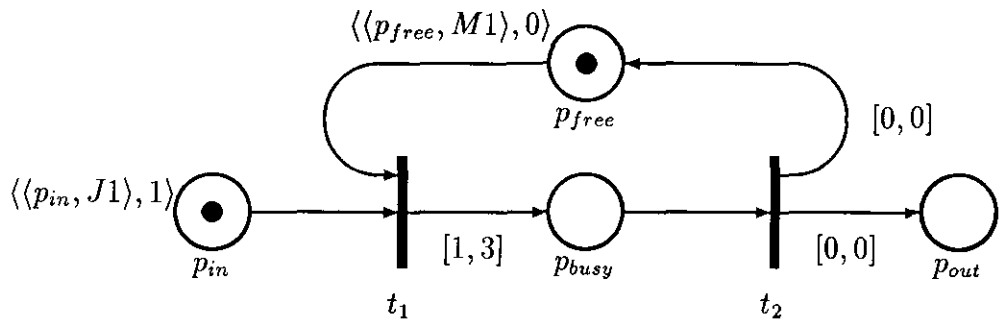

Figure 1: An interval timed coloured Petri net

are three colour sets $\mathcal{M} = \{M1, M2, ..\}$ and $\mathcal{J} = \{J1, J2, ..\}$ and $\mathcal{M} \times \mathcal{J}$. Colour set $\mathcal{J}$ (job types) is attached to place p_{in} and place p_{out}, colour set $\mathcal{M}$ (machine types) is attached to place p_{free}. Colour set $\mathcal{M} \times \mathcal{J}$ is attached to place p_{busy}.
Places and transitions are interconnected by **arcs**. Each arc connects a place and a transition in precisely one direction. Transition t_1 has two input places (p_{in} and p_{free}) and one output place (p_{busy}). Transition t_2 has one input place (p_{busy}) and two output places (p_{free} and p_{out}).
Places are passive components, while transitions are the active components. Transitions cause state changes. A transition is called **enabled** if there are 'enough' tokens on each of its input places. In other words, a transition is enabled if all input places contain (at least) the specified number of tokens (further details will be given later). An enabled transition may **occur** (**fire**) at time x if all the tokens to be consumed have a timestamp not later than time x. The **enabling time** of a transition is the maximum timestamp of the tokens to be consumed. Because transitions are eager to fire, a transition with the smallest enabling time will fire first.
Firing a transition means consuming tokens from the input places and producing tokens on the output places. If, at any time, more than one transition is enabled, then any of the several enabled transitions may be 'the next' to fire. This leads to a non-deterministic choice if several transitions have the same enabling time.
Firing is an atomic action, thereby producing tokens with a timestamp of at least the firing time. The difference between the firing time and the timestamp of such a produced token is called the **firing delay**. This delay is specified by an **interval**, i.e. only delays between a given upper bound and a given lower bound are allowed. In other words, the delay of a token is 'sampled' from the corresponding delay interval. Note that the term 'sampled' may be confusing, because the modeller does not specify a probability distribution, merely an upper and lower bound.
Moreover, it is possible that the modeller specifies a delay interval which is too wide, because of a lack of detailed information. In this case, the actual delays (in the real system) only range over a part of the delay interval.
The number of tokens produced by the firing of a transition may depend upon the values of the consumed tokens. Moreover, the values and delays of the produced tokens

may also depend upon the values of the consumed tokens. The relation between the *multi-set* of consumed tokens and the *multi-set* of produced tokens is described by the **transition function.** Function $F(t_1)$ specifies transition t_1 in the net shown in figure 1:

$$dom(F(t_1)) \;=\; \{\langle p_{in}, j\rangle + \langle p_{free}, m\rangle \mid j \in \mathcal{J} \;\text{ and }\; m \in \mathcal{M}\}$$

For $j \in \mathcal{J}$ and $m \in \mathcal{M}$, we have:[1]

$$F(t_1)(\langle p_{in}, j\rangle + \langle p_{free}, m\rangle) \;=\; \langle\langle p_{busy}, \langle m, j\rangle\rangle, [1,3]\rangle$$

The domain of $F(t_1)$ describes the condition on which transition t_1 is enabled, i.e. t_1 is enabled if there is (at least) one token in place p_{in} and one token in p_{free}. This means that transition t_1 may occur if there is a job waiting and one of the machines is free. Note that, in this case, the enabling of a transition does not depend upon the values of the tokens consumed. The enabling time of transition t_1 depends upon the timestamps of the tokens to be consumed. If t_1 occurs, it consumes one token from place p_{in} and one token from p_{free} and it produces one token for place p_{busy}. The colour of the produced token is a pair $\langle m, j\rangle$, where m represents the machine and j represents the job. The delay of this token is an arbitrary value between 1 and 3, e.g. 2.55 or 4/3.
Transition t_2 is specified as follows:

$$dom(F(t_2)) \;=\; \{\langle p_{busy}, \langle m, j\rangle\rangle \mid j \in \mathcal{J} \;\text{ and }\; m \in \mathcal{M}\}$$

For $j \in \mathcal{J}$ and $m \in \mathcal{M}$, we have:

$$F(t_2)(\langle p_{busy}, \langle m, j\rangle\rangle) \;=\; \langle\langle p_{free}, m\rangle, [0,0]\rangle + \langle\langle p_{out}, j\rangle, [0,0]\rangle$$

Transition t_2 represents the completion of a job. If t_2 occurs, it consumes one token from place p_{busy} and it produces two tokens (one for p_{free} and one for p_{out}) both with a delay equal to zero.

3 Formal definition

In this section we define interval timed coloured Petri nets in mathematical terms, such as functions, multi-sets and relations.

3.1 Multi-sets

A *multi-set*, like a set, is a collection of elements over the same subset of some universe. However, unlike a set, a multi-set allows multiple occurrences of the same element. Another word for multi-set is *bag*. Bag theory is a natural extension of set theory (Jensen [13]).

[1] Note that $\langle p_{in}, j\rangle + \langle p_{free}, m\rangle$ and $\langle\langle p_{busy}, \langle m, j\rangle\rangle, [1,3]\rangle$ are multi-sets, see section 3.1.

Definition 1 (multi-sets)
A **multi-set** b, over a set A is a function from A to $\mathbb{N}$, i.e. $b \in A \rightarrow \mathbb{N}$.[2] If $a \in A$ then $b(a)$ is the number of occurrences of a in the multi-set b. A_{MS} is the set of all multi-sets over A. The empty multi-set is denoted by $\emptyset_A$ (or $\emptyset$). We often represent a multi-set $b \in A_{MS}$ by the formal sum:[3]

$$\sum_{a \in A} b(a)\, a$$

Consider for example the set $A = \{a, b, c, ..\}$, the multi-sets $3a$, $a + b + c + d$, $1a + 2b + 3c + 4d$ and $\emptyset_A$ are members of A_{MS}.

Definition 2
We now introduce some operations on multi-sets. Most of the set operators can be extended to multi-sets in a rather straightforward way. Suppose A a set, $b_1, b_2 \in A_{MS}$ and $q \in A$:

$q \in b_1$ iff $b_1(q) \geq 1$	(membership)
$b_1 \leq b_2$ iff $\forall_{a \in A}\ b_1(a) \leq b_2(a)$	(inclusion)
$b_1 = b_2$ iff $b_1 \leq b_2$ and $b_2 \leq b_1$	(equality)
$b_1 + b_2 = \sum_{a \in A} (b_1(a) + b_2(a))\, a$	(summation)
$b_1 - b_2 = \sum_{a \in A} ((b_1(a) - b_2(a)) \max 0)\, a$	(subtraction)
$\#b_1 = \sum_{a \in A} b_1(a)$	(cardinality of a finite multi-set)

See Jensen [13, 14] for more details.

3.2 Definition of interval timed coloured Petri nets

The ITCPN model presented in this paper is analogous to the model described in [3]. However, in this paper we give a definition which is closer to the definition of Coloured Petri Nets (CPN), see Jensen [12, 13, 14].
Nearly all timed Petri net models use a continuous time domain, so do we.

Definition 3
TS is the **time set**, $TS = \{x \in \mathbb{R} \mid x \geq 0\}$, i.e. the set of all non-negative reals.
$INT = \{[y, z] \in TS \times TS \mid y \leq z\}$, represents the set of all closed intervals.
If $x \in TS$ and $[y, z] \in INT$, then $x \in [y, z]$ iff $y \leq x \leq z$.

We define an interval timed coloured Petri nets as follows:

[2] $\mathbb{N} = \{0, 1, 2, ..\}$
[3] This notation has been adopted from Jensen [13].

Definition 4 (ITCPN)
An **Interval Timed Coloured Petri Net** is a five tuple ITCPN $= (\Sigma, P, T, C, F)$ satisfying the following requirements:

(i) Σ is a finite set of types, called **colour sets.**

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

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

(iv) C is a **colour function.** It is defined from P into Σ, i.e. $C \in P \rightarrow \Sigma$.

(v) $CT = \{\langle p, v\rangle \mid p \in P \wedge v \in C(p)\}$ is the set of all possible **coloured tokens.**

(vi) F is the **transition function.** It is defined from T into functions. If $t \in T$, then:[4]

$$F(t) \in CT_{MS} \not\rightarrow (CT \times INT)_{MS}$$

(i) Σ is a set of types. Each type is a set of colours which may be attached to one of the places.

(ii) and (iii) The places and transitions are described by two disjoint sets, i.e. $P \cap T = \emptyset$.

(iv) Each place $p \in P$ has a set of allowed colours attached to it and this means that a token residing in p must have a value v which is an element of this set, i.e. $v \in C(p)$.

(v) CT is the set of all coloured tokens, i.e. all pairs $\langle p, v\rangle$ where p is the position of the token and v is the value of the token.

(vi) The transition function specifies each transition in the ITCPN. For a transition t, $F(t)$ specifies the relation between the multi-set of consumed tokens and the multi-set of produced tokens. The domain of $F(t)$ describes the condition on which transition t is enabled. Note that the produced tokens have a delay specified by an interval. In this paper, we require that both the multi-set of consumed tokens and the multi-set of produced tokens contain finitely many elements.
Apart from the interval timing and a transition function instead of incidence functions, this definition resembles the definition of a CP-matrix (see Jensen [12, 14]).

3.3 Dynamic behaviour of interval timed coloured Petri nets

The five tuple (Σ, P, T, C, F) specifies the static structure of an ITCPN. In the remainder of this section we define the behaviour of an interval timed coloured Petri net, i.e. the semantics of the ITCPN model.

[4] $A \not\rightarrow B$ denotes the set of all partial functions from A to B.

Definition 5
A **state** is defined as a multi-set of coloured tokens each bearing a timestamp. S is the **state space**, i.e. the set of all possible states:

$$S = (CT \times TS)_{MS}$$

The **marking** of an ITCPN in state $s \in S$ is the 'untimed' token distribution: $M(s) \in CT_{MS}$ and

$$M(s) = \sum_{\langle\langle p,v\rangle,x\rangle \in CT \times TS} s(\langle\langle p,v\rangle,x\rangle)\,\langle p,v\rangle$$

A state of the ITCPN is a multi-set of coloured tokens bearing a timestamp, i.e. a multi-set of tuples $\langle\langle p,v\rangle,x\rangle$ ($p \in P$, $v \in C(p)$ and $x \in TS$). The state shown in figure 1 is $\langle\langle p_{in}, J1\rangle, 1\rangle + \langle\langle p_{free}, M1\rangle, 0\rangle$, that is a state with one token in p_{in} with value $J1$ and one token in p_{free} with value $M1$. The token in p_{in} bears timestamp 1, the token in p_{free} bears timestamp 0.

Definition 6
An **event** is a triple $\langle t, b_{in}, b_{out}\rangle$, which represents the possible firing of transition t while removing the tokens specified by the multi-set b_{in} and adding the tokens specified by the multi-set b_{out}. E is the **event set**:

$$E = T \times (CT \times TS)_{MS} \times (CT \times TS)_{MS}$$

An event $e = \langle t, b_{in}, b_{out}\rangle$ represents the firing of t while consuming the tokens specified by b_{in} and producing the tokens specified by b_{out}. If $\langle\langle p,v\rangle,x\rangle \in b_{in}$, then e consumes a token from p with value v and timestamp x. If $\langle\langle p',v'\rangle,x'\rangle \in b_{out}$, then e produces a token for p' with value v' and *delay* x'. Note that x' is relative to the firing time and x' is a member of one of the delay intervals specified by $F(t)(b_{in})$. To select arbitrary members of these delay intervals, we need the specialization concept.

Definition 7 (Specialization)
To relate multi-sets of tokens bearing timestamps with multi-sets of tokens bearing (time) intervals, we define the **specialization relation**, $\lhd \subseteq (CT \times TS)_{MS} \times (CT \times INT)_{MS}$. For $b \in (CT \times TS)_{MS}$ and $\overline{b} \in (CT \times INT)_{MS}$, $b \lhd \overline{b}$ if and only if each token in b corresponds to exactly one token in $\overline{b}$, such that they are in the same place, have the same value and the timestamp of the token in b is in the (time) interval of the token in $\overline{b}$.
More formally: $b \lhd \overline{b}$ if and only if $(b = \emptyset$ and $\overline{b} = \emptyset)$ or

$$\exists_{\langle\langle p,v\rangle,x\rangle \in b}\ \exists_{\langle\langle p,v\rangle,[y,z]\rangle \in \overline{b}}\ (x \in [y,z]) \text{ and } (b - \langle\langle p,v\rangle,x\rangle) \lhd (\overline{b} - \langle\langle p,v\rangle,[y,z]\rangle)$$

Consider for example:
$\emptyset \lhd \emptyset$
$\langle\langle p_{in}, J1\rangle, 1\rangle \lhd \langle\langle p_{in}, J1\rangle, [0.5, 1.5]\rangle$

$$\langle\langle p_{in}, J1\rangle, 1\rangle + 2\,\langle\langle p_{free}, M1\rangle, 0\rangle \;\lhd\; \langle\langle p_{in}, J1\rangle, [0.5, 1.5]\rangle + 2\,\langle\langle p_{free}, M1\rangle, [0, 1]\rangle$$

Note that $\langle\langle p_{in}, J1\rangle, 1\rangle$ is not a specialization of $\langle\langle p_{in}, J2\rangle, [0.5, 1.5]\rangle$, because the values of the two tokens differ.
If $b \lhd \overline{b}$, then there exists a bijection between the tokens in b and the tokens in $\overline{b}$ such that each token in b corresponds to exactly one token in $\overline{b}$ which is in the same place, has the same value and a 'matching' time-interval.

Definition 8
An event $\langle t, b_{in}, b_{out}\rangle \in E$ is **enabled** in state $s \in S$ iff:

(i) $b_{in} \leq s$

(ii) $M(b_{in}) \in dom(F(t))$

(iii) $b_{out} \lhd F(t)(M(b_{in}))$

An event is enabled iff:
(i) The tokens to be consumed are present in the current state.
(ii) A transition is enabled if there are 'enough' tokens on each of its input places, this is specified by the domain of $F(t)$. Note that the enabling may depend upon the values of the tokens to be consumed, but not on their timestamps.
(iii) The number and values of the tokens to be produced are determined by the multi-set $F(t)(M(b_{in}))$. This multi-set also specifies upper and lower bounds for the delays of these tokens.

Definition 9
The **enabling time** of an event $\langle t, b_{in}, b_{out}\rangle \in E$ is the maximum of all the time-stamps of the tokens consumed, i.e.

$$ET(\langle t, b_{in}, b_{out}\rangle) \;=\; \max_{\langle\langle p,v\rangle,x\rangle \in b_{in}} x$$

An enabled event is **time enabled** iff no other enabled events have a smaller enabling time.

If an event is time enabled, it may occur. In fact, a transition fires as soon as possible (transitions are 'eager'). Although the time domain is continuous ($TS = \{x \in \mathbb{R} \mid x \geq 0\}$), time progresses discontinuously.

Definition 10
The **model time** of a state $s \in S$ is the minimum of all enabling times, i.e.

$$MT(s) \;=\; \min\{ET(e) \mid e \in E \text{ and } e \text{ is enabled in state } s\}$$

The model time only changes if something happens. Note that an enabled event e is time enabled in state s iff $ET(e) = MT(s)$.

If $\langle t, b_{in}, b_{out}\rangle \in E$ an enabled event, then a timestamp in b_{out} represents the delay of the token instead of an absolute timestamp. Therefore we need a function to 'scale' timestamps.

Definition 11
Function $SC \in ((CT \times TS)_{MS} \times TS) \rightarrow (CT \times TS)_{MS}$ scales the timestamps in a multi-set of timed coloured tokens. For $b \in (CT \times TS)_{MS}$ and $y \in TS$, we have:

$$SC(b, y) = \sum_{\langle\langle p,v\rangle,x\rangle \in CT \times TS} b(\langle\langle p, v\rangle, x\rangle)\,\langle\langle p, v\rangle, x + y\rangle$$

A time enabled event e in state s may occur at time $ET(e) = MT(s)$.

Definition 12
When an enabled event $\langle t, b_{in}, b_{out}\rangle$ is time enabled in state s_1, it may **occur**, i.e. transition t **fires** while removing the tokens specified by b_{in} and adding the tokens specified by b_{out}.
If $\langle t, b_{in}, b_{out}\rangle$ occurs in state s_1, then the net changes into the state s_2, defined by:

$$s_2 = (s_1 - b_{in}) + SC(b_{out}, ET(\langle t, b_{in}, b_{out}\rangle))$$

State s_2 is said to be **directly reachable** from s_1 by the occurrence of event $e = \langle t, b_{in}, b_{out}\rangle$, this is also denoted by:

$$s_1 \xrightarrow{e} s_2$$

Moreover, $s_1 \longrightarrow s_2$ means that there exists a time enabled event e such that $s_1 \xrightarrow{e} s_2$. Transitions fire as soon as possible, i.e. if an event occurs, then it occurs at its enabling time.

Definition 13
A **firing sequence** is a sequence of states and events:

$$s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \xrightarrow{e_3} s_4 \xrightarrow{e_4} ..$$

State s_n is **reachable** from s_1 iff there exists a firing sequence of finite length starting in s_1 and ending in s_n:

$$s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \xrightarrow{e_3} .. \xrightarrow{e_{n-1}} s_n$$

An important property of the ITCPN model is the monotonicity of time, i.e. time can only move forward.

Theorem 1 (Monotonicity)
If $s_1, s_2 \in S$ and $s_1 \longrightarrow s_2$, then $MT(s_1) \leq MT(s_2)$.

Proof
Let $s_1, s_2 \in S$ and $e = \langle t, b_{in}, b_{out} \rangle \in E$ such that $s_1 \xrightarrow{e} s_2$.
$ET(e) = MT(s_1)$ and $s_2 = (s_1 - b_{in}) + SC(b_{out}, ET(e))$.
Deleting tokens (b_{in}) may disable events, but it will never enable new ones.
Adding tokens ($SC(b_{out}, ET(e))$) may enable new events. However, in this case these events have an enabling time of at least $ET(e)$.
Since $MT(s_2)$ is the minimum of all enabling times, we have $MT(s_2) \geq MT(s_1)$.
□

In [3], we prove a number of other properties of the ITCPN model (e.g. progressiveness, etc.). In that monograph we also show how to model other timed Petri nets (e.g. Merlin's timed Petri nets) in terms of our model.

4 An analysis method

In this section we present an approach to verify certain properties and to calculate bounds for all sorts of performance measures. This approach is based on an analysis method, called *Modified Transition System Reduction Technique* (MTSRT), which generates a *reduced reachability graph*. The MTSRT method can be applied to arbitrary ITCPNs.
Existing techniques which can be used to analyse the dynamic behaviour of timed and coloured Petri nets, may be subdivided into three classes: simulation, reachability analysis and Markovian analysis.
Simulation is a technique to analyse a system by conducting controlled experiments. Because simulation does not require difficult mathematical techniques, it is easy to understand for people with a non-technical background. Simulation is also a very powerful analysis technique, since it does not set additional restraints. However, sometimes simulation is expensive in terms of the computer time necessary to obtain reliable results. Another drawback is the fact that (in general) it is not possible to use simulation to *prove* that the system has the desired set of properties.
Recent developments in computer technology stimulate the use of simulation for the analysis of timed coloured Petri nets. The increased processing power allows for the simulation of large nets. Modern graphical screens are fast and have a high resolution. Therefore, it is possible to visualize a simulation graphically (i.e. animation).
Reachability analysis is a technique which constructs a reachability graph, sometimes referred to as reachability tree or occurrence graph (cf. Jensen [12, 14]). Such a reachability graph contains a node for each possible state and an arc for each possible state change. Reachability analysis is a very powerful method in the sense that it can be used to prove all kinds of properties. Another advantage is the fact that it does not set additional restraints. Obviously, the reachability graph needed to prove these properties may, even for small nets, become very large (and often infinite). If we want to inspect the reachability graph by means of a computer, we have to solve this problem. This is the reason several authors developed reduction techniques (Hubner et al. [11] and Valmari [24]). Unfortunately, it is not known how to apply these

techniques to timed coloured Petri nets.
For timed coloured Petri nets with certain types of stochastic delays it is possible to translate the net into a *continuous time Markov chain.* This Markov chain can be used to calculate performance measures like the average number of tokens in a place and the average firing rate of a transition.
If all the delays are sampled from a negative exponential probability distribution, then it is easy to translate the timed coloured Petri net into a continuous time Markov chain. Several authors attempted to increase the modelling power by allowing other kinds of delays, for example mixed deterministic and negative exponential distributed delays, and phase-distributed delays (see Ajmone Marsan et al. [15]). Nearly all stochastic Petri net models (and related analysis techniques) do not allow for coloured tokens, because the increased modelling power is offset by computational difficulties. This is the reason stochastic high-level Petri nets are often used in a simulation context only.

Besides the aforementioned techniques to analyse the behaviour of timed coloured Petri nets, there are several analysis techniques for Petri nets without 'colour' or explicit 'time'.

An interesting way to analyse a coloured Petri net is to calculate (or verify) *place and transition invariants.* Place and transition invariants can be used to prove properties of the modelled system. Intuitively, a place invariant assigns a weight to each token such that the weighted sum of all tokens in the net remains constant during the execution of any firing sequence. By calculating these place invariants we find a set of equations which characterizes all reachable states. Transition invariants are the duals of place invariants and the main objective of calculating transition invariants is to find firing sequences with no 'effects'.
Note that we can calculate these invariants for *timed* coloured Petri nets (e.g. an ITCPN). However, in this case, we do not really use the timing information. Therefore, in general, these invariants do not characterize the dynamic behaviour of the system. On the other hand, they can be used to verify properties which are time independent. For more information about the calculation of invariants in a coloured Petri net, see Jensen [12, 14].

In our ITCPN model, a delay is described by an interval rather than a fixed value or some delay distribution. On the one hand, interval delays allow for the modelling of variable delays, on the other hand, it is not necessary to determine some artificial delay distribution (as opposed to stochastic delays). Instead, we have to specify bounds. These bounds are used to specify and to verify time constraints. This is very important when modelling time-critical systems, i.e. *real-time* systems with 'hard' deadlines. These deadlines have to be met for a safe operation of the system. An acceptable behaviour of the system depends not only on the logical correctness of the results, but also on the time at which the results are produced. Therefore, we are interested in techniques to verify these deadlines and to calculate upper and lower

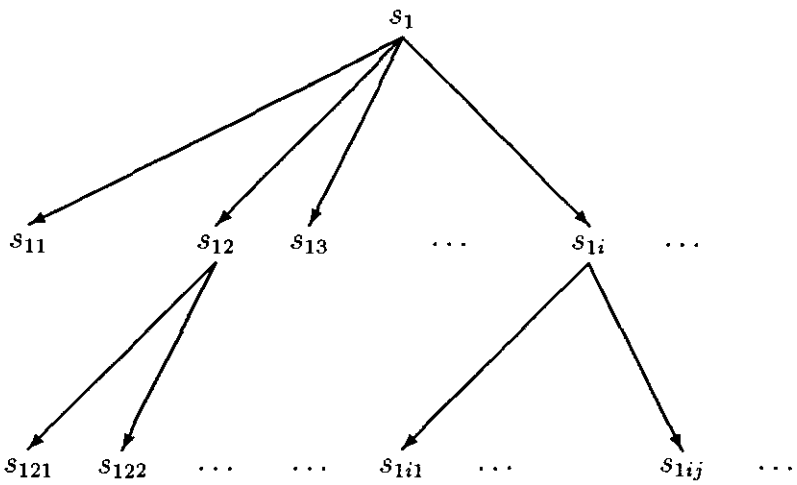

Figure 2: A reachability graph

bounds for all sorts of performance criteria.

4.1 Reachability graphs

In section 3.3, we defined the behaviour of an ITCPN. The definitions of that section can be used to construct the so-called **reachability graph**. The basic idea of a reachability graph is to organize all reachable markings in a graph, where each node represents a state and each arc represents an event transforming one state into another state.
Consider for example the reachability graph shown in figure 2. Suppose that s_1 is the initial state of the ITCPN we want to consider. This state is connected to a number of states $s_{11}, s_{12}, s_{13}, ..$ reachable from s_1 by the firing of some transition, i.e. $s_1 \longrightarrow s_{1i}$. These states are called the 'successors' (or children) of the s_1. Repeating this process produces the graphical representation of the reachability graph, see figure 2. Such a reachability graph contains all relevant information about the dynamic behaviour of the system. If we are able to generate this graph, we can answer 'any' kind of question about the behaviour of the system.

Obviously such a reachability graph may, even for small nets, become very large (often infinite). Many authors have presented analysis techniques for the efficient calculation of a reachability graph of an *untimed coloured* Petri net (e.g. [24], [13], [11], [6]). In this section we focus on the reachability graph of an ITCPN.

In general the number of reachable states of an ITCPN (given an initial state) is infinite. This is mainly caused by the fact that we use interval timing. Consider an enabled transition. In general, there is an infinite number of allowed firing delays, all resulting in a different state. Consider for example the ITCPN shown in figure 3. If transition t occurs, then it consumes one token from p_1 and it produces one token for

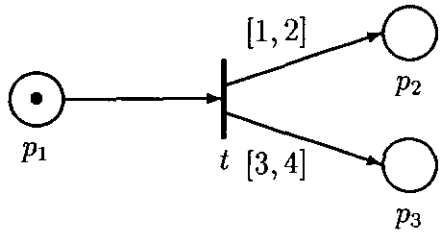

Figure 3: An ITCPN

p_2 and one token for p_3. The delay intervals are given in the figure. Suppose the initial state is such that there is one token in p_1 with timestamp 0. The number of successors of this state is infinite, because all states with one token in p_2 having a timestamp $x \in [1,2]$ and one token in p_3 having a timestamp $y \in [3,4]$ are reachable. It may seem unreasonable that this simple example corresponds to a reachability graph with an infinite number of states. This is the reason we developed the Modified Transition System Reduction Technique described in this section. This technique generates the reachability graph and uses, for computational reasons only, alternative definitions for the dynamic behaviour of an ITCPN.

4.2 Reducing the reachability graph of an ITCPN

The Modified Transition System Reduction Technique (MTSRT) uses alternative definitions for the dynamic behaviour of an ITCPN, i.e. the MTSRT method uses alternative semantics. The main difference between these definitions and the original ones is the fact that we attach a time-*interval* to every token instead of a timestamp, i.e. $\overline{S} = (CT \times INT)_{MS}$. We will show that, using these semantics, it is possible to calculate the set of reachable states (or at least a relevant subset). Since, the reachability graph of the ITCPN using these alternative semantics is much smaller and more coarsely grained than the original one, we call it the **reduced reachability graph**. Every state in the reduced reachability graph corresponds to a (infinite) number of states in the original reachability graph. One may think of these states as *equivalence* or *state classes*. One state class $\overline{s} \in \overline{S}$ corresponds to the set of all states being a specialization of $\overline{s}$, i.e. $\{s \in S \mid s \lhd \overline{s}\}$. Informally speaking, state classes are defined as the union of 'similar' states having the same token distribution (marking) but different timestamps (within certain bounds).

In the remainder of this section we redefine the behaviour of an interval timed coloured Petri net, i.e. we give alternative semantics of the ITCPN model.
In section 4.3, we will show how these two semantics relate to each other. We will see that the alternative definitions given in this section can be used to answer questions about the behaviour of the ITCPN (as specified in section 3.3).

Definition 14
A **state class** is defined as a multi-set of coloured tokens each bearing a time-interval. $\overline{S}$ is the **state space**:[5]

$$\overline{S} = (CT \times INT)_{MS}$$

An **event** is a triple $\langle t, b_{in}, b_{out} \rangle$, which represents the possible firing of transition t while removing the tokens specified by the multi-set b_{in} and adding the tokens specified by the multi-set b_{out}. $\overline{E}$ is the **event set**:

$$\overline{E} = T \times (CT \times INT)_{MS} \times (CT \times INT)_{MS}$$

Note that tokens bear time-*intervals* instead of timestamps.

Definition 15
An event $\langle t, b_{in}, b_{out} \rangle \in \overline{E}$ is **enabled** in state class $s \in \overline{S}$ iff:

(i) $b_{in} \leq s$

(ii) $M(b_{in}) \in dom(F(t))$

(iii) $b_{out} = F(t)(M(b_{in}))$

The point of time a token becomes available is specified by an interval, therefore it is impossible to specify *the* enabling time of an event. However, it is possible to give an upper and lower bound for the enabling time of an event $e \in \overline{E}$.

Definition 16
The **minimum** and **maximum enabling time** of an event $\langle t, b_{in}, b_{out} \rangle \in \overline{E}$ are defined as follows:

$$ET_{min}(\langle t, b_{in}, b_{out} \rangle) = \max_{\langle\langle p,v\rangle,[y,z]\rangle \in b_{in}} y$$

$$ET_{max}(\langle t, b_{in}, b_{out} \rangle) = \max_{\langle\langle p,v\rangle,[y,z]\rangle \in b_{in}} z$$

An enabled event e is **time enabled** iff the minimum enabling time of e is smaller than the maximum enabling time of any other enabled event.

If $\langle t, b_{in}, b_{out} \rangle \in \overline{E}$ an enabled event, then a time-interval in b_{out} represents the delay interval of the token instead of an absolute time-interval. Therefore we need a function to 'scale' time-intervals.

[5]Symbols superscripted by a horizontal line are associated with the alternative semantics, this to avoid confusion.

Definition 17
Function $\overline{SC} \in ((CT \times INT)_{MS} \times INT) \rightarrow (CT \times INT)_{MS}$ scales the time-intervals in a multi-set of timed coloured tokens. For $b \in (CT \times INT)_{MS}$ and $[y, z] \in INT$, we have:

$$\overline{SC}(b, [y, z]) = \sum_{\langle\langle p,v\rangle,[y',z']\rangle \in CT \times INT} b(\langle\langle p, v\rangle, [y', z']\rangle)\, \langle\langle p, v\rangle, [x' + x, y' + y]\rangle$$

A time enabled event e in state class s may occur at a time between $ET_{min}(e)$ and $MT_{max}(s) = \min\{ET_{max}(e) \mid e \in \overline{E} \text{ and } e \text{ is enabled in } s\}$.

Definition 18
When an enabled event $\langle t, b_{in}, b_{out}\rangle$ is time enabled in state class $s_1 \in \overline{S}$, it may **occur**, i.e. transition t **fires** while removing the tokens specified by b_{in} and adding the tokens specified by b_{out}.
If $\langle t, b_{in}, b_{out}\rangle$ occurs in state class s_1, then the net changes into the state class $s_2 \in \overline{S}$, defined by:

$$s_2 = (s_1 - b_{in}) + \overline{SC}(b_{out}, [ET_{min}(\langle t, b_{in}, b_{out}\rangle), MT_{max}(s_1)])$$

State class s_2 is said to be **directly reachable** from s_1 by the occurrence of event $e = \langle t, b_{in}, b_{out}\rangle$, this is also denoted by:

$$s_1 \stackrel{e}{\Longrightarrow} s_2$$

Note that we use double arrows to denote possible state (class) changes given the alternative semantics of an ITCPN. If we use the definitions given in this section for the generation of the reachability graph, then we obtain the **reduced reachability graph**. Comparing these definitions with the definitions given in section 3.3 shows that all differences stem from the fact that the alternative semantics associate a time-interval (instead of a timestamp) with each token. As a result of these intervals, the enabling time of an event and the model time of a state class are both characterized by an upper and lower bound, etc.

4.3 Soundness

The alternative definitions of section 4.2 have been given for computational reasons. However, calculating the reduced reachability graph only makes sense if the reduced reachability graph can be used to deduce properties of the reachability graph which represents the behaviour of the ITCPN. Therefore, we investigate the relation between the two reachability graphs. Examples indicate that such a relation exists.
It is easy to see that the two reachability graphs are not equivalent. Moreover, there is no sensible morphism between them (see [3]). However, state $s \in S$ and state class $\overline{s} \in \overline{S}$ seem to be 'related' if s is a specialization of $\overline{s}$ (i.e. $s \lhd \overline{s}$). Recall that $s \lhd \overline{s}$ if and only if each token in s corresponds to exactly one token in $\overline{s}$, such that they are in the same place, have the same value and the timestamp of the token in s is in the

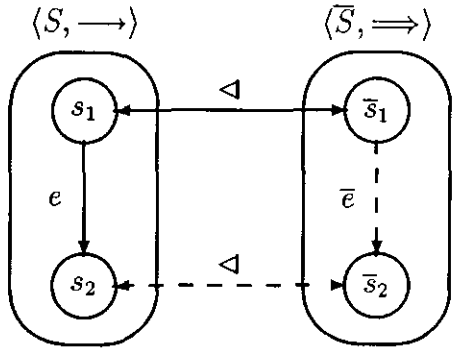

Figure 4: The soundness property

(time) interval of the token in $\overline{s}$ (see definition 7).

Theorem 2 (Soundness)
Let (Σ, P, T, C, F) be an ITCPN and $s_1, s_2 \in S$ such that $s_1 \longrightarrow s_2$. If $\overline{s}_1 \in \overline{S}$ and $s_1 \lhd \overline{s}_1$, then there exists a state class $\overline{s}_2 \in \overline{S}$ such that $\overline{s}_1 \Longrightarrow \overline{s}_2$ and $s_2 \lhd \overline{s}_2$.

Proof [6]
Let $s_1, s_2 \in S$, $\overline{s}_1 \in \overline{S}$ and $e \in E$ such that $s_1 \xrightarrow{e} s_2$ and $s_1 \lhd \overline{s}_1$. Now we have to prove that there exists an $\overline{e}$ such that $\overline{s}_1 \xRightarrow{\overline{e}} \overline{s}_2$ and $s_2 \lhd \overline{s}_2$ (see figure 4).
Suppose $e = \langle t, b_{in}, b_{out} \rangle$ and let $\overline{e} = \langle t, \overline{b}_{in}, \overline{b}_{out} \rangle$ where $\overline{b}_{out} = F(t)(M(b_{in}))$ and $\overline{b}_{in}$ such that $\overline{b}_{in} \leq \overline{s}_1$ and $b_{in} \lhd \overline{b}_{in}$ (this is possible because $s_1 \lhd \overline{s}_1$ and $b_{in} \leq s_1$).

Remains to prove that: (1) $\overline{e}$ is enabled, (2) $\overline{e}$ is time enabled and (3) $s_2 \lhd \overline{s}_2$.
(1) Event $\overline{e}$ is enabled, because $\overline{b}_{in} \leq \overline{s}_1$, $M(\overline{b}_{in}) \in dom(F(t))$ and $\overline{b}_{out} = F(t)(M(\overline{b}_{in}))$.[7]
(2) $ET_{min}(\overline{e}) \leq ET(e) = MT(s_1) \leq MT_{max}(\overline{s}_1)$, i.e. $\overline{e}$ is time enabled.
(3) Since $s_1 \lhd \overline{s}_1$, $b_{in} \lhd \overline{b}_{in}$ and $b_{out} \lhd \overline{b}_{out}$, we have that:
$SC(b_{out}, ET(\langle t, b_{in}, b_{out} \rangle)) \lhd \overline{SC}(\overline{b}_{out}, [ET_{min}(\langle t, \overline{b}_{in}, \overline{b}_{out} \rangle), MT_{max}(\overline{s}_1)])$ and
$s_2 = (s_1 - b_{in}) + SC(b_{out}, ET(\langle t, b_{in}, b_{out} \rangle))$ is a specialization of
$\overline{s}_2 = (\overline{s}_1 - \overline{b}_{in}) + \overline{SC}(\overline{b}_{out}, [ET_{min}(\langle t, \overline{b}_{in}, \overline{b}_{out} \rangle), MT_{max}(\overline{s}_1)])$, i.e. $s_2 \lhd \overline{s}_2$.
□

This theorem tells us that if an event occurs which changes state s_1 into s_2, then there is a corresponding event which changes any state class $\overline{s}_1$ that 'covers' s_1 into a state class which 'covers' s_2 (see figure 4). We say that the alternative semantics are 'sound'.

An implication of theorem 2 is that if $s_1 \lhd \overline{s}_1$ and there exists a firing sequence $s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \xrightarrow{e_3} .. \xrightarrow{e_{n-1}} s_n$, then there exists a firing sequence $\overline{s}_1 \xrightarrow{\overline{e}_1} \overline{s}_2 \xrightarrow{\overline{e}_2}$

[6] A more formal proof is given in [3].
[7] $M(\overline{b}_{in}) = M(b_{in})$, because $b_{in} \lhd \overline{b}_{in}$.

$\overline{s}_3 \xrightarrow{\overline{e}_3} .. \xrightarrow{\overline{e}_{n-1}} \overline{s}_n$ such that $s_n \lhd \overline{s}_n$. For any state reachable from s_1 there is a related state class reachable from $\overline{s}_1$.

If we compare the reachability graph and the reduced reachability of an arbitrary ITCPN, then we see that all the state changes possible in the reachability graph are also possible in the reduced reachability. Note that the opposite does not hold, because of *dependencies* between tokens are not taken into account. Consider for example the net shown in figure 3. Suppose there is one token in $p1$ with a time interval $[0, 1]$ and the other places are empty. In this case t fires between time 0 ($ET_{min}(e)$) and time 1 ($MT_{max}(s)$). The next state of the ITCPN using the alternative definitions given in section 4.2, will be the state with one token in $p2$ (with interval $[1, 3]$) and one token in $p3$ (with interval $[3, 5]$). This suggests that it is possible to have a token in $p2$ with timestamp 1 and a token in $p3$ with timestamp 5. However, this is not possible, because these timestamps are related (i.e. they where produced at the same time). We say that the alternative semantics are not 'complete'.

Despite the non-completeness, the soundness property allows us to answer various questions. We can *prove* that a system has a desired set of properties by proving it for the modified transition system. For example, we can use the reduced reachability graph to prove boundedness, absence of traps and siphons (deadlocks), etc. The reduced reachability graph may also be used to analyse the *performance* of the system modelled by an ITCPN. With performance we mean characteristics, such as: response times, occupation rates, transfer rates, throughput times, failure rates, etc. The MT-SRT method can be used to calculate bounds for performance measures. Although these bounds are sound (i.e. safe) they do not have to be as tight as possible, because of possible dependencies between tokens (non-completeness). However, experimentation shows that the calculated bounds are often of great value and far from trivial. Moreover, we are able to answer questions which cannot be answered by simulation or the method proposed by Berthomieu et al. [5].
We have modelled and analysed many examples using the approach presented in this paper, see Van der Aalst [3, 2, 1] and Odijk [20]. To facilitate the analysis of real-life systems we have developed an analysis tool, called *IAT* ([3]). This tool also supports more traditional kinds of analysis such as the generation of place and transition invariants. IAT is part of the software package *ExSpect* (see Van Hee et al. [9] and Van der Aalst [3, 4]).

5 Conclusion

In this paper, we presented a coloured Petri net model extended with time. This ITCPN model uses a new timing mechanism where time is associated with tokens and transitions determine a delay specified by an interval. The formal semantics of the ITCPN model have been defined in section 3. The fact that time is in tokens results in transparent semantics and a compact state representation. Specifying each delay by an interval rather than a deterministic value or stochastic variable is promising,

since it is possible to model uncertainty without having to bother about the delay distribution.

From the analysis point of view, the ITCPN model is also interesting, since interval timing allows for new analysis methods. In this paper, the MTSRT method has been described. This is a powerful analysis method, since it can be applied to arbitrary nets and answers a large variety of questions. This method constructs a reduced reachability graph. In such a graph a node corresponds to a set of (similar) states, instead of a single state.

A lot of applications have been modelled and analysed using the approach described in this paper and the software package ExSpect which supports the MTSRT method. Experimentation shows that, in general, the results obtained using the MTSRT method are quite meaningful. A direction of further research is to incorporate other reduction techniques for coloured Petri nets into our approach (e.g. [24], [13], [11], [6]).

References

[1] W.M.P. VAN DER AALST, *Interval Timed Petri Nets and their analysis.* Computing Science Notes 91/09, Eindhoven University of Technology, Eindhoven, 1991.

[2] ——, *Modelling and Analysis of Complex Logistic Systems*, in Proceedings of the IFIP WG 5.7 Working Conference on Integration in Production Management Systems, Eindhoven, the Netherlands, 1992, pp. 203–218.

[3] ——, *Timed coloured Petri nets and their application to logistics*, PhD thesis, Eindhoven University of Technology, Eindhoven, 1992.

[4] W.M.P. VAN DER AALST AND A.W. WALTMANS, *Modelling logistic systems with EXSPECT*, in Dynamic Modelling of Information Systems, H.G. Sol and K.M. van Hee, eds., Elsevier Science Publishers, Amsterdam, 1991, pp. 269–288.

[5] B. BERTHOMIEU AND M. DIAZ, *Modelling and verification of time dependent systems using Time Petri Nets*, IEEE Transactions on Software Engineering, 17 (1991), pp. 259–273.

[6] G. CHIOLA, C. DUTHEILLET, G. FRANCESCHINIS, AND S. HADDAD, *On well-formed coloured nets and their symbolic reachability graph*, in Proceedings of the 11th International Conference on Applications and Theory of Petri Nets, Paris, June 1990, pp. 387–411.

[7] G. FLORIN AND S. NATKIN, *Evaluation based upon Stochastic Petri Nets of the Maximum Throughput of a Full Duplex Protocol*, in Application and theory of Petri nets : selected papers from the first and the second European workshop, C. Girault and W. Reisig, eds., vol. 52 of Informatik Fachberichte, Berlin, 1982, Springer-Verlag, New York, pp. 280–288.

[8] H.J. GENRICH AND K. LAUTENBACH, *System modelling with high level Petri nets*, Theoretical Computer Science, 13 (1981), pp. 109–136.

[9] K.M. VAN HEE, L.J. SOMERS, AND M. VOORHOEVE, *Executable specifications for distributed information systems*, in Proceedings of the IFIP TC 8 / WG 8.1 Working Conference on Information System Concepts: An In-depth Analysis, E.D. Falkenberg

and P. Lindgreen, eds., Namur, Belgium, 1989, Elsevier Science Publishers, Amsterdam, pp. 139–156.

[10] K.M. VAN HEE AND P.A.C. VERKOULEN, *Integration of a Data Model and Petri Nets*, in Proceedings of the 12th International Conference on Applications and Theory of Petri Nets, Aarhus, June 1991, pp. 410–431.

[11] P. HUBNER, A.M. JENSEN, L.O. JEPSEN, AND K. JENSEN, *Reachability trees for high level Petri nets*, Theoretical Computer Science, 45 (1986), pp. 261–292.

[12] K. JENSEN, *Coloured Petri Nets*, in Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties, W. Brauer, W. Reisig, and G. Rozenberg, eds., vol. 254 of Lecture Notes in Computer Science, Springer-Verlag, New York, 1987, pp. 248–299.

[13] ——, *Coloured Petri Nets: A High Level Language for System Design and Analysis*, in Advances in Petri Nets 1990, G. Rozenberg, ed., vol. 483 of Lecture Notes in Computer Science, Springer-Verlag, New York, 1990, pp. 342–416.

[14] ——, *Coloured Petri Nets. Basic concepts, analysis methods and practical use.*, to appear in EATCS monographs on Theoretical Computer Science, Springer-Verlag, New York, 1992.

[15] M. AJMONE MARSAN, G. BALBO, A. BOBBIO, G. CHIOLA, G. CONTE, AND A. CUMANI, *On Petri Nets with Stochastic Timing*, in Proceedings of the International Workshop on Timed Petri Nets, Torino, 1985, IEEE Computer Society Press, pp. 80–87.

[16] M. AJMONE MARSAN, G. BALBO, AND G. CONTE, *A Class of Generalised Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems*, ACM Transactions on Computer Systems, 2 (1984), pp. 93–122.

[17] P. MERLIN, *A Study of the Recoverability of Computer Systems*, PhD thesis, University of California, Irvine, California, 1974.

[18] P. MERLIN AND D.J. FABER, *Recoverability of communication protocols*, IEEE Transactions on Communication, 24 (1976), pp. 1036–1043.

[19] S. MORASCA, M. PEZZÈ, AND M. TRUBIAN, *Timed High-Level Nets*, The Journal of Real-Time Systems, 3 (1991), pp. 165–189.

[20] M. ODIJK, *ITPN analysis of ExSpect specifications with respect to production logistics*, Master's thesis, Eindhoven University of Technology, Eindhoven, 1991.

[21] C. RAMCHANDANI, *Performance Evaluation of Asynchronous Concurrent Systems by Timed Petri Nets*, PhD thesis, Massachusetts Institute of Technology, Cambridge, 1973.

[22] W. REISIG, *Petri nets: an introduction*, Prentice-Hall, Englewood Cliffs, 1985.

[23] J. SIFAKIS, *Use of Petri Nets for performance evaluation*, in Proceedings of the Third International Symposium IFIP W.G. 7.3., Measuring, modelling and evaluating computer systems (Bonn-Bad Godesberg, 1977), H. Beilner and E. Gelenbe, eds., Elsevier Science Publishers, Amsterdam, 1977, pp. 75–93.

[24] A. VALMARI, *Stubborn sets for reduced state space generation*, in Proceedings of the 10th International Conference on Applications and Theory of Petri Nets, Bonn, June 1989.

[25] W.M. ZUBEREK, *Timed Petri Nets and Preliminary Performance Evaluation*, in Proceedings of the 7th annual Symposium on Computer Architecture, vol. 8(3) of Quarterly Publication of ACM Special Interest Group on Computer Architecture, 1980, pp. 62–82.

In this series appeared:

90/1	W.P.de Roever- H.Barringer- C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, p. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.
90/8	M.B. Josephs	Receptive Process Theory, p. 16.
90/9	A.T.M. Aerts P.M.E. De Bra K.M. van Hee	Combining the functional and the relational model, p. 15.
90/10	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).
90/11	P. America F.S. de Boer	A proof system for process creation, p. 84.
90/12	P.America F.S. de Boer	A proof theory for a sequential version of POOL, p. 110.
90/13	K.R. Apt F.S. de Boer E.R. Olderog	Proving termination of Parallel Programs, p. 7.
90/14	F.S. de Boer	A proof system for the language POOL, p. 70.
90/15	F.S. de Boer	Compositionality in the temporal logic of concurrent systems, p. 17.
90/16	F.S. de Boer C. Palamidessi	A fully abstract model for concurrent logic languages, p. p. 23.
90/17	F.S. de Boer C. Palamidessi	On the asynchronous nature of communication in logic languages: a fully abstract model based on sequences, p. 29.

90/18	J.Coenen E.v.d.Sluis E.v.d.Velden	Design and implementation aspects of remote procedure calls, p. 15.
90/19	M.M. de Brouwer P.A.C. Verkoulen	Two Case Studies in ExSpect, p. 24.
90/20	M.Rem	The Nature of Delay-Insensitive Computing, p.18.
90/21	K.M. van Hee P.A.C. Verkoulen	Data, Process and Behaviour Modelling in an integrated specification framework, p. 37.
91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.

91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.
91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.
91/31	H. ten Eikelder	Some algorithms to decide the equivalence of recursive types, p. 26.

91/32	P. Struik	Techniques for designing efficient parallel programs, p. 14.
91/33	W. v.d. Aalst	The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.
91/34	J. Coenen	Specifying fault tolerant programs in deontic logic, p. 15.
91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.
92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.
92/17	W.M.P. van der Aalst	Interval timed coloured Petri nets and their analysis, p. 20.

92/18	R.Nederpelt F. Kamareddine	A unified approach to Type Theory through a refined lambda-calculus, p. 30.
92/19	J.C.M.Baeten J.A.Bergstra S.A.Smolka	Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.
92/20	F.Kamareddine	Are Types for Natural Language? P. 32.
92/21	F.Kamareddine	Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.
92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.