Interval timed Petri nets and their analysis

Interval Timed Petri Nets
and their analysis

by

W.M.P. van der Aalst

91/09

May, 1991

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 or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands

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

Interval Timed Petri Nets and their analysis

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

Abstract

In this paper we present a new timed Petri Net model, called *Interval Timed Petri Net* (ITPN). Tokens have a timestamp and the transitions determine a delay described by an interval. This model allows for the representation of time constraints. A number of analysis methods are presented, all producing safe bounds for the time behaviour of a net. With this approach it is possible to verify dynamic properties of the systems modelled by these nets. This is very useful when modelling time-critical systems, such as real-time (computer) systems and just-in-time manufacturing systems. We have developed a software package called ExSpect. This package contains a tool, called IAT, to analyse these nets using the techniques presented in this paper.

*This research is supported by IPL-TUE/TNO

Contents

1 Introduction

Petri Nets ([Petri 80]) are often used to describe and analyse concurrent systems. Although the basic Petri Net model does not incorporate the concept of time, many researchers tried to include timing into their models. These models are called Timed Petri Nets (TPN) models.
In nearly all of these models time is in transitions and the delay of such a transition is specified by some distribution. Nets belonging to these models are the so-called stochastic Petri Nets. Two widespread models are the SPN model by Florin and Natkin ([Florin et al. 82]) and the GSPN model by Ajmone Marsan ([Marsan et al. 84]). They are mainly used for performance analysis.
Analysis of stochastic Petri Nets is possible (in theory) because under certain conditions the reachability graph can be regarded 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 ([Marsan et al. 85]). This is the reason many researchers resorted to using simulation to study the behaviour of the net. Other researchers use deterministic delays to model time (see for example [Zuberek 80]).
In this paper we present a new model called the *Interval Timed Petri Net model.* In this model firing delays are described by an interval and time is associated with tokens instead of transitions. We restrict ourselves to specifying the upper and lower bounds of the firing delays to overcome the disadvantages of stochastic Petri Nets. However the expressive power is increased considerably compared to deterministic timed Petri Nets. A similar approach was used in [Merlin 74] and [Berthomieu et al. 83], where the enabling time of a transition is specified by a minimal and a maximal time.
The Interval Timed Petri Net model has a great descriptive power. Nevertheless the model allows for various kinds of analysis. In this paper we present four *new* analysis techniques. Our first analysis technique produces upper and lower bounds for the time it takes to reach a certain place. There is some resemblance with calculating the critical path in an activity network. The second analysis technique generates a "reduced" reachability tree to answer a variety of questions such as liveness, boundedness and upper and lower bounds for the arrival time of the n-th token in a place.
The other two analysis techniques are used to analyse *Interval Timed Event Graphs*, a subclass of ITP-nets.
Note that these analysis methods are used for performance evaluation and for the verification for dynamic properties. Answering questions about the dynamic behaviour is very important in a number of fields, for instance real-time computer systems and just-in-time manufacturing systems. In this paper we will model and analyse three examples with Interval Timed Petri Nets.
We have developed a software tool, called IAT (*ITPN Analysis Tool*), to analyse these nets. At the moment IAT uses the first three analysis techniques. IAT is part of the CASE-tool *ExSpect* ([Hee et al. 89], [Aalst et al. 91]), which is based on a high level Petri Net model like the CPN-model ([Jensen 87]). We will often refer to this tool as ExSpect/IAT. The

ITPN-model is a subset of the ExSpect-model in the sense that tokens are colourless. One of the advantages of this intergration is the fact that we can use the graphical interface of ExSpect to construct or to generate Interval Timed Petri Nets.

2 Interval Timed Petri Nets

Before we describe the analysis methods we give a formal description of the ITPN model and the type of questions we want to answer for these nets.

A Petri Net is a directed labeled bipartite graph with two node types called *places* and *transitions*. The nodes are connected via labelled *arcs*. Connections between two nodes of the same type are not allowed. Places are represented by circles, transitions by bars and directed arcs by lines. A place p is called an *input place* of a transition t if there exists a directed arc from p to t. A place p is called an *output place* of a transition t if there exists a directed arc from t to p. Places may contain zero or more *tokens*, drawn as black dots. The number of tokens may change during the execution of the net. 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. An enabled transition can *fire*. Firing a transition means consuming tokens from the input places and producing tokens on the output places.

The Interval Timed Petri Net model deviates from most existing models in two ways.
Unlike traditional Petri Nets we attach a *timestamp* to every token. This timestamp indicates the time it becomes *available*. This timing concept has been adopted from [Hee et al. 89].
The *enabling time* of a transition is the maximum timestamp of the tokens to be consumed. Because transitions are eager to fire, the transition with the smallest enabling time will fire first. 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*.
The second difference with most existing net models is the fact that this delay is specified by a non negative *interval* instead of a distribution or a fixed value. In other words the delay of a token is sampled from the corresponding interval.

Our formalisms are based on *bag* theory, an extension of set theory. If you are not familiar with bags, we suggest you read appendix A.

Definition 1 (ITPN)
An ITPN is defined by a five tuple, ITPN $= (P, T, I, O, TS)$ with:

- P, the set of places
- T, the set of transitions
- $I \in T \rightarrow \mathbb{B}(P)$, the input places of transitions
- $O \in T \rightarrow \mathbb{B}(P \times INT)$, the output places of transitions with the corresponding delay intervals
- TS, the time set (a subset of $\mathbb{R}^+ \cup \{0\}$)
- $INT = \{\langle t_1, t_2 \rangle \in TS \times TS \mid t_1 \leq t_2\}$, the set of intervals

Function I gives the *multiplicity* of the input places for each transition. If $P = \{p_1, p_2, p_3\}$, $I_t(p_1) = 2$, $I_t(p_2) = 1$ and $I_t(p_3) = 0$ (or $I_t = [p_1, p_1, p_2]$) then transition t is enabled if there are at least two tokens in p_1 and one in p_2.
O is also a function over T. If $t \in T$ then O_t is the bag of tokens produced with the associated delay interval. For example, if $O_t = [\langle p_1, \langle 1,2 \rangle\rangle, \langle p_2, \langle 2,4 \rangle\rangle]$, then t produces two tokens: one for place p_1 with a delay between 1 and 2 and one for place p_2 with a delay between 2 and 4.
To simplify notations we sometimes use the $\bullet$-operator. For all $t \in T$ and $p \in P$: $\bullet t = \{k \in P \mid k \in I_t\}$, $t\bullet = \{k \in P \mid \exists_{x \in INT} \langle k, x \rangle \in O_t\}$, $\bullet p = \{v \in T \mid \exists_{x \in INT} \langle p, x \rangle \in O_v\}$ and $p\bullet = \{v \in T \mid p \in I_v\}$.

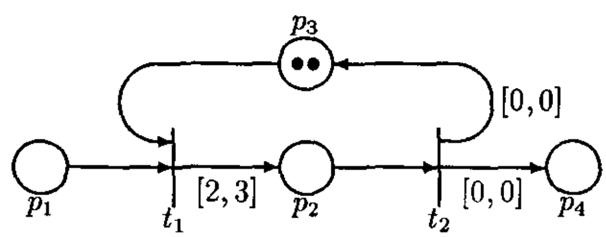


Figure 1: An ITPN for a queuing system

Figure 1 shows an ITPN for a queuing system with two servers. Service times are between 2 and 3 minutes. In this example: $P = \{p_1, p_2, p_3, p_4\}$, $T = \{t_1, t_2\}$, $I_{t_1} = [p_1, p_3]$, $I_{t_2} = [p_2]$ and $O_{t_1} = [\langle p_2, \langle 2,3 \rangle\rangle]$, $O_{t_2} = [\langle p_3, \langle 0,0 \rangle\rangle, \langle p_4, \langle 0,0 \rangle\rangle]$.

2.1 Semantics of an ITPN

We describe the semantics of an ITPN by a *transition system*. This transition system is also used to prove the correctness of our second analysis method (see section 4),. This brings on some heavy notation, so the casual reader is advised to skim through this section

and section 4. A transition system is a pair $\langle S, R\rangle$ where S is the *state space* and $R \subseteq S \times S$ the *transition relation.* If $s_1, s_2 \in S$ the $s_1 R s_2$ means that a transition from state s_1 to s_2 is possible.
In the transition system describing an ITPN we attach an unique label to every token (in addition to the timestamp). Id is an infinite set of *token labels.* The state space of the transition system is $S = Id \not\rightarrow (P \times TS)$ [1]. So, in fact, a state $s \in S$ is a set of triples representing identity, location and timestamp, and the first one is unique. If $s \in S$ then $dom(s)$ is the set of token labels corresponding to the tokens in the net. If $i \in dom(s)$ then $s(i)$ is a pair representing the *position* and *timestamp* of the corresponding token. Note that we speak about *state* rather than *marking* because a token has a position *and* a timestamp.
To switch between the bag and the set representation, we introduce the functions $\mathcal{BS}$ and $\mathcal{SB}$.

Definition 2
If A is a set then we define $\mathcal{BS} \in \mathbb{B}(A) \rightarrow (Id \not\rightarrow A)$ and $\mathcal{SB} \in (Id \not\rightarrow A) \rightarrow \mathbb{B}(A)$ such that:

$$\forall_{s \in Id \not\rightarrow A}\ \mathcal{SB}(s) = \lambda_{a \in A} \#\{i \in dom(s) \mid s(i) = a\}$$
$$\forall_{s \in \mathbb{B}(A)} \mathcal{SB}(\mathcal{BS}(s)) = s$$

It is easy to verify that these functions exist ('Axiom of Choice'). To sample a delay from the delay interval specified by O we introduce the concept of *specialisation.* But let us start with some useful notations; if $q \in P \times TS$ then $place(q) = \pi_1(q)$ [2] and $time(q) = \pi_2(q)$. If $\overline{q} \in P \times INT$ then $place(\overline{q}) = \pi_1(\overline{q})$ and $time(\overline{q}) = \pi_2(\overline{q})$.

Definition 3 (Specialisation)
For $s \in Id \not\rightarrow (P \times TS)$ and $\overline{s} \in Id \not\rightarrow (P \times INT)$:
$s \triangleleft \overline{s}\ \equiv$ there exists a bijective function $f \in dom(s) \rightarrow dom(\overline{s})$ with: [3]

$$\forall_{i \in dom(s)}\ place(s(i)) = place(\overline{s}(f(i)))\ \wedge\ time(s(i)) \in time(\overline{s}(f(i)))$$

If $s \triangleleft \overline{s}$ then every token in s corresponds to a token in $\overline{s}$ (and vica versa) such that they are in the same place and the timestamp of the token in s is an element of the time interval of the token in $\overline{s}$.

The transition system
An ITPN $= (P, T, I, O, TS)$ defines a *transition system* $\langle S, R\rangle$, with a state space S and a transition relation R:

- $S = Id \not\rightarrow (P \times TS)$, the state space

[1] $A \not\rightarrow B$ is the set of all partial functions from set A to set B.
[2] If $x = \langle x_1, x_2, ..x_n\rangle \in A_1 \times A_2 \times .. \times A_n$ then for all $i \in 1..n$ $\pi_i(x) = x_i$.
[3] If $x \in TS$ and $v \in INT$ then $x \in v \equiv \pi_1(v) \leq x \leq \pi_2(v)$.

- $E = T \times S \times S$, event set
- $AE(s) =$

$$
\begin{array}{llr}
\{ & \langle t, q_{in}, q_{out} \rangle \in E \mid q_{in} \subseteq s \ \wedge & (1)\\
& I_t = \lambda_{p \in P} \ \#\{i \in dom(q_{in}) \mid place(s(i)) = p\} \ \wedge & (2)\\
& \forall_{i \in dom(q_{in})} \forall_{j \in dom(s) \setminus dom(q_{in})} \ place(s(i)) = place(s(j)) \Rightarrow time(s(i)) \leq time(s(j)) \ \wedge & (3)\\
& dom(q_{out}) \cap dom(s) = \emptyset \ \wedge & (4)\\
& q_{out} \lhd \mathcal{BS}(O_t) \ \} & (5)
\end{array}
$$

  , set of allowed events in state $s \in S$

- $et(e) = \max_{i \in dom(\pi_2(e))} time(\pi_2(e)(i))$, event time of $e \in E$
- $tt(s) = \min_{e \in AE(s)} et(e)$, transition time in $s \in S$
- $scale(q, x) = \lambda_{i \in dom(q)} \langle place(q(i)), time(q(i)) + x \rangle$, scales timestamps in $q \in S$ with $x \in TS$
- Finally the transition relation R is defined by: [4]

$$s_1 R s_2 \equiv \exists_{\substack{e \in AE(s_1) \\ et(e) = tt(s_1)}} \ s_2 = s_1 \upharpoonright (dom(s_1) \setminus \pi_2(e)) \ \cup \ scale(\pi_3(e), tt(s_1))$$

  , $s_1, s_2 \in S$

We call the firing of a transition an *event*. An event changes a state into a new state, described by the transition relation. An event e is a triple indicating the transition that fires ($\pi_1(e)$), the tokens consumed ($\pi_2(e)$) and the tokens produced ($\pi_3(e)$). $AE(s)$ is the set of allowed events in state s. Such an event satisfies 5 conditions. The first conditions is about the requirement that consumed tokens have to exist. The transition that fires consumes the correct number of tokens from the input places (condition (2)). Tokens are consumed in order of their timestamps (condition (3)). Produced tokens bear a unique label, condition (4) checks whether the label of a produced token does not exist already. The delay of a produced token is sampled from the corresponding delay interval (condition (5)). Note that we use the specialisation concept to state that the delays are sampled from the delay intervals of O_t. The function $\mathcal{BS}$ is used to convert the bag O_t into a partial function (i.e. $\mathcal{BS}(O_t) \in S$).
The *event time* of an event e is the maximal timestamp of the tokens consumed ($et(e)$). The *transition time* of a state is the minimum of the event times of the allowed events. If there are two or more events with an event time equal to the transition time, then these

[4] If $f \in A \not\rightarrow B$ and $X \subseteq A$ then $f \upharpoonright X = \lambda_{i \in X \cap dom(f)} f(i)$.

events are in *conflict*. These conflicts are resolved non-deterministically. The timestamps of the produced tokens are rescaled using the function *scale*.

Our model has a great descriptive power compared to other TPN-models. There are two notable differences with conventional Timed Petri Net models.

The first difference is the fact that time is in tokens and each token bears an unique label. This we adopted from [Hee et al. 91]. This results in a transparent semantics and a very compact state representation. Firing is an atomic action; if a transition fires it is immediately available for a new firing (if it is enabled). The produced tokens are unavailable for some period specified by the delay interval, this can be interpreted as the time it takes before a token arrives in the output place.
It is also possible to model a transition being busy for a while. We call such a transition a *timed transition*, it removes tokens, withholds them for some time before tokens appear in the output places (see [Zuberek 80]). Note that a busy (timed) transition cannot accept new tokens. If we think of a timed transition as a single server in a queuing network then a transition in the ITPN model represents an infinite server. Figure 2 shows how to model a timed transition. The timed transition (represented by a box) is replaced by two transitions and two places. Transition t^{start} removes tokens from the input places and puts a token in the place t^{busy} with a delay representing the time the transition is busy. There is always a token in t^{busy} or in t^{free} (but not in both) indicating whether the transition is busy of free. Transition t^{end} represents the termination of the firing. Note that tokens in the original places are always available (i.e. they have a timestamp smaller or equal to the previous transition time).
This construction shows that time in tokens is a very powerful concept. A similar concept was used in [Sifakis 78] where time is associated with places.

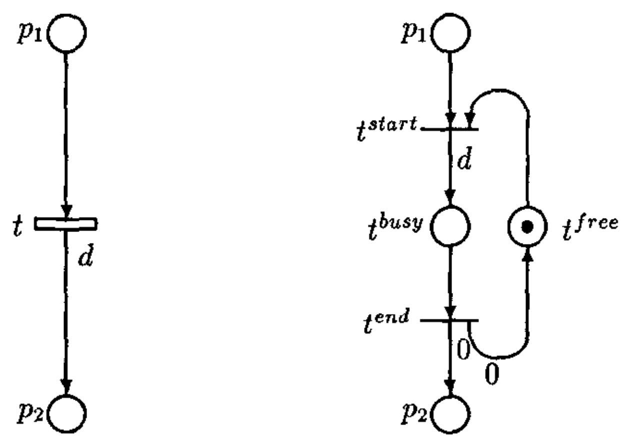


Figure 2: Construction of a timed transition (left) using two ITPN transitions (right)

The second difference is the fact that the firing delay is non-deterministic *and* non-stochastic. Specifying the delay by means of an interval rather than deterministic or stochastic variables allows for the representation of time constraints. This is very important when modelling time-critical systems. Examples of such systems are real-time (computer) systems and just-in-time manufacturing systems.

2.2 Interesting questions

Reachability is the basis for studying the behaviour of a system. Given a transition system $\langle S, R\rangle$ describing the semantics of an ITPN, a state s_2 is said to be *immediate reachable* from s_1 if and only if $s_1 R s_2$.

Definition 4 (Reachability)
For $s \in S$:
$R(s) = \{\hat{s} \in S \mid sR\hat{s}\}$ is the *one step reachability set*,
$R^n(s) = \{\hat{s} \in S \mid sR^n\hat{s}\}$ is the *n-step reachability set* of $s \in S$.
$RS(s) = \bigcup_{n \in \mathbb{N}} R^n(s)$ [5] is the set of all states that are reachable from s.
$S^T = \{s \in S \mid R(s) = \emptyset\}$ is the set of *terminal states*.

The *process* of an ITPN is described by the set of all possible *paths* (given a set of initial states). A path is a sequence of states such that any successive pair belongs to the transition relation. A path starts in an initial state and either it is infinite or it ends in a terminal state.

Definition 5 (Process)
If $A \subseteq S$ is a set of initial states then:

$$\begin{aligned}\Pi(A) = \{\sigma \in \mathbb{N} \not\twoheadrightarrow S \quad | \quad & 0 \in dom(\sigma) \ \wedge \ \sigma_0 \in A \\ & \wedge \ \forall_{i \in dom(\sigma) \setminus \{0\}} \ (i-1) \in dom(\sigma) \ \wedge \ \sigma_{i-1} R \sigma_i \\ & \wedge \ \forall_{i \in dom(\sigma)} \ (\forall_{j \in dom(\sigma)} j \leq i) \Rightarrow \sigma_i \in S^T \ \}\end{aligned}$$

represents the process (or behaviour) of the ITPN. It is the set of all possible paths for an ITPN with an initial state in A. Π is also defined for a single initial state $s \in S$; $\Pi(s) = \Pi(\{s\})$.

For all paths $\sigma \in \Pi(A)$ and $n \in \mathbb{N}$; $\sigma \restriction \{k \in \mathbb{N} \mid 0 \leq k < n\}$ is called a *firing sequence* (or *trace*).
A path is a sequence of states. Consider the path $s_0, s_1, ..s_{i-1}, s_i, s_{i+1}, \ldots$ At time $tt(s_{i-1})$ an event occurred transforming state s_{i-1} into s_i. At time $tt(s_i)$ an event occurred transforming state s_i into s_{i+1}. Between $tt(s_{i-1})$ and $tt(s_i)$ the system was in state s_i. We are often interested in the state *at a certain moment in time.*

[5] $\mathbb{N} = \{0, 1, 2, ..\}$

Definition 6 (State function)
If $A \subseteq S$ and $\sigma \in \Pi(A)$ then $F(\sigma) \in TS \to \mathbb{B}(P)$ with

$$\forall_{x \in TS}\ F(\sigma)(x) = \sigma_{\min\{i \in dom(\sigma)\ \mid\ x \leq tt(\sigma_i)\}}$$

is the *state function* of path σ.

Figure 3 shows the relation between a path and the corresponding state function.

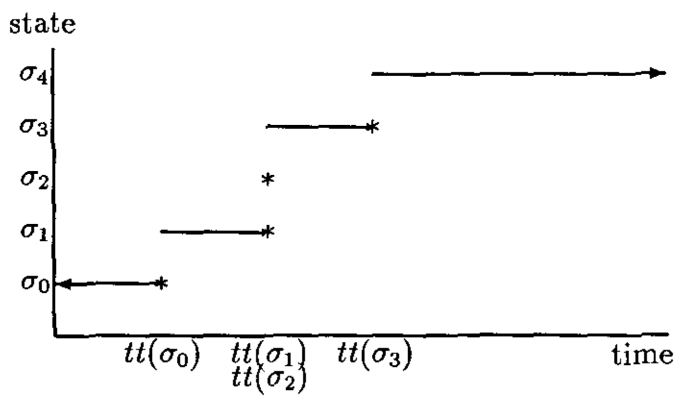


Figure 3: Relation between a path and the corresponding state function

Sometimes we are only interested in the position of a token and not in its timestamp. This leads to the definition of the *marking* of a state. A marking is denoted as a multiset of place indices. Function $M \in S \to \mathbb{B}(P)$ gives the marking of each state. If $s \in S$ then $M(s) = \lambda_{p \in P}\ \#\{i \in dom(s) \mid place(s(i)) = p\ \}$. For example if $s \in S$ and $p \in P$ then $M(s)(p) = 3$ means that there are three tokens in place p.

If one models systems where time aspects are important one is often interested in characteristics like throughput and response times. This is the reason we defined the *earliest* and *latest first arrival time* for a place in the net. To define these we need the operation *place projection* ($\upharpoonright$), returning the bag of timestamps of tokens in a certain place p given a state s. For $s \in S$, $p \in P$; $s \upharpoonright p = \lambda_{x \in TS}\ \#\{i \in dom(s) \mid s(i) = \langle p, x \rangle\}$. So, $\min(s \upharpoonright p)$ is the smallest timestamp of all tokens in place p.

Definition 7 ($\mathcal{EAT}, \mathcal{LAT}$)
Given an ITPN, a state $s \in S$ and a place $p \in P$ we define:

$$\begin{aligned}\mathcal{EAT}(s,p) &= \min_{\sigma \in \Pi(s)} \min_{i \in dom(\sigma)} \min(\sigma_i \upharpoonright p) \\ \mathcal{LAT}(s,p) &= \max_{\sigma \in \Pi(s)} \min_{i \in dom(\sigma)} \min(\sigma_i \upharpoonright p)\end{aligned}$$

for the *earliest arrival time* and the *latest first arrival time* respectively.

Note that $\mathcal{EAT}(s,p)$ and $\mathcal{LAT}(s,p)$ are only defined for the first token to arrive in p. It is possible to generalise these arrival times for a set of initial states $A \subseteq S$ and n tokens:

$$\begin{aligned}\mathcal{EAT}_n(A,p) &= \min_{\sigma\in\Pi(A)}\min_{i\in dom(\sigma)} \min_n(\sigma_i \restriction p)\\ \mathcal{LAT}_n(A,p) &= \max_{\sigma\in\Pi(A)}\min_{i\in dom(\sigma)} \min_n(\sigma_i \restriction p)\end{aligned}$$

where $\min_n b = \min_{\hat{b}\subseteq b \,\wedge\, \#\hat{b}=n}(\max \hat{b})$.
If a bag $b \in \mathbb{B}(TS)$ contains at least n elements then $\min_n b$ is the n^{th} timestamp in the bag (selected in ascending order) otherwise $\min_n b$ is infinite.
If $\mathcal{EAT}_n(A,p) \leq x$ then there exists a path starting in a state $s \in A$ that visits a state with at least n tokens in p with a timestamp less or equal to x. If $\mathcal{LAT}_n(A,p) \geq x$ then there exists a path such that all states visited by this path do not have n tokens in p with a timestamp smaller than x. If p is a *sink* place (i.e. $p\bullet = \emptyset$) then $\mathcal{EAT}_n(A,p)$ is the *earliest n^{th} arrival time* and $\mathcal{LAT}_n(A,p)$ is the *latest n^{th} arrival time.*
We use $\mathcal{EAT}$ and $\mathcal{LAT}$ to measure things like throughput times and response times. Another interesting characteristic of a system is the utilisation of a resource; for example the occupation rate of a machine or the stock level in a distribution centre. This characteristic is closely related to the number of tokens in a certain place during the execution of the net.

Sometimes it is useful to know the *maximum* number of tokens in a place. A place $p \in P$ is *K-bounded* in $s \in S$ if the number of tokens in p cannot exceed an integer K. More formally $\forall_{\hat{s}\in RS(s)}\ \#(\hat{s} \restriction p) \leq K$. A net is called K-bounded in $s \in S$ if all places are K-bounded in s. Nets that are 1-bounded are called *safe*. Places are often used to represent buffers. By verifying that the net is bounded or safe, it is guaranteed that there will be no overflows of the buffers, no matter what firing sequence is taken.

We are also interested in the average number of tokens in a place. Because our model is non-deterministic we define the average number of tokens in a place given a path.

Definition 8 (U)
If $s \in S$ and $\sigma \in \Pi(s)$, $p \in P$ and $t \in TS$ then

$$U(\sigma,p,t)(x) = \frac{1}{t}\int_0^t M(F(\sigma)(x))(p)\ \lambda(d\,x)$$

is the *average number of tokens* in p during $[0,t]$, where λ is the Lebesque measure.

Now we are able to define a lower and an upper bound for the *occupation rate* of a place.

Definition 9 ($\mathcal{LOR}, \mathcal{HOR}$)
If $s \in S$, $p \in P$ and $t \in TS$ then we define:

$$\begin{aligned}\mathcal{LOR}(s,p,t) &= \min_{\sigma\in\Pi(s)} U(\sigma,p,t)\\ \mathcal{HOR}(s,p,t) &= \max_{\sigma\in\Pi(s)} U(\sigma,p,t)\end{aligned}$$

for the *lowest occupation rate* and *highest occupation rate* respectively.

Given an initial state s the average number of tokens in p during $[0, t]$ is between $\mathcal{LOR}(s, p, t)$ and $\mathcal{HOR}(s, p, t)$. This allows us to analyse logistical concepts like machine utilisation and stock levels.

A net is said to be *monotonous* for an initial state $s \in S$ if the time in the net is always increasing. Sometimes this property is too strong. Thus, we relax the liveness condition and define *livelock free*. A net is livelock free for an initial state if the time in the net is increasing or a terminal state is encountered. If the execution of the net always stops after a number of events then we say that the net is *dead*. A net is *progressive* if any time from TS can and will be reached. Because progressiveness is a new concept we give a formal definition.

Definition 10
For an initial state $s \in S$ an ITPN is said to be *progressive* in s if and only if

$$\forall_{x \in TS} \, \forall_{\sigma \in \Pi(s)} \, \exists_{i \in dom(\sigma)} \, tt(\sigma_i) > x$$

With Interval Timed Petri Nets one is able to model a large variety of systems. The major strength of these nets is the natural representation of concurrency and timing aspects. However modelling itself is of little use. Analysis of the modelled system should be the main goal. Analysis techniques help the modeller to gain insight into the dynamic behaviour of the system.
In this paper we will present four analysis techniques for ITPN.

3 Method ATCFN

The first analysis method we present is called *Arrival Times in Conflict Free Nets* (ATCFN). Suppose we have a conflict free[6] progressive ITPN where all input arcs have multiplicity 1. In this case we give a linear time (in the size of the net) algorithm to find $\mathcal{EAT}$ and $\mathcal{LAT}$ given a place p and an initial state s. If we consider an ITPN that does not satisfy these restrictions (conflict free, progressive, multiplicity 1) then the results of this algorithm can be interpreted as lower bounds for $\mathcal{EAT}$ and $\mathcal{LAT}$.

There is some similarity with "the Dijkstra algorithm" to calculate the shortest path ([Dijkstra 59]) and the calculation of earliest event times in activity networks (CPM,PERT) ([Price 71]). It is in fact an extension with two node types: transitions and places. See appendix C for more information on this subject.

To describe the algorithm we have to quantify the relation between a transition and an output place.

[6] A net is conflict free if for all $p \in P$; $\#(p\bullet) \leq 1$.

Definition 11 (D^{min}, D^{max})
Given an ITPN, a transition t and a place p:

$$\begin{aligned} D^{min}(t,p) &= \min\{x \mid \langle p, \langle x,y\rangle\rangle \in O_t\} \\ D^{max}(t,p) &= \min\{y \mid \langle p, \langle x,y\rangle\rangle \in O_t\} \end{aligned}$$

$D^{min}(t,p)$ ($D^{max}(t,p)$) is the minimal (maximal) time between the firing of t and the arrival of the first token in p corresponding to this firing. An interpretation of $D^{min}(t,p)$ ($D^{max}(t,p)$) is the minimal (maximal) distance between a transition t and a place p.

First we consider the algorithm to calculate $\mathcal{EAT}$ given an initial state $s \in S$. In this algorithm we assign a label to every place. There are two kinds of labels; *permanent* and *tentative* labels. A label has a (time) value indicating the earliest arrival of the first token in the corresponding place.
We represent the set of places bearing a permanent label by X_p and the set of places bearing a tentative label by X_t. The value of a label is given by d^{min}. For a place p with a permanent label, $d^{min}(p)$ is the earliest arrival time of a token in p. If $p \in X_t$ then $d^{min}(p)$ is the earliest arrival time found so far.

Algorithm

step 1 Assign a tentative label to every place ($X_t = P$, $X_p = \emptyset$). For every place p, the (time) value is set to the smallest timestamp of the tokens initially present in p. If, initially, there are no tokens in p then the value of the label is set to ∞. In other words: $d^{min}(p) = \min(s \upharpoonright p)$.

step 2 If there are no places with a tentative label and a finite value then terminate. Otherwise select a place p with a tentative label and the smallest value ($p \in X_t$ and $d^{min}(p) = \min\{d^{min}(l) \mid l \in X_t\}$). Declare the label of p to be permanent instead of tentative.

step 3 Consider all transitions t satisfying the following conditions: p is an input place of t and all input places bear a permanent label ($t \in p\bullet$ and $\bullet t \subseteq X_p$).

For every t consider all output places k that bear a tentative label ($k \in (t\bullet) \cap X_t$). If the value of the label attached to k is greater then the sum of the value of the label attached to p and $D^{min}(t,k)$ then change the value of the label attached to k to $d^{min}(p) + D^{min}(t,k)$.

If all transitions t with the corresponding output places k have been considered go to step 2.

Alternatively we can give a more compact description of the algorithm using pseudo-code.

input ITPN,s

$X_t := P$;
$X_p := \emptyset$;
for $p \in P$ **do** $d^{min}(p) = \min s \Uparrow p$ **end**;

while $\min\{d^{min}(l) \mid l \in X_t\} < \infty$
  do
  select $p \in X_t$ **with** $d^{min}(p) = \min\{d^{min}(l) \mid l \in X_t\}$;
  $X_t := X_t \setminus \{p\}$;
  $X_p := X_p \cup \{p\}$;
  for $t \in \{v \in p\bullet \mid \bullet v \subseteq X_p\}$
    do
    for $k \in (t\bullet) \cap X_t$
      do
      $d^{min}(k) := d^{min}(k) \min (d^{min}(p) + D^{min}(t,k))$;
      end;
    end;
  end;

output X_t, X_p, d^{min}

The algorithm to calculate $\mathcal{LAT}$ is similar: D^{min} and d^{min} are replaced by D^{max} and d^{max}.

Theorem 1
For a conflict free, progressive ITPN where all input arcs have multiplicity 1, a place $p \in P$ and an initial state $s \in S$:

$$\begin{aligned} d^{min}(p) &= \mathcal{EAT}(s,p) \\ d^{max}(p) &= \mathcal{LAT}(s,p) \end{aligned}$$

Proof.
We prove this theorem by showing that there exists an invariant and a termination argument. The outer loop in the algorithm satisfies four invariant relations:

$Q1$: $X_p \cup X_t = P$ and $X_p \cap X_t = \emptyset$

$Q2$: $\forall_{k \in X_p} \forall_{l \in X_t}\, d^{min}(k) \leq d^{min}(l)$

$Q3$: $\forall_{k \in X_p}\, d^{min}(k) = \mathcal{EAT}(s,k)$

$Q4$: $\forall_{k \in X_t}\, d^{min}(k) = (\min s \Uparrow k) \min (\min_{\substack{v \in T \\ \bullet v \subseteq X_p}} \max_{l \in \bullet v} d^{min}(l) + D^{min}(v,k))$

It is easy to show that these invariants hold after initialisation.
If an element p is transferred form X_t to X_p then $Q1$ still holds and because p is a minimal element $Q2$ also holds.
$Q3$ also holds because $d^{min}(p) = \mathcal{EAT}(s,p)$, this follows from $Q2$,$Q3$ and $Q4$. To prove this observe the subexpression $(\min_{\substack{v\in T \\ \bullet v\subseteq X_p}} \max_{l\in\bullet v} d^{min}(l) + D^{min}(v,k))$ of Q4. Because all input places l are permanent, $d^{min}(l) = \mathcal{EAT}(s,l)$ (use Q3). It is sufficient to consider only transitions with permanent input places because all transitions having a tentative input place do not fire before $d^{min}(p)$ (use Q2). Furthermore, a transition v will fire at its enabling time because the net is conflict free and progressive. Therefore this subexpression evaluates to the smallest possible timestamp of a token in p produced by any transition.
If the smallest possible timestamp of a token in p was not produced by a transition then it was initially there; $\mathcal{EAT}(s,p) = (\min s \Downarrow p)$. Using Q4 this implies that $d^{min}(p) = \mathcal{EAT}(s,p)$ (i.e. Q3 holds).
Invariant $Q4$ is violated by the transfer of p from X_t to X_p. This is repaired by the two inner for loops.
The algorithm terminates because the number of elements in X_t is decreasing. The remaining places in X_t are not reachable.
An analogous proof holds for the upper bound of the first arrival.
□

This theorem tells us that the algorithm can be used to calculate $\mathcal{EAT}$ and $\mathcal{LAT}$ for a restricted class of nets. A serious restriction is the fact that conflicts between transitions are not allowed. If there are conflicts in the net, for example to model shared resources, the algorithm can give incorrect results. It is however possible to model certain kinds of parallelism and synchronisation without having conflicts.
If the ITPN does not satisfy the conditions mentioned in theorem 1 then $d^{min}(p) \leq \mathcal{EAT}(s,p)$ and $d^{max}(p) \leq \mathcal{LAT}(s,p)$ (for an arbitrary ITPN a place p and an initial state $s \in S$). In other words: the algorithm produces lower bounds for $\mathcal{EAT}$ and $\mathcal{LAT}$. For an arbitrary net, the first token in place p does not arrive before $d^{min}(p)$ and it is possible to construct a firing sequence where the first token does not arrive before $d^{max}(p)$.

The most serious drawback is that this approach only produces statements about the arrival time of the first token in a place. In general we are interested how the system performs under a specific workload and therefore equally interested in the subsequent tokens. We also want to verify dynamic properties such as liveness and boundedness. This is the reason we developed a more general solution described in the following section.

4 Method MTSRT

The second analysis technique we present is called *Modified Transition System Reduction Technique* (MTSRT). In section 2.1 we saw that the semantics of an ITPN are described by a transition system. In this transition system we attach an unique label to every token. The state space of the transition system is therefore $S = Id \not\rightarrow (P \times TS)$ (Id is the set of token labels).
Calculating the set of reachable states is (generally) impossible because the firing delay of a token is sampled from an interval. In general there is an infinite number of allowed firing delays, all resulting in a different state.

For computational reasons we define a modified model; the $\overline{ITPN}$ model. The semantics of this modified model are described by a *modified transition system* ($< \overline{S}, \overline{R} >$). In this transition system we attach a time *interval* to every token instead of a timestamp, $\overline{S} = Id \not\rightarrow (P \times INT)$. One can think of this transition system as some kind of generalisation of the the original transition system describing the semantics of an ITPN.

After a formal definition of the modified transition system we will show how these two transition systems relate to each other. We will see that we can use the modified transition system to answer questions about the original transition system and, therefore, about the behaviour of the ITPN.
Before giving a description of the modified transition system we define the relation ($\leq_i$) to compare intervals.

Definition 12 ($\leq_i$)
If $v, w \in INT$ then: $v \leq_i w \;\equiv\; (\pi_1(v) \leq \pi_1(w)) \;\wedge\; (\pi_2(v) \leq \pi_2(w))$

Note that $\leq_i$ defines a partial ordering because $\leq_i$ is reflexive, antisymmetric and transitive. Sometimes we use the notation $v <_i w$ to denote that $v \leq_i w$ and $v \neq w$. If $\overline{q} \in P \times INT$ then $place(\overline{q}) = \pi_1(\overline{q})$, $time(\overline{q}) = \pi_2(\overline{q})$, $time^{min}(\overline{q}) = \pi_1(time(\overline{q}))$ and $time^{max}(\overline{q}) = \pi_2(time(\overline{q}))$.

The modified transition system

An ITPN $= (P, T, I, O, TS)$ defines a *modified transition system* $\langle \overline{S}, \overline{R} \rangle$, with a state space $\overline{S}$ and a transition relation $\overline{R}$:

- $\overline{S} = Id \not\rightarrow (P \times INT)$, the state space
- $\overline{E} = T \times \overline{S} \times \overline{S}$, event set
- $\overline{AE}(s) =$

$$\{ \;\; \langle t, q_{in}, q_{out} \rangle \;\in\; \overline{E} \;\mid\; q_{in} \subseteq s \;\wedge \tag{1}$$

$$I_t = \lambda_{p \in P} \; \#\{ i \in dom(q_{in}) \mid place(s(i)) = p \} \;\; \wedge \tag{2}$$

$$\forall_{i \in dom(q_{in})} \forall_{j \in dom(s) \setminus dom(q_{in})} \; place(s(i)) = place(s(j)) \Rightarrow \neg(time(s(j)) <_i time(s(i)) \quad (3)$$
$$dom(q_{out}) \cap dom(s) = \emptyset \;\wedge \quad (4)$$
$$\mathcal{SB}(q_{out}) = O_t \;\} \quad (5)$$

, set of allowed events in state $s \in \overline{S}$

- $et^{min}(e) = \max_{i \in dom(\pi_2(e))} time^{min}(\pi_2(e)(i))$, lower bound event time of $e \in \overline{E}$
- $et^{max}(e) = \max_{i \in dom(\pi_2(e))} time^{max}(\pi_2(e)(i))$, upper bound event time of $e \in \overline{E}$
- $tt^{min}(s) = \min_{e \in \overline{AE}(s)} et^{min}(e)$, lower bound transition time in $s \in \overline{S}$
- $tt^{max}(s) = \min_{e \in \overline{AE}(s)} et^{max}(e)$, upper bound transition time in $s \in \overline{S}$
- $\overline{scale}(q, x, y) = \lambda_{i \in dom(q)} \langle place(q(i)), \langle time^{min}(q(i)) + x, time^{max}(q(i)) + y \rangle \rangle$, scales timestamps, $q \in \overline{S}$ and $x, y \in TS$
- Finally the transition relation $\overline{R}$ is defined by:

$$s_1 \overline{R} s_2 \equiv \exists_{\substack{e \in \overline{AE}(s_1) \\ et^{min}(e) \leq tt^{max}(s_1)}} \; s_2 = s_1 \restriction (dom(s_1) \setminus \pi_2(e)) \;\cup\; \overline{scale}(\pi_3(e), et^{min}(e), tt^{max}(s_1))$$

, $s_1, s_2 \in \overline{S}$

Similar to R,RS and Π we define $\overline{R}$, $\overline{RS}$ and $\overline{\Pi}$. Symbols superimposed with a horizontal line are associated with the modified transition system.

An event e is a triple indicating the transition that fires ($\pi_1(e)$), the tokens consumed ($\pi_2(e)$) and the tokens produced ($\pi_3(e)$). $\overline{AE}(s)$ is the set of allowed events in state s. Such an event satisfies 5 conditions. The first condition is about the requirement that consumed tokens have to exist. The transition that fires consumes the correct number of tokens from the input places (condition (2)). Tokens are consumed in order of their timestamps (condition (3)). Produced tokens bear a unique label and their delay interval is as specified (condition (4) and (5)).
The event time of an event e in isolation is between $et^{min}(e)$ and $et^{max}(e)$. The first event in state s_1 will occur between $tt^{min}(s_1)$ and $tt^{max}(s_1)$. An allowed event e in state s_1 will occur between $et^{min}(e)$ and $tt^{max}(s_1)$ (if it occurs). Therefore we have to rescale the (relative) intervals of the produced tokens using the function $\overline{scale}$. This function adds $et^{min}(e)$ to the lower bound of the delay interval and adds $tt^{max}(s_1)$ to the upper bound of the delay interval.

Note the resemblance with the original transition system described in section 2.1.

To give an impression of the modified transition system, consider the net shown in figure 4. Initially there is one token in place $p1$ with an interval of $[0, 3]$, there is one token in $p2$

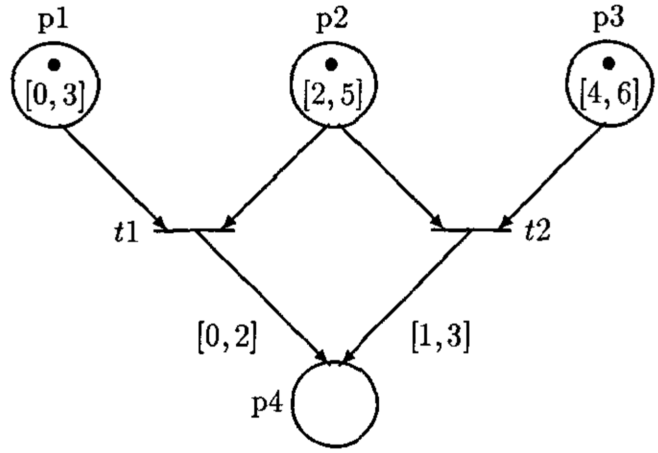


Figure 4: An example used to describe the modified model

with an interval of $[2,5]$ and there is one token in $p3$ with an interval of $[4,6]$. Note that this state in the modified model corresponds to an infinite number of states in the original model, for instance the state with a token in $p1$ with timestamp 2.4 and a token in $p2$ with timestamp 2.118 and a token in $p3$ with timestamp 5.22.
There are two allowed events; event e_1 is the firing of $t1$ while consuming the tokens in $p1$ and $p2$, event e_2 is the firing of $t2$ while consuming the tokens in $p2$ and $p3$. The event time of e_1 is between 2 ($et^{min}(e_1)$) and 5 ($et^{max}(e_1)$), the event time of e_2 is between 4 ($et^{min}(e_2)$) and 6 ($et^{max}(e_2)$). All events having a lower bound for the event time (et^{min}) smaller or equal to the upper bound of the transition time (tt^{max}) can happen. If e_1 occurs it will be between 2 ($et^{min}(e_1)$) and 5 ($tt^{max}(s)$), if e_2 occurs it will be between 4 and 5. In both cases a token is produced for place p4. There are two possible terminal states: one with a token in $p3$ and $p4$ and one with a token in $p1$ and $p4$. In the first case the time interval of the token in $p4$ is $[2,7]$, because the delay interval of a token produced by $t1$ is $[0,2]$. In the second case the time interval of the token in $p4$ is $[5,8]$. Using intervals rather than timestamps prevented us from having to consider all possible delays between $[0,2]$ or $[1,3]$, i.e. it suffices to consider upper and lower bounds.

The following theorem shows that the upper and lower bounds of the transition times are 'non-decreasing'. This property is called 'the monotonicity of time', i.e. time can only move forward.

Theorem 2
For an ITPN with states $s_1, s_2 \in \overline{S}$ such that $s_2 \in \overline{R}(s_1)$; $tt^{min}(s_1) \leq tt^{min}(s_2)$ and $tt^{max}(s_1) \leq tt^{max}(s_2)$.

Proof.
Because $s_2 \in \overline{R}(s_1)$ there exists an event $e \in \overline{AE}(s_1)$ such that $et^{min}(e) \leq tt^{max}(s_1)$ and $s_2 = s_1 \restriction (dom(s_1) \setminus \pi_2(e)) \cup \overline{scale}(\pi_3(e), et^{min}(e), tt^{max}(s_1))$. The definition of $\overline{scale}$ tells us that the lower bound of the produced token is at least $et^{min}(e)$ and the upper bound

is at least $tt^{max}(s_1)$. Therefore for all new events $h \in \overline{AE}(s_2) \setminus \overline{AE}(s_1)$ we find that $et^{min}(h) \geq et^{min}(e) \geq tt^{min}(s_1)$ and $et^{max}(h) \geq tt^{max}(s_1)$. All events that where already enabled also have a lower bound event time of at least $tt^{min}(s_1)$ and an upper bound event time of at least $tt^{max}(s_1)$. By the definition of tt^{min} and tt^{max} we conclude that $tt^{min}(s_1) \leq tt^{min}(s_2)$ and $tt^{max}(s_1) \leq tt^{max}(s_2)$.
□

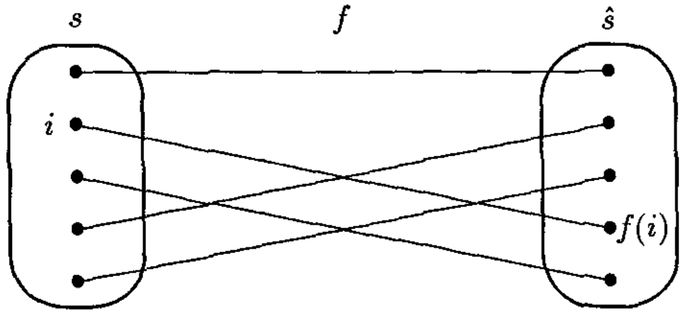


Figure 5: Specialisation: $s \lhd \overline{s}$

In section 2.1 we introduced the concept of specialisation, this allows us to compare states of the original transition system with states of the modified transition system. Suppose $s \in S$, $\overline{s} \in \overline{S}$ and $s \lhd \overline{s}$ then there exists a bijective function $f \in dom(s) \rightarrow dom(\overline{s})$ such that every token with label $i \in dom(s)$ corresponds to a token with label $f(i) \in dom(\overline{s})$ that is in the same place and has an interval containing the timestamp of i (see figure 5). The concept of specialisation also allows for the definition of *soundness and completeness of the relation between the two transition systems.* Soundness means that all transitions possible in $\langle S, R\rangle$ are also possible in $\langle \overline{S}, \overline{R}\rangle$. Completeness means that all transitions possible in $\langle \overline{S}, \overline{R}\rangle$ are also possible in $\langle S, R\rangle$.

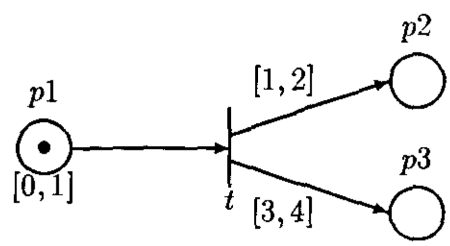


Figure 6: Non-completeness caused by dependencies

Completeness does not hold, this is caused by the fact that *dependencies* between tokens are not taken into account. Consider for example the net shown in figure 6. 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 ($tt^{max}(s)$). The next state in the modified transition system 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 (in the original transition system) because these timestamps are related (i.e. they where produced at the same time).

Fortunately, soundness holds.

Theorem 3 (Soundness)
For all $s_1 \in S$ and $\overline{s}_1 \in \overline{S}$ such that $s_1 \lhd \overline{s}_1$: $\quad \forall_{s_2 \in R(s_1)} \exists_{\overline{s}_2 \in \overline{R}(\overline{s}_1)} \; s_2 \lhd \overline{s}_2$

Proof.
Because $s_1 \lhd \overline{s}_1$ there exists a specialisation function f.
Suppose $s_2 \in R(s_1)$, then there is an event e such that:

(i) $e \in AE(s_1)$

(ii) $et(e) = tt(s_1)$

(iii) $s_2 = s_1 \upharpoonright (dom(s_1) \setminus \pi_2(e)) \cup scale(\pi_3(e), tt(s_1))$

Define: $\overline{e} = \langle \pi_1(e), \overline{s}_1 \upharpoonright f(dom(\pi_2(e))), q \rangle \in \overline{E}$ where $q \in \overline{S}$ such that conditions (5) and (6) hold and $\overline{s}_2 = \overline{s}_1 \upharpoonright (dom(\overline{s}_1) \setminus dom(\pi_2(\overline{e}))) \cup \overline{scale}(\pi_3(\overline{e}), et^{min}(\overline{e}), tt^{max}(\overline{s}_1))$

Now we have to prove that:

(i) $\overline{e} \in \overline{AE}(\overline{s}_1)$

(ii) $et^{min}(\overline{e}) \leq tt^{max}(\overline{s}_1)$

(iii) $s_2 \lhd \overline{s}_2$

(i) Event $\overline{e}$ is an element of $\overline{AE}(\overline{s}_1)$ if it satisfies the five conditions stated in the definition of $\overline{AE}$. All conditions except condition (3) follow directly from the definition of $\overline{e}$ and the fact that $e \in AE(s_1)$. To prove the fact that condition (3) holds we have to impose additional restrictions on f, however it is always possible to transform ("massage") f such that (3) holds (see appendix B).
(ii) Because $\pi_2(e) \lhd \pi_2(\overline{e})$ $et^{min}(\overline{e}) \leq et(e)$:
$et^{min}(\overline{e}) = \max_{i \in dom(\pi_2(\overline{e}))} time^{min}(\pi_2(\overline{e})(i)) \leq \max_{i \in dom(\pi_2(e))} time(\pi_2(e)(i)) = et(e)$
It is also easy to verify that: $tt(s_1) \leq tt^{max}(\overline{s}_1)$ because $s_1 \lhd \overline{s}_1$.
Therefore: $et^{min}(\overline{e}) \leq et(e) = tt(s_1) \leq tt^{max}(\overline{s}_1)$.

(iii) From $s_1 \lhd \overline{s}_1$ and the definition of $\pi_2(\overline{e})$ we deduce that:
$s_1 \upharpoonright (dom(s_1) \setminus dom(\pi_2(e))) \;\lhd\; \overline{s}_1 \upharpoonright (dom(\overline{s}_1) \setminus dom(\pi_2(\overline{e})))$

Because $et^{min}(\overline{e}) \leq tt(s_1) \leq tt^{max}(\overline{s}_1)$:
$scale(\pi_3(e), tt(s_1)) \triangleleft \overline{scale}(\pi_3(\overline{e}), et^{min}(\overline{e}), tt^{max}(\overline{s}_1))$
This implies that $s_2 \triangleleft \overline{s}_2$.
□

This theorem tells us that if a transition is possible from s_1 to s_2 in the original transition system, there is a corresponding transition in the modified transition system from every state $\overline{s}_1$ that 'covers' s_1.

How are the paths in the modified transition system related to the paths in the original transition system? To investigate this we also define specialisation for paths ($\triangleleft_\pi$).

Definition 13 (Specialisation)
For $\sigma \in \mathbb{N} \not\rightarrow S$ and $\overline{\sigma} \in \mathbb{N} \not\rightarrow \overline{S}$: $\quad \sigma \triangleleft_\pi \overline{\sigma} \equiv (dom(\sigma) = dom(\overline{\sigma}) \ \wedge \ \forall_{i \in dom(\sigma)} \ \sigma_i \triangleleft \overline{\sigma}_i)$

Now it is possible to show that soundness also holds for the processes (Π and $\overline{\Pi}$) generated by the two transition systems.

Lemma 1
For all $s_1 \in S$ and $\overline{s}_1 \in \overline{S}$ such that $s_1 \triangleleft \overline{s}_1$: $\quad \forall_{\sigma \in \Pi(s_1)} \ \exists_{\overline{\sigma} \in \overline{\Pi}(\overline{s}_1)} \ \sigma \triangleleft_\pi \overline{\sigma}$

Proof.
If σ is an infinite path (i.e. $dom(\sigma) = \mathbb{N}$), then we have to prove that there is an $\overline{\sigma}$ such that $dom(\overline{\sigma}) = \mathbb{N}$ and $\forall_{i \in dom(\sigma)} \ \sigma_i \triangleleft \overline{\sigma}_i$. Because $s_1 \triangleleft \overline{s}_1$ we find that $\sigma_0 \triangleleft \overline{\sigma}_0$. For all $i \geq 0$ take $\overline{\sigma}_{i+1} \in \overline{R}(\overline{\sigma}_i)$ such that $\sigma_{i+1} \triangleleft \overline{\sigma}_{i+1}$. This is possible because of soundness. If σ is a finite path of length n, then we have to prove that $\overline{\sigma}_{n-1}$ is a terminal state. We know that $R(\sigma_{n-1}) = \emptyset$ and that $\sigma_{n-1} \triangleleft \overline{\sigma}_{n-1}$. This implies that $\overline{R}(\overline{\sigma}_{n-1}) = \emptyset$ because if $AE(\sigma_{n-1}) = \emptyset$ then $\overline{AE}(\overline{\sigma}_{n-1}) = \emptyset$ ($\overline{R}(\overline{\sigma}_{n-1}) = \emptyset$ implies that there is no transition with enough tokens on its input places).
□

Despite of the non-completeness, the soundness property allows us to answer some of the questions stated in section 2.2. We can *prove* that a system has a desired set of properties by proving it for the modified transition system. For example:

Lemma 2
If the modified transition system indicates that an ITPN is K-bounded (or safe) for an initial state with respect to the modified model then the net is K-bounded (or safe) for that initial state with respect to the original model.

Proof.
Use theorem 3.
□

We also use the modified transition system to calculate bounds for the arrival times of tokens in a place. In other words: the modified transition system gives us an indication about the arrival times. Although these bounds are sound they do not have to be as rigid as possible because of possible dependencies between tokens (non-completeness). First we define the earliest and latest arrival time for the modified transition system, to do this we need to define place projection ($\Uparrow^{min}$ and $\Uparrow^{max}$) for $\overline{S}$. For all $\overline{s} \in \overline{S}$, $p \in P$:
$\overline{s} \Uparrow^{min} p = \lambda_{x \in TS}\ \#\{i \in dom(\overline{s}) \mid place(\overline{s}(i)) = p\ \wedge\ time^{min}(\overline{s}(i)) = x\}$
$\overline{s} \Uparrow^{max} p = \lambda_{x \in TS}\ \#\{i \in dom(\overline{s}) \mid place(\overline{s}(i)) = p\ \wedge\ time^{max}(\overline{s}(i)) = x\}$
(i.e. $\Uparrow^{min}$ ($\Uparrow^{max}$) gives the bag of lower (upper) bounds of the intervals of the tokens in p).
If $\overline{A} \subseteq \overline{S}$ and $p \in P$ then:

$$\overline{\mathcal{EAT}}_n(A,p) = \min_{\overline{\sigma} \in \overline{\Pi}(\overline{A})} \min_{i \in dom(\overline{\sigma})} \min_n(\overline{\sigma}_i \Uparrow^{min} p)$$

$$\overline{\mathcal{LAT}}_n(A,p) = \max_{\overline{\sigma} \in \overline{\Pi}(\overline{A})} \min_{i \in dom(\overline{\sigma})} \min_n(\overline{\sigma}_i \Uparrow^{max} p)$$

Lemma 3
If $A \subseteq S, \overline{A} \subseteq \overline{S}, p \in P$ and $A \subseteq \Gamma(\overline{A})$ then:

- $\overline{\mathcal{EAT}}_n(\overline{A},p) \leq \mathcal{EAT}_n(A,p)$
- $\overline{\mathcal{LAT}}_n(\overline{A},p) \geq \mathcal{LAT}_n(A,p)$

Proof.
Use lemma 1.
□

We have demonstrated that we can use the modified model to answer all kinds of questions about the original model. The reason we use a modified model is the fact that it is possible to calculate the set of reachable states (or at least a subset) for this model. The software tool IAT uses the modified transition system to generate (a part of) the reachability tree. Because reachability tree of the modified model is much smaller and more coarsely grained than the original we call it the *reduced reachability tree*. Every state in the reduced reachability tree corresponds to a (infinite) number of states in the reachability tree of the original model. One can think of these states as equivalence classes.

A possible drawback of the analysis method MTSRT is the fact that answers are not always as strict as possible because of dependencies between tokens. The computational efficiency depends upon the size and the structure of the net ("conflicts are considered harmful").
A similar approach is described in [Berthomieu et al. 83] using Merlin's Timed Petri Nets ([Merlin 74]). We believe our method is more efficient for large nets because their method involves solving linear equations to calculate state classes. The efficient implementation of IAT allows the user to generate reachability trees with thousands of states in less than a minute.

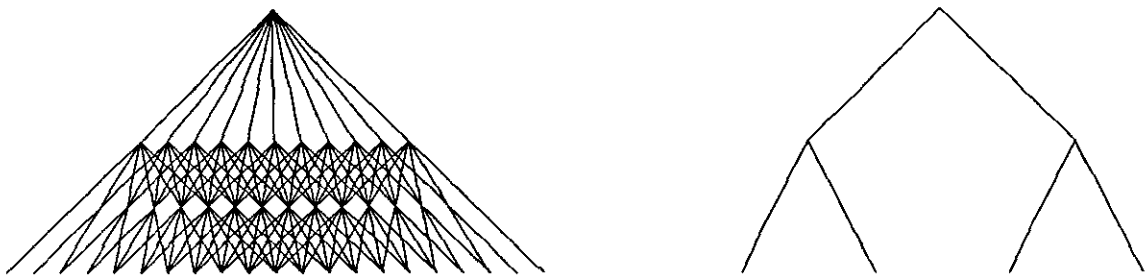


Figure 7: The reachability tree of the original model versus the reachability tree of the modified model

5 Other analysis methods

In this section we discuss the two remaining analysis methods; method 3 and method 4. These analysis methods can be applied to a subclass of ITP-nets, the so-called *Interval Timed Event Graphs* (ITEG). The underlying net structure of such net is a Marked Graph [7].

Definition 14 (Interval Timed Event Graph)
An ITPN = (P, T, I, O, TS) is an Interval Timed Event Graph if for all $p \in P$:

$$\sum_{t \in T} I_t(p) \leq 1$$
$$\sum_{t \in T} \sum_{v \in INT} O_t(\langle p, v \rangle) \leq 1$$

i.e. the number of input arcs and the number of output arcs of a place is 0 or 1 and the multiplicity of each arc is 1.

An ITEG can be seen as a generalisation of *Timed Event Graph* in the sense that we use intervals to specify delays instead of a deterministic value. The dynamic behaviour of Timed Event Graphs has been studied by a lot of people (see [Ramamoorthy et al. 80] and [Carlier et al. 87]). A lot of applications have been modelled using Timed Event Graphs (especially in the field of flexible manufacturing, see [Silva 89]). Even though (Interval) Timed Event Graphs allow the modelling of parallelism and synchronisation of events, shared resources (i.e. competition relationships) cannot be modelled. Interval Timed Event Graphs represent a generalisation of PERT/CPM graphs, allowing for the study of repetitive schedules (see appendix C).
There exist a number of analysis techniques for Timed Event Graphs using the absence of *confusion* in these nets. In this paper we present two alternative analysis techniques. The first one is based on the analysis technique discussed in section 4. This analysis technique calculates terminal states in a net very efficiently. The second one allows for evaluating the steady-state performance of a system. It is a generalisation of the performance analysis technique described in [Ramamoorthy et al. 80].

[7] A Petri Net is a Marked Graph if and only if for each place $p \in P$; $\bullet p \leq 1$ and $p \bullet \leq 1$.

5.1 Method CFNRT

The third analysis technique we present is called *Confusion Free Net Reduction Technique* (CFNRT). Method CFNRT uses the special network structure of an ITEG to calculate the set of reachable states very efficiently. Because the origin and destination of a token is known from the topology of the net we call these nets *confusion free.* A nice property of confusion free nets is the fact that the order in which the transitions fire does not matter when you are calculating the set of terminal states. Method CFNRT uses the modified transition system described in the previous section in a slightly altered way. This way the size of the (reduced) reachability tree is reduced considerably.
IAT detects the absence of confusion and uses this to calculate the results more efficiently.

First we formalise the concept confusion free, then we prove that a confusion free net has a desirable property. Finally we show that under some conditions an ITEG is confusion free. We start with some auxiliary definitions.

Definition 15 (Well-formed)
A state $\overline{s} \in \overline{S}$ is *well-formed* if and only if:

$$\forall_{i,j \in dom(\overline{s})}\, place(\overline{s}(i)) = place(\overline{s}(j)) \Rightarrow (time(\overline{s}(i)) \leq_i time(\overline{s}(j)) \vee time(\overline{s}(j)) \leq_i time(\overline{s}(i)))$$

A state is well-formed if the time intervals of any pair of tokens in the same place are comparable. In other words, of any two tokens in the same place having distinct intervals, one interval is smaller than the other.

Definition 16 (Chronological)
An ITPN is *chronological* with respect to a state $\overline{s} \in \overline{S}$ if and only if:

1. $\overline{s}$ is well-formed
2. for all $t \in T$, $\mathcal{BS}(O_t)$ is well-formed
3. $\forall_{\hat{s} \in \overline{RS}(\overline{s})}\, \forall_{\tilde{s} \in \overline{R}(\hat{s})}\, \forall_{i \in dom(\hat{s})}\, \forall_{j \in dom(\tilde{s}) \setminus dom(\hat{s})}\, place(\hat{s}(i)) = place(\tilde{s}(j)) \Rightarrow time(\hat{s}(i)) \leq_i time(\tilde{s}(j))$

The third requirement says that the time intervals of the tokens arriving in each place have to be ascending in the order of arrival. All produced tokens have an interval of at least any interval of the tokens contained by the (corresponding) place until then.

Lemma 4
If an ITPN is chronological with respect to $\overline{s} \in \overline{S}$ then for all $\hat{s} \in \overline{RS}(\overline{s})$:
$\hat{s}$ is well-formed and the net is chronological with respect to $\hat{s}$.

Proof.
If $\hat{s} \in \overline{RS}(\overline{s})$ then there exists an $n \in \mathbb{N}$ such that $\hat{s} \in \overline{R}^n(\overline{s})$.
Let $P(n)$ be the proposition that all $\hat{s} \in \overline{R}^n(\overline{s})$ are well-formed and the net is chronological w.r.t. $\hat{s}$.
$P(0)$ is trivial because $\hat{s} = \overline{R}^0(\overline{s}) = \overline{s}$ is well-formed and the net is chronological w.r.t. $\overline{s}$.
Suppose $n > 0$ and $P(n-1)$ (induction hypothesis).
For all $\tilde{s} \in \overline{R}^n(\overline{s})$ there exists a state $\hat{s} \in \overline{R}^{n-1}(\overline{s})$ such that $\tilde{s} \in \overline{R}(\hat{s})$. Because the net is chronological and well-formed (induction) with respect to $\hat{s}$, the corresponding event e which transforms $\hat{s}$ into $\tilde{s}$ adds tokens to each output place such that the state remains well-formed. This is guaranteed by the fact that the produced tokens are well-formed and any produced token with interval v and any token (with interval w) contained by the corresponding place until then, satisfy $w \leq_i v$.
The net is also chronological w.r.t. $\tilde{s}$, because $\tilde{s}$ is well-formed, for all $t \in T$, $\mathcal{BS}(O_t)$ is well-formed and the third requirement also holds (because $\overline{RS}(\hat{\tilde{s}}) \subseteq \overline{RS}(\hat{s})$).
□

Our definition of *confusion free* deviates of traditional definitions.

Definition 17 (Confusion free)
An ITPN is *confusion free* with respect to $\overline{s} \in \overline{S}$ if and only if the net is conflict free and chronological with respect to $\overline{s}$.

A confusion free net has the nice property that if it terminates, it always terminates in the 'same state'. This is expressed in the following theorem that holds after a minor alteration of the modified transition system of section 4; replace $\overline{scale}(\pi_3(e), et^{min}(e), tt^{max}(s_1))$ by $\overline{scale}(\pi_3(e), et^{min}(e), et^{max}(e))$. This way the time intervals of the produced tokens do not depend upon the other allowed events. This transition system satisfies all properties mentioned in this section and the previous section (soundness, ..) because $et^{max}(e) \geq tt^{max}(s_1)$.

Theorem 4
If an ITPN is confusion free and dead with respect to an initial state $\overline{s} \in \overline{S}$ then:

$$\#\{\mathcal{SB}(\hat{s}) \mid \hat{s} \in \overline{RS}(\overline{s}) \ \wedge \ \overline{R}(\hat{s}) = \emptyset\} = 1$$

Proof.
Because the net is confusion free, all $\hat{s} \in \overline{RS}(s)$ are well-formed (see lemma 4). This implies that $e_1, e_2 \in \overline{AE}(\hat{s})$ and $\pi_1(e_1) = \pi_1(e_2) \Rightarrow e_1 \doteq e_2$ [8], because there are no conflicts between tokens on the input places (observe condition (3) in the definition of $\overline{AE}$). If an event e occurs the intervals of the produced tokens only depend upon e and not upon any other event (see remark about the minor alteration of the modified transition system). Once an event e is firable: $e \in \overline{AE}(\hat{s})$ and $et^{min}(e) \leq tt^{max}(\hat{s})$, it remains firable until it occurs.

[8] If $e_1, e_2 \in \overline{E}$ then $e_1 \doteq e_2$ iff $\pi_1(e_1) = \pi_1(e_2) \wedge \mathcal{SB}(\pi_2(e_1)) = \mathcal{SB}(\pi_2(e_2)) \wedge \mathcal{SB}(\pi_3(e_1)) = \mathcal{SB}(\pi_3(e_2))$.

In other words an event will not be disabled by any other event. If another event, say h, occurs in $\hat{s}$ then e is still firable in: $\hat{\hat{s}} = \hat{s}\restriction(dom(\hat{s})\backslash\pi_2(h)) \cup \overline{scale}(\pi_3(h), et^{min}(h), et^{max}(h))$ because:

1. $dom(\pi_2(h)) \cap dom(\pi_2(e)) = \emptyset$, because of the absence of conflicts (condition (1) in the definition of $\overline{AE}$ holds)
2. $\forall_{i\in dom(\pi_2(e))}\forall_{j\in dom(\hat{s})\backslash dom(\pi_2(e))}\ place(\hat{\hat{s}}(i)) = place(\hat{\hat{s}}(j)) \Rightarrow \neg(time(\hat{\hat{s}}(j)) \leq_i time(\hat{\hat{s}}(i)))$ because the net is chronological w.r.t. $\hat{s}$, (condition (3) in the definition of $\overline{AE}$ holds)
3. The other conditions (2,4 and 5) in the definition of $\overline{AE}$ still hold for e (sometimes $\pi_3(e)$ has to be relabelled because some labels are already used)
4. $et^{min}(e) \leq tt^{max}(\hat{s}) \leq tt^{max}(\hat{\hat{s}})$, see theorem 2

Because the net is dead, the set of allowed event becomes empty after a while. This and the fact that an event will not be disabled implies that the ordering of event is not important, i.e. $\#\{\mathcal{SB}(\hat{s}) \mid \hat{s} \in \overline{RS}(\overline{s})\ \wedge\ \overline{R}(\hat{s}) = \emptyset\} = 1$.
□

This theorem tells us that it does not matter which events are chosen during the execution of the net; all paths (firing sequences) lead to the same terminal state in the modified transition system. Therefore this terminal state can be calculated very efficiently; resolve all choices by selecting an arbitrary event.

One way to calculate the terminal state is a simulation using a *coloured timed* Petri Net. Every place (transition) in the coloured net corresponds to a place (transition) in the ITPN net. The value of a token in the coloured net is the time interval of the token in the ITPN net. The delay of the token is some value in the delay interval. The value of a produced token is calculated using the values of consumed tokens and the specified delay interval. We will not go into this subject because IAT calculates the terminal state more efficiently.

For an arbitrary net it is very difficult to verify whether the net is confusion free. However there is an important class of nets for which we can prove that they are confusion free. This is expressed by theorem 5. To prove theorem 5 we need the following lemma which tells us that the maximal (interval) sequence of two ascending (interval) sequences is ascending.

Lemma 5
If $n \in \mathbb{N}$, $v_1, v_2, .., v_n \in INT$ and $w_1, w_2, .., w_n \in INT$ such that $\forall_{i\in\{1..n-1\}}\ (v_i \leq_i v_{i+1})\ \wedge\ (w_i \leq_i w_{i+1})$ then: [9]

$$\forall_{i\in\{1..n-1\}}\ (v_i \max w_i) \leq_i (v_{i+1} \max w_{i+1})$$

[9]If $v, w \in INT$ then $v \max w = \langle\pi_1(v) \max \pi_1(w), \pi_2(v) \max \pi_2(w)\rangle$.

Proof.
For $i \in \{1..n-1\}$, $v_i \leq_i v_{i+1} \;\wedge\; w_i \leq_i w_{i+1}$ implies that
$\pi_1(v_i) \leq \pi_1(v_{i+1})$, $\pi_1(w_i) \leq \pi_1(w_{i+1})$, $\pi_2(v_i) \leq \pi_2(v_{i+1})$ and $\pi_2(w_i) \leq \pi_2(w_{i+1})$.
$\pi_1(v_i \max w_i) = \pi_1(v_i) \max \pi_1(w_i) \leq \pi_1(v_{i+1}) \max \pi_1(w_{i+1}) = \pi_1(v_{i+1} \max w_{i+1})$
$\pi_2(v_i \max w_i) = \pi_2(v_i) \max \pi_2(w_i) \leq \pi_2(v_{i+1}) \max \pi_2(w_{i+1}) = \pi_2(v_{i+1} \max w_{i+1})$
Therefore: $(v_i \max w_i) \leq_i (v_{i+1} \max w_{i+1})$.
□

An ITEG is confusion free if the initial state is well-formed, all start places $P^S = \{p \in P \mid \bullet\, p = \emptyset\}$ contain tokens with an interval of at least any other token in a non-start place and all tokens in non-start places have the same interval. This property of Interval Timed Event Graphs is expressed in the following theorem.

Theorem 5
An Interval Timed Event Graph with an initial state $\overline{s} \in \overline{S}$ such that:

1. $\overline{s}$ is well-formed
2. $\forall_{i,j \in dom(\overline{s})}\ (place(\overline{s}(i)) = place(\overline{s}(j)) \;\wedge\; place(\overline{s}(i)) \in (P \setminus P^S)) \Rightarrow time(\overline{s}(i)) = time(\overline{s}(j))$
3. $\forall_{i,j \in dom(\overline{s})}\ place(\overline{s}(i)) \in (P \setminus P^S) \;\wedge\; place(\overline{s}(j)) \in P^S \Rightarrow time(\overline{s}(i)) \leq_i time(\overline{s}(j))$

is confusion free with respect to s.

Proof.
By definition an ITEG is conflict free. Remains to prove that the net is chronological with respect to $\overline{s}$.
(i) $\overline{s}$ is well-formed.
(ii) $\forall_{t \in T}\ BS(O_t)$ is well-formed because the net is an ITEG
(iii) Remains to prove that:

$$\forall_{\hat{s} \in \overline{RS}(\overline{s})} \forall_{\hat{\hat{s}} \in \overline{R}(\hat{s})} \forall_{i \in dom(\hat{s})} \forall_{j \in dom(\hat{\hat{s}}) \setminus dom(\hat{s})}\ place(\hat{s}(i)) = place(\hat{\hat{s}}(j)) \Rightarrow time(\hat{s}(i)) \leq_i time(\hat{\hat{s}}(j))$$

A first observation tells us that requirement (iii) holds for all tokens in a start place ($P^S = \{p \in P \mid \bullet\, p = \emptyset\}$), because no event will add tokens to one of these places.
If $t \in T$ a transition such that the net is chronological in all input places of t (w.r.t. $\overline{s}$), then all output places are chronological too, because t is the only transition producing tokens for these places, the tokens initially available satisfy (2.) and (3.) and lemma 5 tells us that if the intervals of the tokens on the input places are ascending then the tokens on the output places are also ascending.
Consider a place $p \in P$ with $\bullet p \neq \emptyset$. Suppose the net is *not* chronological in p w.r.t. s. Then there exist(ed) two tokens in p with intervals v and w such that the token with interval v existed before the token with interval w and $\neg(v \leq_i w)$ with overlapping intervals. Suppose both tokens existed in the initial state s, this is not possible because then $v = w$

(see(3.)). Suppose that initially there was only one token in p (with interval v), then there is a contradiction because all tokens produced by a transition have an interval w of at least v; i.e. $v \leq_i w$. Otherwise both tokens are produced by some transition t (every place has only one input transition). But this means that one of the input places of t contained a token with interval $\hat{v}$ and a token with interval $\hat{w}$ such that the token with interval $\hat{v}$ existed before the token with interval $\hat{w}$ and $\neg(\hat{v} \leq_i \hat{w})$, this follows from lemma 5. Continue this reasoning until a contradiction is encountered, either because all input places of t have no incoming arcs or because one reaches the initial state s which is well-formed.
□

This theorem tells us that under some conditions an ITEG is confusion free. If the net is dead then there is just one terminal state in the modified transition system. This terminal state can be calculated very efficiently. Because of the soundness properties stated in section 4 we can answer a number of questions. For example we can calculate the earliest arrival time ($\mathcal{EAT}$) and the latest first arrival time ($\mathcal{LAT}$) of a place p without outgoing arcs. Note that because of the absence of confusion the produced bounds are as "rigid" as possible.

5.2 Method SSPAT

The fourth analysis technique we present is called *Steady State Performance Analysis Technique* (SSPAT). The analysis techniques described so far are based on the principle of calculating the reachable states from one or more initial states. This approach allows for the analysis of *open systems* and *closed systems*. An open system is system whose environment is not modelled explicitly, the behaviour of an environment is modelled via the initial state(s) (marking). *Closed systems* are systems where the environment is modelled explicitly. In a closed system the initial state represents the available resources, not the behaviour of the environment. The analysis methods presented so far are also capable of analysing both kinds of systems, but in general we are also interested in the steady-state functioning of the net and not only in the reachable states. Therefore we present an analysis method to calculate the steady-state performance of a closed system. This is a generalisation of the technique presented in [Ramamoorthy et al. 80], which is based on Timed Event Graphs with time in transitions. It is a generalisation in the sense that it produces upper and lower bounds for the performance and in the sense that time is in tokens allowing for the modelling of the two kinds of delay discussed in section 2.1.

The measure of performance we consider is the average time between two successive firings of a transition. We start by giving some properties of (Interval) Timed Event Graphs.

Definition 18 (Directed path)
A *directed path* ρ is a sequence of places: $\rho \in \mathbb{N} \not\rightarrow P$ such that: $0 \in dom(\rho)$ and for all $i \in dom(\rho) \setminus \{0\}$: $(i-1) \in dom(\rho)$ and $\rho_{i-1} \in \bullet(\bullet\rho_i)$.

A directed path starts in a place $begin(\rho) = \rho_0$ and ends in a place $end(\rho) = \rho_j$ where $j = \max\ dom(\rho)$. For any successive pair of places ρ_{i-1} and ρ_i in a directed path there exists a transition t such that ρ_{i-1} is an input place of t and ρ_i is an output place of t $(\rho_{i-1} \in \bullet(\bullet\rho_i)$.

Definition 19 (N_ρ)
For a directed path ρ we define the number of tokens in the places contained by ρ for a state $s \in S$ as follows: $N_\rho(s) = \#\{i \in dom(s) \mid place(s(i)) \in rng(\rho)\}$

Definition 20 (Directed circuit)
A *directed circuit* ρ is a directed path with $begin(\rho) = end(\rho)$.

A directed circuit is called *elementary* if all elements (except the first one) differ:
$\forall_{i,j \in dom(\rho)\setminus\{0\}}\ i \neq j \Rightarrow \rho_i \neq \rho_j$.

Theorem 6
For an ITEG the number of tokens in a directed circuit ρ remains the same under any firing sequence, more formally for all $s \in S$ and $\hat{s} \in RS(s)$: $N_\rho(\hat{s}) = N_\rho(s)$.

Proof.
Tokens in a directed circuit can only be produced or consumed by a transition contained by the circuit. Every place in a directed circuit has exactly one input transition and one output transition. If such a transition fires, the number of tokens consumed from the circuit equals the number of tokens produced back into the circuit. Therefore, the number of tokens in a directed circuit remains the same under any firing sequence.
□

Definition 21 ($G_\rho^{min}, G_\rho^{max}$)
For an ITEG having a directed path ρ we define G_ρ^{min} and G_ρ^{max}: [10]

$$G_\rho^{min} = \sum_{i \in dom(\rho)\setminus\{0\}} d^{min}(\rho_{i-1}, \rho_i)$$
$$G_\rho^{max} = \sum_{i \in dom(\rho)\setminus\{0\}} d^{max}(\rho_{i-1}, \rho_i)$$

the upper and lower bound for the sum of the delays in a directed path ρ.

If every delay interval of an ITEG is a point interval (i.e. an interval of length 0) then for all directed paths ρ; $G_\rho^{min} = G_\rho^{max}$. In this case we speak about G_ρ, the *length* of an directed path.

Definition 22 (Strongly connected)
A Petri Net (or ITPN) is *strongly connected* if and only if every pair of places is contained in a directed circuit.

[10] If $t \in T$ such that $p_1 \in \bullet t$ and $\langle p_2, \langle x, y\rangle\rangle \in O_t$ then $d^{min}(p_1, p_2) = x$ and $d^{max}(p_1, p_2) = y$.

Definition 23 (Consistent)
An ITEG is called *consistent* with respect to an initial state $s \in S$ if and only if the net is strongly connected, every circuit contains at least one token and the net is progressive.

Consistent ITEGs form the subclass of nets we are going to analyse. These nets have a number of convenient properties.

Definition 24 (τ)
For an ITPN with initial state $s \in S$ and $\sigma \in \Pi(s)$ we define $\tau(\sigma) \in dom(\sigma) \setminus \{0\} \rightarrow T$ such that for all $i \in dom(\sigma) \setminus \{0\}$:

$$\begin{aligned}\tau(\sigma)(i) = \; \{ \; \pi_1(e) \mid \; & e \in AE(\sigma_{i-1}) \;\wedge\; et(e) = tt(\sigma_{i-1}) \;\wedge \\ & \sigma_i = \sigma_{i-1} \restriction (dom(\sigma_{i-1}) \setminus \pi_2(e)) \;\cup\; scale(\pi_3(e), tt(\sigma_{i-1}))\}\end{aligned}$$

Note that $\tau(\sigma)(i)$ is a singleton if the net is an ITEG. In this case $\tau(\sigma)$ represents the sequence of transitions that fired during the execution of σ.

Definition 25 (S)
For an ITPN with initial state $s \in S$, $\sigma \in \Pi(s)$, $t \in T$ and $n \in \mathbb{N} \setminus \{0\}$:

$$S(\sigma, t, n) = \min_{\substack{i \in dom(\sigma) \\ \#\{0 < j \leq i \;\mid\; t \in \tau(\sigma)(j)\} = n}} tt(\sigma_i)$$

$S(\sigma, t, n)$ is the time at which transition t initiates its n^{th} execution under the firing sequence (path) σ.

Because we are interested in the upper and lower bound of the performance we define σ^{min} and σ^{max}.

Definition 26 ($\sigma^{min}, \sigma^{max}$)
For an ITEG with initial state $s \in S$ we define $\sigma^{min}(s)$ ($\sigma^{max}(s)$) as a path where all delays are equal to the lower (upper) bound of the corresponding delay interval.

It is easy to see that $\sigma^{min}(s)$ and $\sigma^{max}(s)$ represent two extreme behaviours of a net starting in state s. This is expressed in the following theorem.

Theorem 7
For an ITEG with initial state $s \in S$ and a transition $t \in T$:

$$\forall_{n \in \mathbb{N}} \; \forall_{\sigma \in \Pi(s)} \; S(\sigma^{min}(s), t, n) \leq S(\sigma, t, n) \leq S(\sigma^{max}(s), t, n)$$

Proof.
Informal. Because of the absence of conflicts a transition is never disabled. Using minimal delays results in a firing sequence where transitions fire as early as possible because a non-minimal delay can only delay the firing of a transition. Using maximal delays results in firing sequences where transitions fire as late as possible, because a non-maximal delay can only result in an earlier firing. Furthermore, using minimal (maximal) delays results in a 'valid' firing sequence.
□

Definition 27 (TEG)
An ITEG is a *Timed Event Graph* (TEG) if all delay intervals are point intervals, i.e. for all $t \in T$ and $\langle p, \langle x, y \rangle \rangle \in O_t$; $x = y$.

Definition 28 (Cycle time)
For a consistent Timed Event Graph with respect to $s \in S$, $\sigma \in \Pi(s)$ and $t \in T$ we define:

$$C_t = \lim_{n \to \infty} \frac{S(\sigma, t, n)}{n}$$

the *cycle time* of a transition t.

This limit exists because a Timed Event Graph has a deterministic behaviour and the fact that the net is consistent implies that its behaviour is even periodical. Note that C_t does not depend upon σ because of the absence of conflicts and the fact that all delays have a fixed value. We interpret the cycle time of a transition as a performance measure. A shorter cycle time corresponds to a better performance (shorter processing times, more (production) throughput).

Theorem 8
For a consistent TEG with respect to $s \in S$ and a path $\sigma \in \Pi(s)$, all transitions $t \in T$ have the same cycle time C_t

Proof.
Let $t, \hat{t}$ be two transitions, then there exists an elementary directed circuit containing both transitions because the net is strongly connected. If we partition this circuit into two directed paths; a path ρ from t to $\hat{t}$ and a path $\hat{\rho}$ from $\hat{t}$ to t. If t initiates its n^{th} execution in state σ_i then $\hat{t}$ has fired at least $n - N_\rho(\sigma_i)$ times but no more than $n + N_{\hat{\rho}}(\sigma_i)$ times. This implies that $S(\sigma, \hat{t}, n - N_\rho(\sigma_i)) \leq S(\sigma, t, n) \leq S(\sigma, \hat{t}, n + N_{\hat{\rho}}(\sigma_i))$. Because $N_\rho(\sigma_i)$ and $N_{\hat{\rho}}(\sigma_i)$ are finite if $n \to \infty$:

$$C_{\hat{t}} = \lim_{n \to \infty} \frac{S(\sigma, \hat{t}, n - N_\rho(\sigma_i))}{n} \leq \lim_{n \to \infty} \frac{S(\sigma, t, n)}{n} \leq \lim_{n \to \infty} \frac{S(\sigma, \hat{t}, n + N_{\hat{\rho}}(\sigma_i))}{n} = C_{\hat{t}}$$

Therefore, $C_{\hat{t}} = C_t$.
□

This implies that we can speak about *the* cycle time of the net.

Theorem 9
For a consistent TEG with respect to $s \in S$ the cycle time is: [11]

$$C = \max_\rho \frac{G_\rho}{N_\rho(s)}$$

[11] Maximise for all (elementary) circuits ρ.

Proof.
We are able to speak about the cycle time of a net (C) because the delays are fixed and the net is consistent with respect to s, therefore $\forall_{t \in T} \forall_{\sigma \in \Pi(s)} C_t(\sigma) = C$.
First we show that $C \geq \max_\rho \frac{G_\rho}{N_\rho(s)}$ by showing that for every circuit ρ, $CN_\rho(s) \geq G_\rho$ holds. The cycle time C is the time between two successive firings of the same transition. $N_\rho(s)$ is the number of tokens in the places of ρ. So, for all transitions contained by the circuit ρ, the average time it takes to process all tokens once is $CN_\rho(s)$ units. But on the other hand, if such a transition consumes a specific token then it takes at least G_ρ time units until this token is consumed again by the same transition. Therefore, $CN_\rho(s) \geq G_\rho$.
Remains to prove that there exists a circuit ρ such that $CN_\rho(s) = G_\rho$. Consider a critical circuit ρ (i.e. a circuit where $\frac{G_\rho}{N_\rho(s)} = C$) in isolation; its cycle time is C. Consider another circuit $\hat{\rho}$ containing one or more transitions of the critical circuit ρ in isolation. This circuit $\hat{\rho}$ also has a cycle time C because it is blocked by the critical circuit but it cannot block the transitions in the critical circuit (in steady state) because $CN_{\hat{\rho}}(s) \leq G_{\hat{\rho}}$. continue this process until all circuits have been considered.
□

This theorem implies that we can calculate C by evaluating every circuit. More formal proofs of this theorem have been given in [Ramamoorthy et al. 80] and [Chretienne 83]. This result is included here to show that it also holds for TEG's with time in tokens.

A drawback of this approach is that all circuits have to be considered. The number of circuits grows very fast with the size of the net. More efficient procedures to verify the performance of a Timed Event Graph have been suggested by several authors ([Ramamoorthy et al. 80],[Hillion et al. 89]). It is very easy to adapt these procedures for our TEG-nets.

Definition 29 **(C^{min},C^{max})**
For a consistent ITEG with respect to an initial state $s \in S$ and a transition $t \in T$ we define:

$$C_t^{min} = \lim_{n \to \infty} \frac{S(\sigma^{min}(s), t, n)}{n}$$
$$C_t^{max} = \lim_{n \to \infty} \frac{S(\sigma^{max}(s), t, n)}{n}$$

the *minimal cycle time* and the *maximal cycle time* of a transition t.

In a consistent TEG the cycle time of all transitions is the same, therefore we can speak about C^{min} (C^{max}); the minimal (maximal) cycle time of the net.

Theorem 10
For a consistent ITEG with respect to $s \in S$ the minimum cycle time (maximal performance) is given by:

$$C^{min} = \max_\rho \frac{G_\rho^{min}}{N_\rho(s)}$$

and the maximum cycle time (minimal performance) is given by:

$$C^{max} = \max_\rho \frac{G_\rho^{max}}{N_\rho(s)}$$

where

Proof.
Follows directly from the definition of C^{min} and C^{max} and theorem 9.
□

Theorem 7 and theorem 10 tell us that we can calculate the upper and lower bound of the steady state performance of an ITEG by enumerating all (elementary) circuits. For an ITEG with initial state $s \in S$, $t \in T$ $\sigma \in \Pi(s)$ and any $n \in \mathbb{N}$:

$$\frac{S(\sigma^{min}(s), t, n)}{n} \leq \frac{S(\sigma, t, n)}{n} \leq \frac{S(\sigma^{max}(s), t, n)}{n}$$

and

$$C^{min} = \lim_{n\to\infty} \frac{S(\sigma^{min}(s), t, n)}{n} \qquad C^{max} = \lim_{n\to\infty} \frac{S(\sigma^{max}(s), t, n)}{n}$$

I.e. the "average" cycle time of a transition is between C^{min} and C^{max}.

6 Some examples

6.1 The reader/writers problem

The first example we consider is the application of the ITPN model to a variant of the reader/writers problem ([Peterson 81]). Suppose we want to model a (computer) system with a shared resource, lets say a disk. The disk can be used to read from or to write on. There are two kinds of processes; reader processes and writer processes. Reader processes are allowed to read simultaneously (maximal number of readers is n). Because a writer process modifies the data on the disk it has to mutually exclude all other reader and writer processes. Both types of processes are generated by jobs arriving at the computer system. Each job comprises two read processes and one write process.

```
place jobsin;
place jobsout;
place me init 5;
place p1;
place p2;
place p3;
place p4;
place p5;
place p6;
trans start in jobsin out p1[1.0,2.0],p1[1.0,2.0],p2[1.0,2.0];
trans sr in p1,me out p3[2.5,3.0];
trans sw in p2,me,me,me,me,me out p4[4.0,5.0];
trans er in p3 out me,p5[0.0,1.0];
trans ew in p4 out me,me,me,me,me,p6[0.0,1.0];
trans complete in p5,p5,p6 out jobsout[1.0,2.0];
```

Figure 8 shows the corresponding ITPN net. A textual specification is shown in the box (n=5). Jobs arrive via place *jobsin* and leave the system via place *jobsout*. Initially there are n tokens in place *me*. Because the input arc of transition **ws** has a multiplicity of n a writer process mutually excludes all other processes.

We have analysed this system using IAT. Figure 9 shows a screendump of IAT in action. Analysis shows that the system can handle at least 7.5 jobs per minute. Figure 10 shows some results for the first 20 arrivals. The "static report" (reporting the results of method ATCFN) is always available in a few seconds. The response time of "dynamic report" (using method MTSRT) depends on the net and the initial state, in this case only a few seconds (IAT generates about 2000 states per minute).

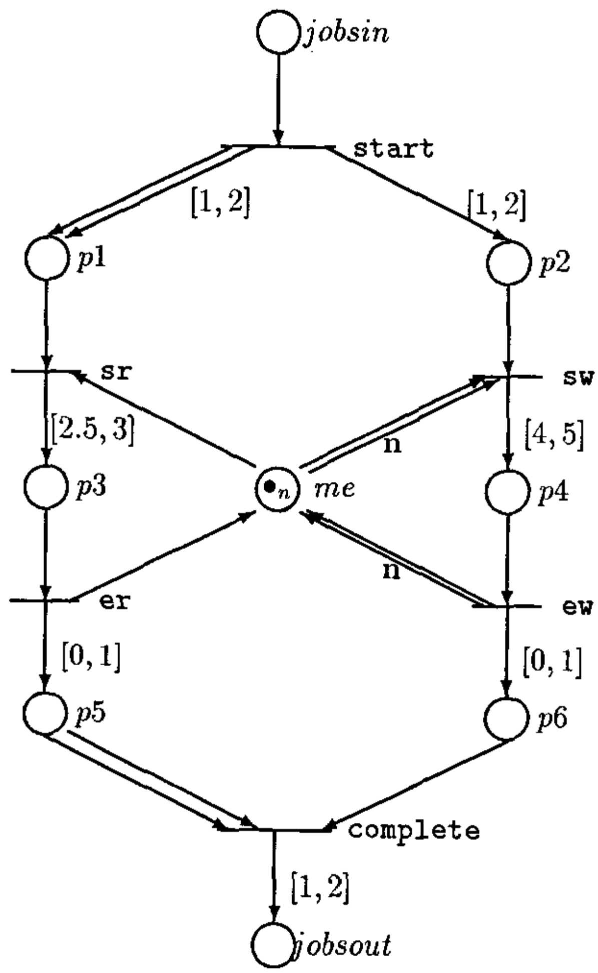


Figure 8: The Readers and Writers system

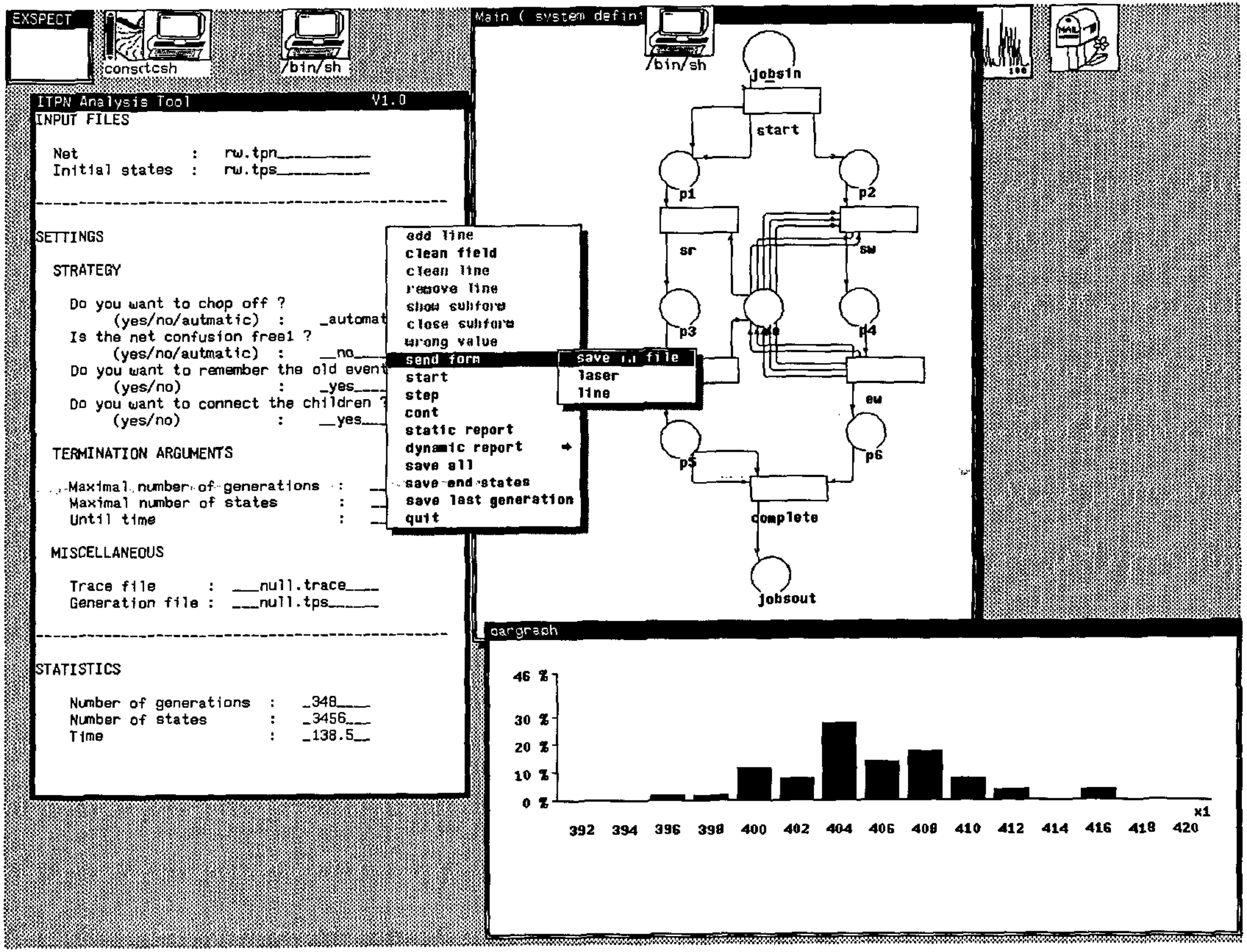


Figure 9: A screendump of ExSpect/IAT

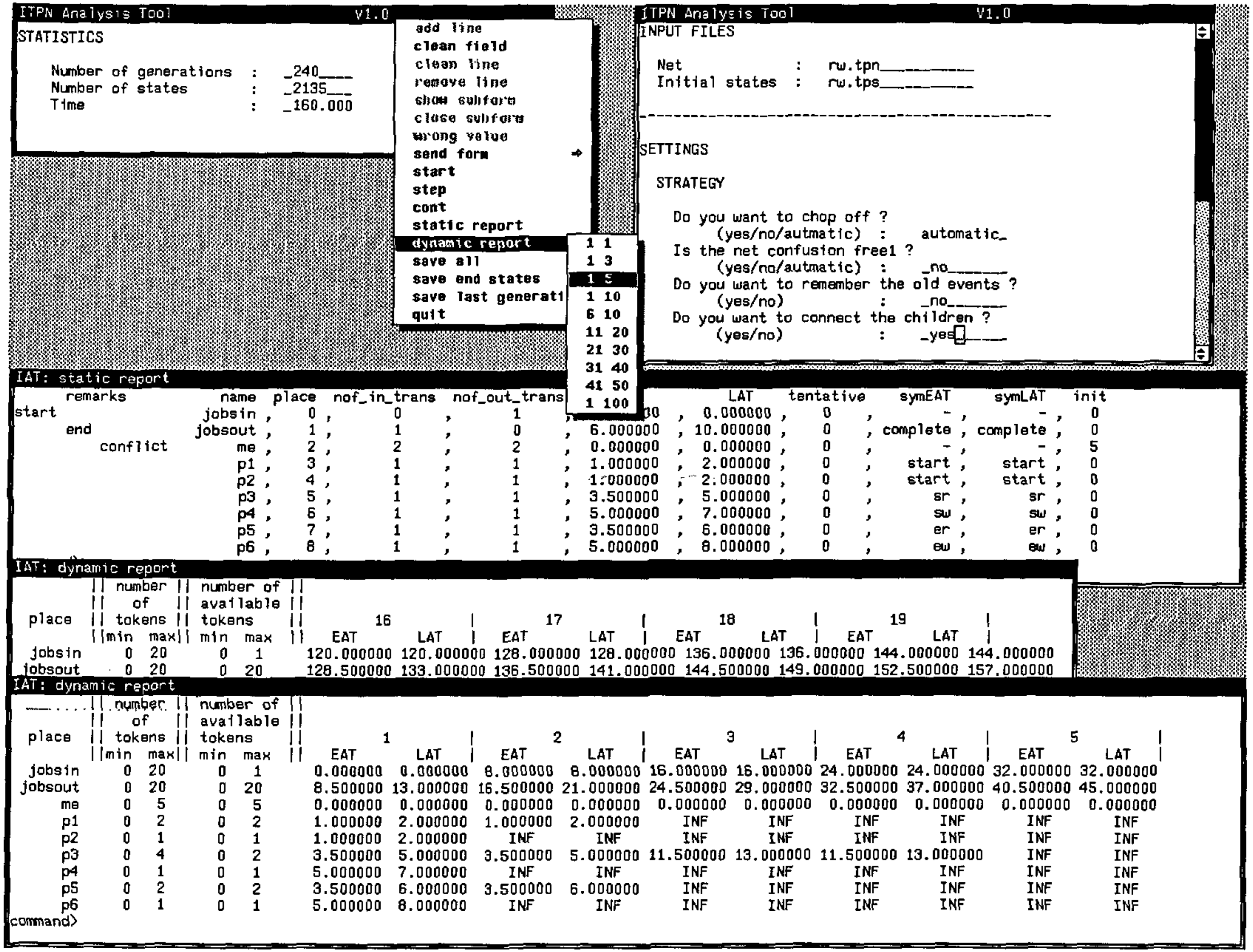


Figure 10: A screendump of IAT analysing the Reader/Writers problem

6.2 A production/assembly system

In order to illustrate the power and accuracy of the techniques presented in the previous sections, we consider a production system with two types of control; *push* control and *pull* control.
Figure 11 shows the bill of material. The production system produces an item $\mathcal{H}$ using

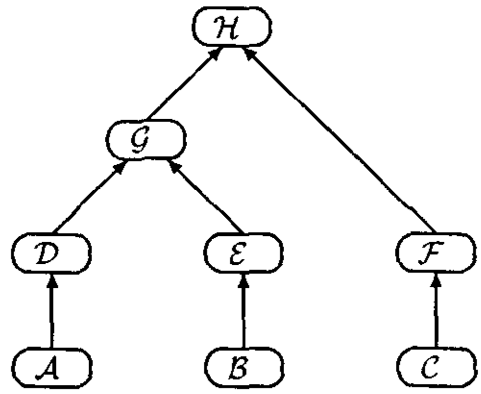


Figure 11: The bill of material

raw materials $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$. There are three machines; **M1** transforms $\mathcal{A}$ into $\mathcal{D}$, **M2** transforms $\mathcal{B}$ into $\mathcal{E}$ and **M3** transforms $\mathcal{C}$ into $\mathcal{F}$. There is one subassembly composing $\mathcal{D}$ and $\mathcal{E}$ into $\mathcal{G}$ and one final assembly composing $\mathcal{G}$ and $\mathcal{F}$ into $\mathcal{H}$.

Let us consider the net in figure 12. Places *p1*,*p2*, .. ,*p11* represent the flow of products. Raw materials $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ enter the system via places *p1*, *p2* and *p3* respectively. Product $\mathcal{D}$ is stored in *p6*, $\mathcal{E}$ in *p7*, $\mathcal{F}$ in *p8*, $\mathcal{G}$ in *p9* and $\mathcal{H}$ in *p10*. Finished products $\mathcal{H}$ leave the system via place *p11*. The demand for product $\mathcal{H}$ arrives via the place *demand*.

Machine **M3** transforms products $\mathcal{C}$ into $\mathcal{F}$ and is modelled by a queuing system represented by the subnetwork containing transitions *t1* and *t2*. Initially there is one token in place *free3* indicating that the machine is ready to operate.
Machines **M1** and **M2** need a setup every time an item is processed. This setup is performed by a person working on both machines. We can think of this person as a *shared resource*. The setup of **M1** is represented by transition *t4*, the setup of **M2** is represented by transition *t3*. The person is represented by a token in place *h*. There is a conflict between *t3* and *t4* because they share the same input place *h*. The remaining parts of **M1** and **M2** are modelled similar to **M3**. Note that we use a *push* control to direct machines **M1**, **M2** and **M3**. Each time raw material is available and the machine is free, an operation is started.
We use a *pull* control to direct the two assembly processes (i.e. assemble to order). In this example a *Kanban*-like control technique is used to reduce the in-process inventory. This technique has been developed in Japan to achieve a Just-in-Time production. Assembling

is allowed if the components needed for the assembly are available and if a certain *card*, called Kanban, has been received. A new Kanban is supplied the moment an assembled product is removed. This way one gets a demand-driven assembly process.
The subassembly and the final assembly are represented by *t9* and *t10*. The delivery of item $\mathcal{H}$ is modelled by transition *t11*. Transition *t11* fires if there is a demand and a finished product. If *t11* fires a new Kanban is supplied to the final assembly process (*t10*). If *t10* fires a new Kanban is supplied to the subassembly process (*t9*). Note that the maximum amount of stored products $\mathcal{G}$ and $\mathcal{H}$ depend on the number of tokens initially available in *kanban1* and *kanban2*.
Figure 12 also shows the delay interval associated with every time consuming operation.

Lets assume that the production system receives a steady flow of raw materials ($\mathcal{A}$,$\mathcal{B}$ and $\mathcal{C}$). Every 20 minutes the system receives an order for one product $\mathcal{H}$. Initially there is one Kanban in *kanban1* and one Kanban in *kanban2*. Now we are interested in the arrival times of tokens in place *p11*. The table below shows some results obtained using method MTSRT. For example the 10^{th} order (generated after (10-1)*20 = 180 minutes) was delivered between 229 ($\mathcal{EAT}_{10}$) and 265 ($\mathcal{LAT}_{10}$) minutes. Therefore the lead time of this order is between 49 and 85 minutes.

ordernumber (n)	$\mathcal{EAT}_n$	$\mathcal{LAT}_n$	minimal lead time	maximal lead time
1	49	67	49	67
2	69	89	49	69
3	89	111	49	71
10	229	265	49	85
50	1029	1145	49	165

The maximal lead time is increasing because the final assembly process sometimes needs 22 minutes per job and this is more than the interarrival time (=20 minutes). The minimal lead time is constant because under ideal circumstances there is an abundance of capacity. Figure 13 shows a screendump of IAT analysing the production system. It takes less than a minute to generate the reachability tree for the first five arrivals.

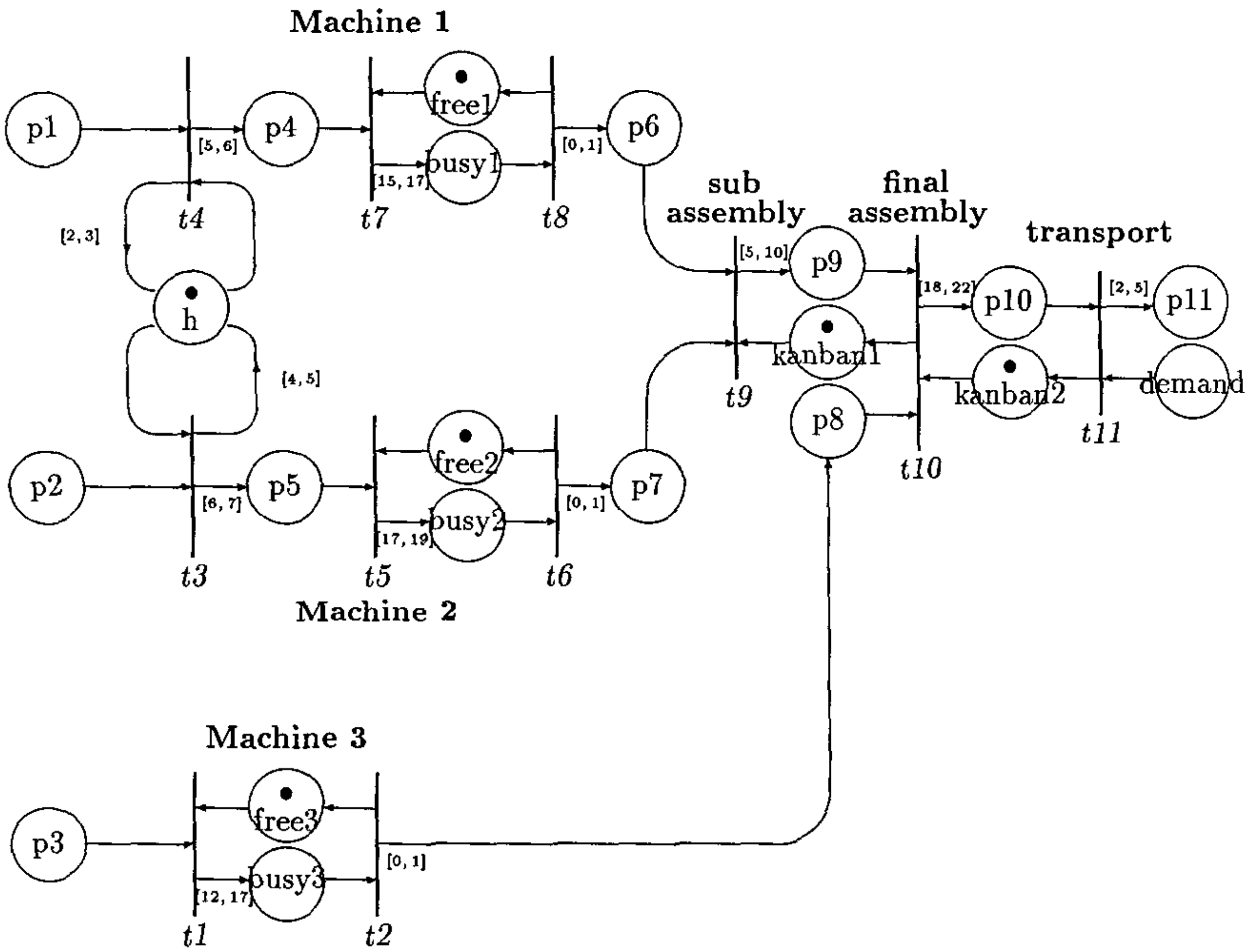


Figure 12: A production system

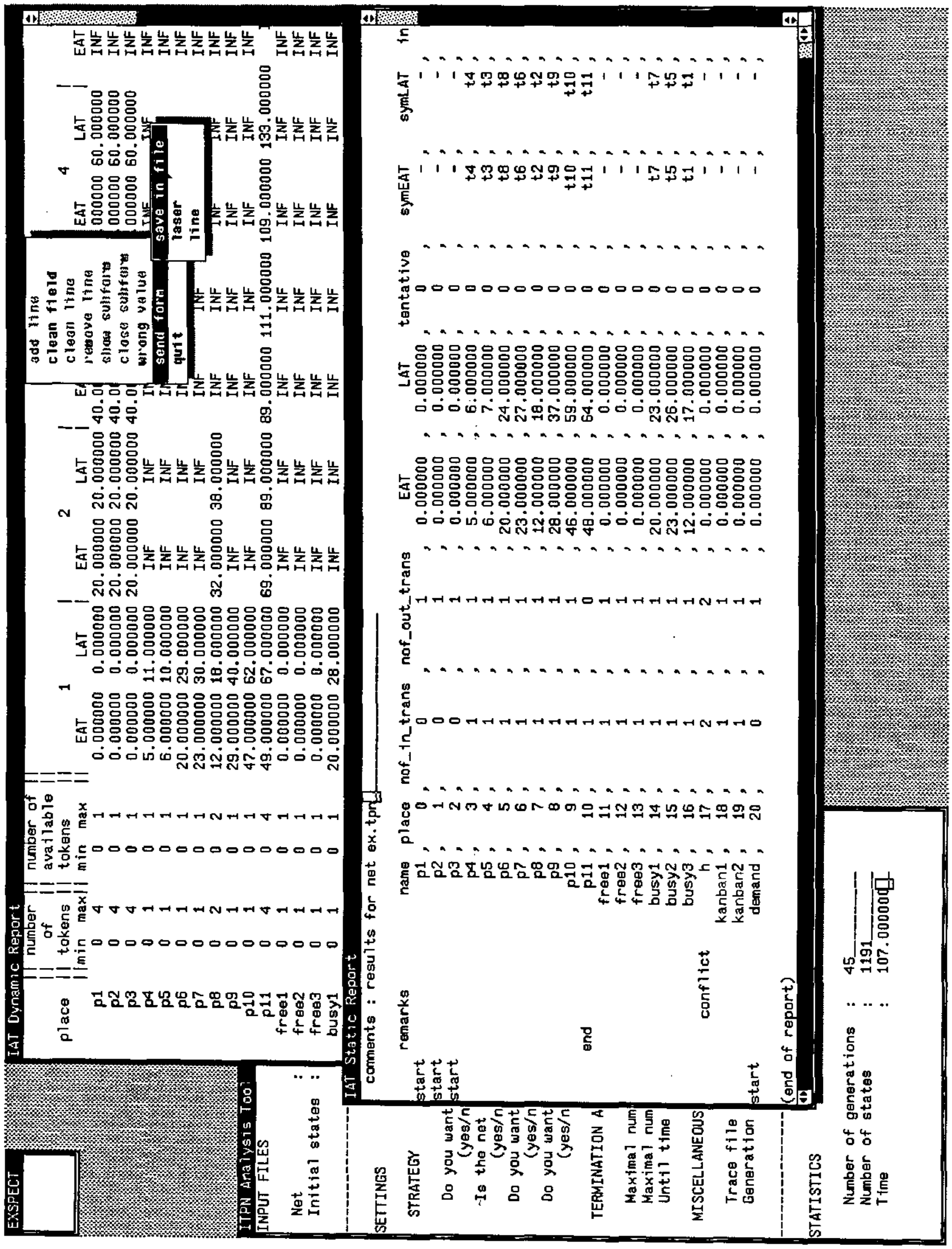


Figure 13: A screendump of IAT analysing the production system

6.3 A job-shop system

The third example we discuss is a job-shop system, with a cyclic production process and a fixed product mix. In [Hillion et al. 89] such a job-shop was modelled using Timed Event Graphs. We will show how to model this job-shop using an ITEG.
Let us consider a job-shop system with three machines, numbered 1,2 and 3. There are 3 different products (or jobs), denoted by A, B and C. The product mix consists of: 25 percent of product A, 50 percent of product B and 25 percent of product C.
The manufacturing process of a product is specified by a *routing* through the system (i.e. a sequence of machines to visit with the corresponding processing times).
Product (or job-type) A starts with an operation on machine 1 (duration between 7 and 10 minutes), then an operation on machine 2 (duration between 15 and 16 minutes) and finally an operation on machine 3 (duration between 5 and 15 minutes).
Product B starts with an operation on machine 2 (duration between 12 and 16 minutes) and ends with an operation on machine 1 (duration between 10 and 13 minutes).
Product C starts with an operation on machine 1 (duration between 11 and 14 minutes) and ends with an operation on machine 3 (duration between 10 and 21 minutes).
Figure 14 shows the routing of the products on the machines. The machines are represented by timed transitions (square box). A timed transition corresponds to the subnetwork shown in figure 2, see section 2.1 for more information. The repetitive functioning is modelled by a number of circuits, called *processing circuits* (we use the same terminology as in [Hillion et al. 89]). Because of the fixed production mix there are two processing circuits for product B and only one processing circuit for A and one processing circuit for C.
There is fixed sequencing of the jobs (products) on the machines. In this example, machine 1 processes product A first, then product C then product B and finally product B again. Machine 2 starts with processing product A, then product B twice. Machine 3 also starts with product A, then C. The sequencing of products on the machines is modelled via the so-called *command circuits*. A command circuit connects all timed transitions corresponding to the same machine. Figure 15 shows the complete model.

Observing the net structure tells us that the net is in fact a consistent ITEG. Therefore the analysis technique described in section 5.2 can be applied. This technique produces the minimum (maximum) cycle time by considering all (elementary) circuits. The are three types of circuits: processing circuits, command circuits and *mixed circuits*. These latter circuits include places of both processing and command circuits. If we supply (all the places of) the processing circuits with sufficient tokens then the minimum (maximum) cycle time is equal to minimum (maximum) cycle time of the command circuits. In this case the job-shop functions at maximal rate. In [Hillion et al. 89] a method is presented to minimise the jobs in-process (i.e. the tokens in the processing circuits).
In this particular example there are three command circuits: machine 1 (cycle time between 38 and 50 minutes), machine 2 (cycle time between 39 and 48 minutes) and machine 3 (cycle time between 15 and 36 minutes). Therefore the the minimum cycle time is equal to 39 minutes and the maximum cycle time is equal to 50 minutes. Machine 1 and machine 2

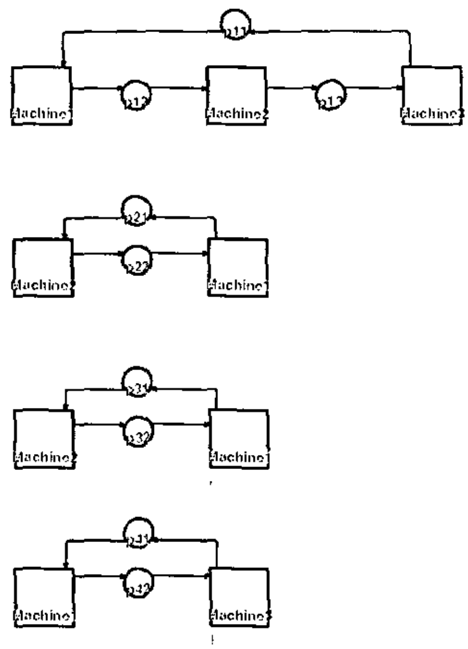


Figure 14: The processing circuits

are the bottleneck machines.

It is also possible to model the job-shop as shown in figure 16. Now the job-shop is modelled as an open system where supplies arrive via places $p11$, $p21$, $p31$ and $p41$. Products leave the system via the places $p14$, $p23$, $p33$ and $p43$. This net is an ITEG which satisfies the properties stated in theorem 5 (provided that all tokens initially available have a timestamp 0). Therefore we can analyse this net using IAT (method CFNRT) or a simulation with a timed coloured net. Some results are shown in the following table.

ordernumber	product A		product B				product C	
	line 1		line 2		line 3		line 4	
(n)	$\mathcal{EAT}_n$	$\mathcal{LAT}_n$	$\mathcal{EAT}_n$	$\mathcal{LAT}_n$	$\mathcal{EAT}_n$	$\mathcal{LAT}_n$	$\mathcal{EAT}_n$	$\mathcal{LAT}_n$
1	27	41	56	71	44	55	37	62
2	83	112	112	142	100	126	93	133
3	139	183	168	213	156	197	149	204
5	251	325	280	355	268	339	261	346
10	531	680	560	710	548	694	541	701
100	5571	7070	5600	7100	5588	7084	5581	7091

Note that (in steady-state) the interarrival time of two finished products of the same type

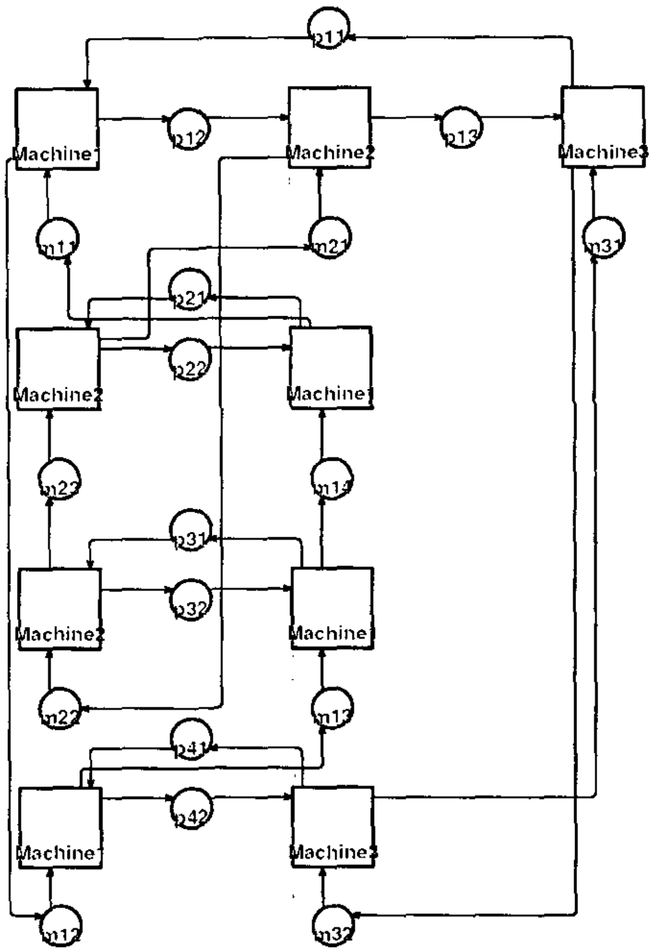


Figure 15: The job-shop system

is between 56 minutes and 71 minutes. This tells us that the job-shop is not functioning at maximal rate (i.e. the critical circuit is not a command circuit). If we increase the number of jobs in-process such that the places $p12$, $p13$, $p22$, $p32$ and $p42$ initially contain a token then a simulation gives the following results.

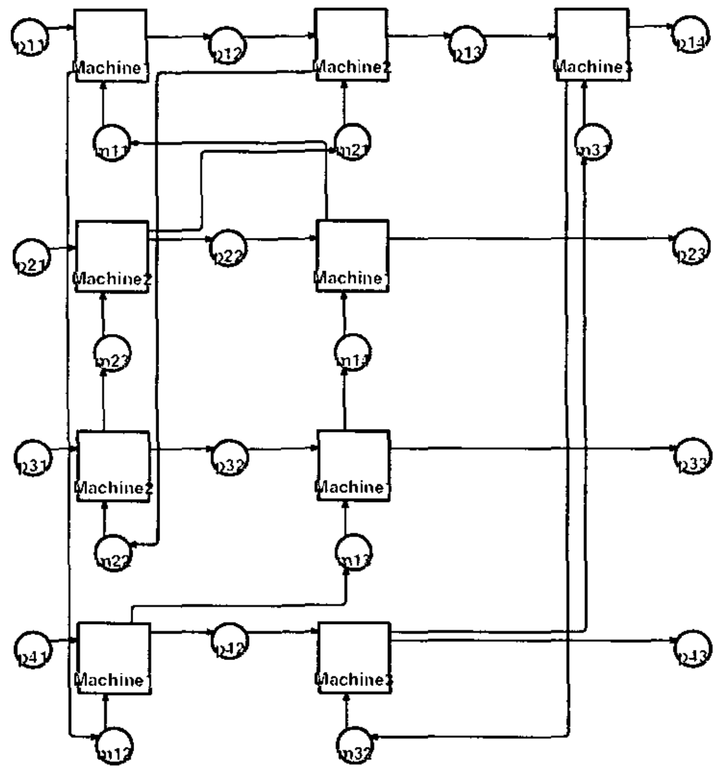


Figure 16: The job-shop system revisited

ordernumber	product A line 1		product B line 2		line 3		product C line 4	
(n)	$\mathcal{EAT}_n$	$\mathcal{LAT}_n$	$\mathcal{EAT}_n$	$\mathcal{LAT}_n$	$\mathcal{EAT}_n$	$\mathcal{LAT}_n$	$\mathcal{EAT}_n$	$\mathcal{LAT}_n$
1	5	15	38	50	28	37	15	36
2	20	51	76	100	66	87	30	72
3	59	87	114	150	104	137	69	108
5	137	175	190	200	180	237	147	196
10	332	415	380	500	370	487	342	445
100	3842	4910	3871	5000	3860	4987	3852	4945

A close observation of the results tells us that the behaviour is cyclic (the steady-state functioning starts after about 30 cycles). The throughput of the jobshop is at least $60/50 = 1.20$ items an hour, but no more than $60/39 = 1.54$ items an hour (for each product). Note that these figures match with the figures obtained by calculating the minimum (maximum) cycle time.

Note that it is possible to incorporate other kinds of repetitive production processes, for example the assembly of products.

7 Concluding remarks

In this paper a new Timed Petri Net model was introduced. Representing the time by means of an interval rather than deterministic or stochastic variables is promising because it allows for the representation of time constraints.
Four new analysis methods have been developed. The first one (ATCFN) answers a limited set of questions for a restricted class of nets. The second analysis method (MTSRT) can be used for several kinds of questions and for an arbitrary ITPN. It generates a reduced reachability tree. The third one calculates the final state in an ITEG. The last one analyses the steady-state performance of an ITEG.
The model is very useful when validating the dynamic behaviour of the system modelled. It is not meant to replace existing Stochastic Petri Net models but to support them. Especially in the field of time-critical systems our approach will prove to be useful.
We have developed a tool called ExSpect/IAT based on these analysis methods. Experience with this tool shows that the analysis methods produce useful results.

References

[Aalst et al. 89] Aalst, W.M.P van der, M. Voorhoeve and A. W. Waltmans, The TASTE project,
in: Proceedings of the 10th International Conference on Applications and Theory of Petri Nets, Bonn 1989.

[Aalst et al. 90] Aalst, W.M.P van der A.W. Waltmans, Modelling Flexible Manufacturing Systems with EXSPECT,
in: Proceedings of the 1990 European Simulation Multiconference, Nuremberg 1990.

[Aalst et al. 91] Aalst, W.M.P van der and A.W. Waltmans, Modelling logistic systems with EXSPECT,
in: Dynamic Modelling of Information Systems, editors: H.G. Sol and K.M. van Hee, North-Holland, 1991.

[Berthomieu et al. 83] Berthomieu B. and M. Menasche, An enumerative approach for analyzing Time Petri Nets,
in: Information Processing 83, IFIP, North-Holland 1983.

[Carlier et al. 87] Carlier J., Ph. Chretienne and C. Girault, Modelling scheduling problems with Timed Petri Nets,
in: Petri Nets, central models and their properties, (LNCS 188, Springer, Berlin, 1987) pp. 62-82.

[Chretienne 83] Chretienne P. Les réseaux de petri temporisés,
Univ. Paris VI,Thése d'Etat, France, 1983.

[Dijkstra 59] Dijkstra E.W., A note on two problems in connection with graphs,
Numerische Mathematic, 1, 1959.

[Florin et al. 82] Florin G.,Natkin S., Evaluation based upon Stochastic Petri Nets of the Maximum Throughput of a Full Duplex Protocol,
in: Informtik Fachberichte 52, Girealt C., Reisig W. eds, Springer-Verlag, 1982.

[Jensen 87] Jensen, K., Coloured Petri Nets,
in: Petri Nets, central models and their properties, (LNCS 188, Springer, Berlin, 1987) pp. 248-299.

[Hee et al. 89] Hee, K.M. van, L.J. Somers and M. Voorhoeve, Executable specifications for distributed information systems,
in: Information System Concepts: An In-depth Analysis, North-Holland 1989.

[Hee et al. 91] Hee, K.M. van, P.A.C. Verkoulen, Intergration of a Data Model and Petri Nets,
in: Proceedings of the 12th International Conference on Application and Theory of Petri Nets, Aarhus 1991.

[Hillion et al. 89] Hillion H.P., J.P Proth, Performance Evaluation of Job-Shop Systems Using Timed Event-Graphs,
in: IEEE Transactions of Automatic Control, vol. 34, no. 1, 1989.

[Marsan et al. 84] Ajmone Marsan M., G. Bablo and G. Conte, A Class of Generalised Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems,
in: ACM Transactions on Computer Systems, 2(2), May 1984.

[Marsan et al. 85] Ajmone Marsan M., G. Bablo, A. Bobbio, G. Chiola, G. Conte and A. Cumani, On Petri Nets with Stochastic Timing,
in: proc. Int. Workshop on Timed Petri Nets, IEEE, Torino, Italy 1985.

[Merlin 74] Merlin P., A Study of the Recoverability of Computer Systems,
Ph.D. Thesis, University of California, Irvine, 1974.

[Peterson 81] Peterson J.L., Petri net theory and the modeling of systems,
Prentice Hall 1981.

[Petri 80] Petri, C. Introduction to general net theory,
in: Brauer, W. (ed.), Net theory and applications (LNCS 84, Springer, Berlin, 1980) pp. 1-20.

[Price 71] Price W.L., Graphs and networks, an introduction,
Butterworths, London, 1971.

[Ramamoorthy et al. 80] Ramamoorthy C.V., G.S Ho, Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets,
in: IEEE Transactions of Software Engineering, vol. SE-6, no. 5, 1980.

[Sifakis 78] Sifakis, J., Performance Evaluation of Systems using Nets,
in: Brauer, W. (ed.), Net theory and applications (LNCS 84, Springer, Berlin, 1980) pp. 307-319.

[Silva 89] Silva, M. and R. Valette, Petri Nets and Flexible Manufacturing,
in: Advances in Petri Nets, (LNCS 424, Springer, Berlin, 1989) pp. 374-417.

[Whitehouse 73] Whitehouse G.E., Systems analysis and design using network techniques, Prentice Hall 1973.

[Zuberek 80] Zuberek, W.M., Timed Petri Nets and Preliminary Performance Evaluation,
in: Brauer, W. (ed.), The 7th Annual Symposium on Computer Architecture, (1980),pp. 88-96.

A Bags

Intuitively a *bag* is the same as a set, except for the fact that a bag may contain multiple occurrences of the same element. Another word for bag is *multiset*. A multiset is defined over a set A which means that elements of this multiset are taken from A. A multiset b over A is defined by a function from A to $\mathbb{N}$; $b \in A \rightarrow \mathbb{N}$. If $a \in A$ then $b(a)$ is the number of occurrences of a in the bag b. $\mathbb{B}(A)$ is the set of all multisets over A.
Most of the set operators can be applied to bags in a rather straightforward way.
We use square brackets to denote multisets by enumeration. Suppose A a set, $n \in \mathbb{N}$ and $q_0, q_1, .., q_n \in A$ then $[q_0, q_1, .., q_n] = \lambda_{a \in A}\ \#\{i \in \{0, .., n\} \mid q_i = a\}$. For example $[a, a, b, a]$ is the bag containing 3 elements a and one b.We use $[\ ]$ to denote the empty bag.

B Assignment problem

If $s \in S$, $\overline{s} \in \overline{S}$ and $s \lhd \overline{s}$ then there exists a function $f \in dom(s) \rightarrow dom(\overline{s})$ satisfying the conditions presented in section 2.1. The following lemma shows that if such a function f exists there also exists a function g satisfying the same constraints and the additional constraint that g is 'order-preserving'. This function g is some kind of *isomorphism* satisfying some additional constraints. We need this lemma to prove theorem 3. To simplify the notations we only consider the time of a token, not the position.

Lemma 6 (Assignment Problem)
If $q \in Id \not\rightarrow TS$ and $\overline{q} \in Id \not\rightarrow INT$ such that there exists a function $f \in dom(q) \rightarrow dom(\overline{q})$ with:

(i) f is bijective

(ii) $\forall_{i \in dom(q)}\ q(i) \in \overline{q}(f(i))$

then there also exists a function $g \in dom(q) \rightarrow dom(\overline{q})$ with:

(iii) g is bijective

(iv) $\forall_{i \in dom(q)}\ q(i) \in \overline{q}(g(i))$

(v) $\forall_{i,j \in dom(q)}\ q(i) \leq q(j) \Rightarrow \neg(\overline{q}(g(j)) <_i \overline{q}(g(i)))$

Proof.
It is easy to find a function g that satisfies (iii) and (iv) because f is such a function. In this proof we will show that it is possible to "transform" f until (v) holds (i.e. we give an algorithm to calculate g). First we define a linear (total) ordering ($\leq_l$) on $dom(q)$ such that $i \leq_l j \Rightarrow q(i) \leq q(j)$. This is possible because $q(i) \leq q(j)$ defines a pre-ordering (a pre-ordering (quasi-ordering) is reflexive and transitive).
Now we are able to define the *conflict set* of f:

$$C(f) = \{\langle i,j \rangle \in dom(q) \times dom(q) \mid i \leq_l j\ \wedge\ \overline{q}(f(i)) >_i \overline{q}(f(j))\}$$

Note that $C(f) = \emptyset$ implies that $\forall_{i,j \in dom(q)}\ q(i) \leq q(j) \Rightarrow \neg(\overline{q}(f(i)) >_i \overline{q}(f(j)))$.
Consider the following program to transform f (in pseudo code):

```
while  C(f) ≠ ∅
begin
       ⟨i,j⟩ ∈ C(f)
       { select an i and j in conflict }
       f := (f↾(dom(q) \ {i,j}) ∪ {⟨i,f(j)⟩,⟨j,f(i)⟩}
       { swap i and j }
end
```

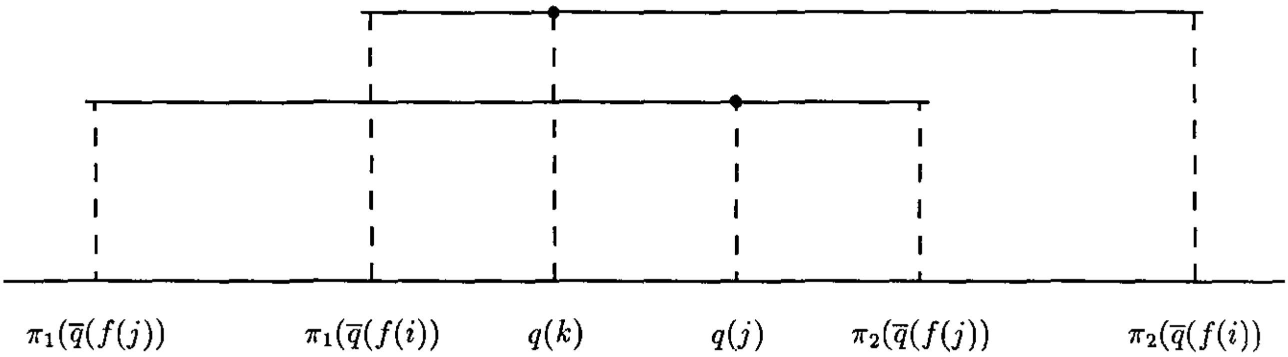
Because, $C(f) = \emptyset$ implies (v), it is sufficient to prove that (iii) and (iv) are invariant and that the program terminates.

First we prove that (iii) and (iv) are invariant. Initially both invariants hold because of the definition of f. Suppose (iii) and (iv) hold and $\langle i,j \rangle \in C(f)$ and
$\hat{f} := (f \restriction (dom(q) \setminus \{i,j\}) \cup \{\langle i, f(j)\rangle, \langle j, f(i)\rangle\}$
Now we have to show that both invariants hold for $\hat{f}$.

If f bijective then $\hat{f}$ also bijective ((iii) holds).

To prove (iv) we have to show that for any $k \in dom(q)$; $q(k) \in \overline{q}(\hat{f}(k))$.
(a) If $k \neq i$ and $k \neq j$ then $q(k) \in \overline{q}(f(k)) = \overline{q}(\hat{f}(k))$.
(b) If $k = i$ then $q(i) \in \overline{q}(f(i)) = \overline{q}(\hat{f}(j))$.
We also know that $q(i) \leq q(j)$ and $\overline{q}(f(i)) >_i \overline{q}(f(j))$ because $\langle i,j \rangle \in C(f)$.

The fact that $\overline{q}(f(i)) >_i \overline{q}(f(j))$ implies that $(\pi_1(\overline{q}(f(i))) \geq \pi_1(\overline{q}(f(j))))$ and $(\pi_2(\overline{q}(f(i))) \geq \pi_2(\overline{q}(f(j))))$. This situation is shown in the following figure:



$q(k) \geq \pi_1(\overline{q}(f(k))) \geq \pi_1(\overline{q}(f(j))) = \pi_1(\overline{q}(\hat{f}(k)))$
$q(k) \leq q(j) \leq \pi_2(\overline{q}(f(j))) = \pi_1(\overline{q}(\hat{f}(k)))$
So $q(k) \in \overline{q}(\hat{f}(k))$.
(c) A similar reasoning holds for $k = j$.

Finally we have to prove that the program terminates. Observe that there are only a finite number of bijective functions from $dom(q)$ to $dom(\overline{q})$ $((\#dom(q))!)$.
Using the linear ordering $\leq_l$ it is possible to construct a lexicographic ordering $(\leq_f)$ on the set of functions from $dom(q)$ to $dom(\overline{q})$: If $f, f' \in dom(q) \rightarrow dom(\overline{q})$ then:

$$\begin{aligned} f \leq_f f' \;\; \equiv \;\; & \exists_{k \in dom(q)} (\forall_{\substack{l \in dom(q) \\ l <_l k}} f(l) = f'(l)) \;\wedge\; \overline{q}(f(k)) <_i \overline{q}(f'(k))) \;\;\vee \\ & \forall_{k \in dom(q)} f(k) = f'(k) \end{aligned}$$

This ordering is a partial ordering because $\leq_i$ is a partial ordering. It is easy to verify that $\leq_f$ is reflexive and antisymmetric ($\leq_i$ is antisymmetric). The ordering is also transitive: $f \leq_f f'$ and $f' \leq_f f''$ implies that $f \leq_f f''$ ($\leq_i$ is transitive).

If $\hat{f}$ is the result of swapping i and j then $\hat{f} <_f f$, because $\forall_{\substack{l \in dom(q) \\ l <_l i}} \hat{f}(l) = f(l)$ and $\overline{q}(\hat{f}(i)) <_i \overline{q}(f(i)))$.

The fact that f is "descending" with respect to $\leq_f$ and that the number of possible functions is finite tells us that the algorithm will terminate. Therefore, there exists a function g that satisfies the conditions (iii),(iv) and (v).
□

C Relation between ITPN (ITEG) and activity networks

Network planning is an established technique for project planning. It is the logical step when a project becomes too complex to plan it just on intuition. There are three basic network types: *activity networks*, *event networks* and *precedence networks.*
In an activity networks, *activities* (or tasks) are represented by arcs each beginning and ending in an identifiable point in the planning network. These points are called *events* and are represented by circles. Events are instantaneous and activities are time consuming (i.e. they have a time duration). Figure 17 shows an activity network.

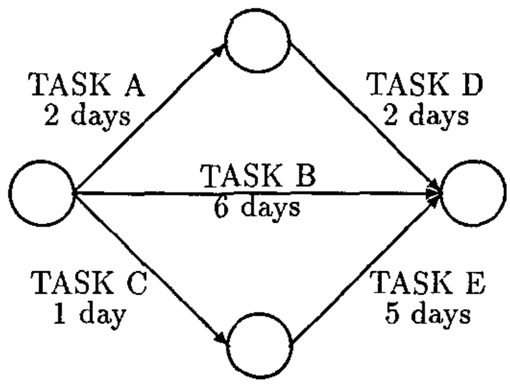


Figure 17: An activity network

An event network has a similar network structure, however the interpretation differs from an activity network. Arcs represent events, circles represent *milestones.* Now time is associated with events. Activity and event networks are frequently combined into *activity/event networks.*
In a precedence network an activity is represented by a box and arrows are used to define the relations between activities.

Two widespread network planning systems are the *CPM* (*Critical Path Method*) system and the *PERT* (*Program Evaluation and Review Technique*). They are both based on activity/event networks. In a PERT-network the time duration of an activity is specified by: an optimistic estimate, a pessimistic estimate and a most likely estimate.

An event (or milestone) is called a *start event* if there is no input arc. Events without output arcs are called *end events*. In general a planning network is acyclic and it has one start event and one end event.
The *critical path* in a planning network is the longest path from the start event to the end event. The project duration is given by the length of the critical path. The critical path can be calculated using a *forward calculation* (an activity starts if all previous activities have been completed) or *backward calculation* (an activity ends if one of next activities has

to start). A forward calculation produces the *earliest event time* of all events, a backward calculation produces the *latest event time* of all events.
The term *float time* is used to describe the amount of *extra* time available for the completion of an activity. There are various kinds of float time; *total float*, *free float*, *independent float*, these are calculated using a forward and backward calculation. For more information on network planning, see [Whitehouse 73].

Interval Timed Event Graphs are a generalisation of classical activity/event networks in the sense that they allow for the definition of optimistic and pessimistic estimates of the time durations and in the sense that they allow the study of repetitive schedulings.

An event (or milestone) in a planning network corresponds to a transition in an ITEG, an activity (or event) corresponds to a place. In other words replace the circles by transition bars and the arcs by places connecting two transitions. Figure 18 shows the ITEG net corresponding to the the activity net shown in figure 17.

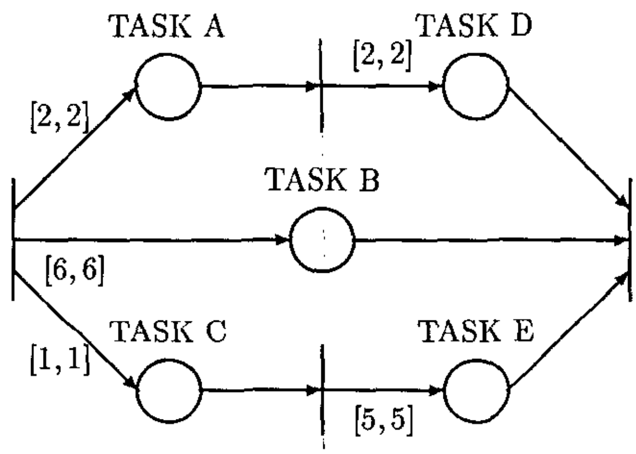


Figure 18: An ITEG representing an activity network

An ITEG constructed like this contains no circuits and has one transition without input places (start event) and one transition without any output places (end event). The transition without the input places fires once (at time 0), this can be modelled by an input place with initially one token with timestamp 0. A forward calculation can be done by applying method CFNRT, the results are upper and lower bounds for the earliest event time. By redirection of all arcs in the ITEG such a simulation produces upper and lower bounds for the latest event time. Therefore it is possible to calculate various kinds of float times.

Because circuits are allowed in an ITEG it is possible to study repetitive schedulings using the concept of critical cycles, discussed in section 5.2.

Conflict free Interval Timed Petri Nets are also a generalisation of activity/event networks with two node types: transition (**and** nodes) and places (**or** nodes). Because a place can have multiple input transitions it is possible to define alternatives. In section 3 we presented an algorithm to calculate upper and lower bounds for the length of a critical path.

In this series appeared:

89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a paralell program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.
89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.
89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.

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 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.
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.