Real-Time Systems, 9, 1–23 (1994)
© 1994 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Analysis of Railway Stations by Means of Interval Timed Coloured Petri Nets *

W.M.P. VAN DER AALST wsinwa@win.tue.nl
Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB, Eindhoven, The Netherlands

M.A. ODIJK odijk@twi.tudelft.nl
Department of Mathematics and Computing Science, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands

Abstract. In this paper *interval timed coloured Petri nets* ([3]) are used to model and analyse railway stations. We will show that this approach can be used to evaluate both station operating schedules and the infrastructure of a station.
An interval timed coloured Petri net (ITCPN) is a coloured Petri net extended with time; time is in tokens and transitions determine a delay for each produced token. This delay is specified by an upper and lower bound, i.e. an interval. The ITCPN model allows for the modelling of the dynamic behaviour of large and complex systems, without loosing the possibility of formal analysis. In addition to the existing analysis techniques for coloured Petri nets, we use a new analysis method to analyse the temporal behaviour of the net. This method constructs a reduced reachability graph and exploits the fact that delays are described by an interval. We will also discuss other (Petri net based) methods that can be used to analyse railway stations.

Keywords: Petri nets, analysis of timed Petri nets, railway stations

1. Introduction

The Dutch Railway Company (NS) is very interested in improving the handling of trains in large stations. This interest stems from the fact that the NS is currently involved in several large station development projects. Existing stations are extended by adding platform tracks, track sections, switches, etc. To be able to evaluate these costly station development projects, the NS is looking for tools and techniques to determine the quality of traffic control in large railway stations.

Given a new infrastructure the NS has to analyse various station operating schedules. The NS aims at station operating schedules that conform to a number of quality criteria:

- The station operating schedule should be robust, i.e. an operating schedule where a delay incurred by some train causes a cascade of delayed trains, should be avoided.

* By order of and in co-operation with the Dutch Railway Company (NS), Department of Traffic Development.

- There should be no unnecessary waiting of trains, i.e. the throughput times of trains should be as small as possible.
- If possible, trains with the same geographical destination should be assigned to the same (set of) platforms.

Moreover, the station operating schedule has to satisfy a number of constraints (e.g. safety constraints, technical constraints, etc.).

In cooperation with Bakkenist, Delft University of Technology and Eindhoven University of Technology, the NS has developed a general simulation model to evaluate both station operating schedules and the infrastructure of a station (see [21], [22]). They have used the software package *ExSpect* ([5]) for this purpose. ExSpect is based on a coloured Petri net model extended with time. Besides this simulation study, we also applied some more advanced Petri net based analysis techniques. For this purpose we have used the *Interval Timed Coloured Petri Net* (ITCPN) model and an analysis method, called *MTSRT*, that is based on this model ([2], [3]).

The ITCPN model uses a rather new timing mechanism where time is associated with tokens. This timing concept has been adopted from Van Hee et al. [10], [11]. In the ITCPN model we attach a *timestamp* to every token. This timestamp indicates the time a token becomes available. Associating time with tokens seems to be the natural choice for high-level Petri nets, since the colour is also associated with tokens. The *enabling time* of a transition is the maximum timestamp of the tokens to be consumed. Transitions are *eager* to fire (i.e. they fire as soon as possible), therefore the transition with the smallest enabling time will fire first. Firing is an atomic action, thereby producing tokens with a timestamp of at least the firing time. The difference between the firing time and the timestamp of such a produced token is called the *firing delay*. The (firing) delay of a produced token is specified by an *upper* and *lower bound*, i.e. an *interval*.

Instead of using 'interval timing', we could have used a Petri net model with fixed delays or stochastic delays.

Petri nets with fixed (deterministic) delays have been proposed in [11], [23], [25], [27]. They allow for simple analysis methods but are not very expressive, because in a real system the durations of most activities are variable.

One way to model this variability, is to assume certain delay distributions, i.e. to use a timed Petri net model with delays described by probability distributions. These nets are called *stochastic Petri nets* ([8], [16], [17]). Analysis of stochastic Petri nets is possible (in theory), since the reachability graph can be regarded, under certain conditions, as a Markov chain or a semi-Markov process. However, these conditions are severe: all firing delays have to be sampled from an exponential distribution or the topology of the net has to be of a special form (Ajmone Marsan et al. [16]). Since there are no general applicable analysis methods, several authors resorted to using simulation to study the behaviour of the net (see section 3 and subsection 4.4).

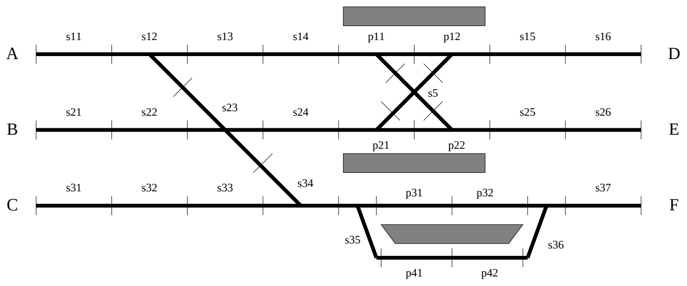

Figure 1. A railway station

To avoid these problems, we propose delays described by an *interval* specifying an upper and lower bound for the duration of the corresponding activity. On the one hand, interval delays allow for the modelling of variable delays, on the other hand, it is not necessary to determine some artificial delay distribution (as opposed to stochastic delays). Instead, we have to specify bounds. These bounds can be used to verify time constraints. This is very important when modelling time-critical systems, i.e. *real-time* systems with 'hard' deadlines. These hard (real-time) deadlines have to be met for a safe operation of the system. An acceptable behaviour of the system depends not only on the logical correctness of the results, but also on the time at which the results are produced. Clearly, the control of a railway junction is an example of such a system.

Now, let us focus on the the problem addressed by this paper. In this paper we study a station and its access lines, i.e. a railway junction. The network of platforms, accesslines, switches and track sections can be represented by a graph (see figure 1). In this particular example there are 6 access lines (`A`, `B`, ..) and 8 platforms or parts of a platform (`p11`, `p12`, ..). Note that platforms, switches and track sections are divided into logical *sections*. At any moment a section is either free of occupied. A section is occupied if it is *claimed* (*locked*) by some train. Two trains cannot occupy a section at the same time. There are three possible reasons for occupying a section: (1) a train resides on the section, (2) the train will visit the section in the near future or (3) the section is locked for safety reasons only.

We assume that the *route-locking sectional-release* principle (Bourachot [7]) is used, i.e. a train which arrives claims an entire route (a list of sections), a section is cleared when the train tail passes the release point of this section. An example of a route from entry point `A` to exit point `E` is: `s11,s12,s13,s14,p11,s5,p22,s25,s26`. A train t may enter the first section if and only if all the sections on that route are free and subsequently these sections become claimed by train t. A claimed section

is occupied and it cannot be claimed by any other train until it is released. If a train has to stop at a specific platform, then we identify two separate routes: one from the entry point to the platform and one from the platform to the exit point (e.g. `s11,s12,s13,s14,p11,s5,p22` and `p22,s25,s26`).

Given arrival times of trains, stopping times and timing characteristics of sections, platforms, trains, etc. we want to analyse: (1) throughput and waiting times of trains and (2) occupation rates of sections. In this paper we will show that we can use an approach based on interval timed coloured Petri nets to calculate upper and lower bounds for these performance figures.

In section 2 we introduce the ITCPN model. Section 3 deals with the analysis of interval timed coloured Petri nets. In this section, we describe the MTSRT analysis method. In section 4 we show how a station can be modelled in terms of an ITCPN. We will also show some analysis results obtained by applying the MTSRT method and other more traditional analysis methods.

2. Interval Timed Coloured Petri Nets

In this section we give an informal introduction to the ITCPN model. The formal definition is given in appendix A. For the formal semantics of the ITCPN model the reader is referred to [2] or [3]. We use an example to introduce the notion of interval timed coloured Petri nets. Figure 2 shows an ITCPN composed of four **places** (`train_in`, `busy_section`, `free_section` and `train_out`) and two **transitions** (`enter` and `leave`). At any moment, a place contains zero or more **tokens**, drawn as black dots. In the ITCPN model, a token has three attributes: a position, a value and a timestamp, i.e. we can use the tuple $\langle\langle p, v\rangle, x\rangle$ to denote a token in place p with value v and timestamp x. The value of a token is often referred to as the **token colour**. Each place has a **colour set** attached to it which specifies the set of allowed values, i.e. each token residing in place p must have a colour (value) which is a member of the colour set of p.

The ITCPN shown in figure 2 represents a section in a railway station, trains arrive via place `train_in` and leave the system via place `train_out`. Sections are either 'free' or 'busy'. Each section is represented by a token which is either in place `busy_section` or in place `free_section`. There are three colour sets $\mathcal{T}$ = { `T1, T2, T3, ..` }, $\mathcal{S}$ = { `S1, S2, S3, ..` } and $\mathcal{T} \times \mathcal{S}$. Colour set $\mathcal{T}$ (train types) is attached to place `train_in` and place `train_out`, colour set $\mathcal{S}$ (section identifiers) is attached to place `free_section`. Colour set $\mathcal{T} \times \mathcal{S}$ is attached to place `busy_section`.

Places and transitions are interconnected by **arcs**. Each arc connects a place and a transition in precisely one direction. Transition `enter` has two input places (`train_in` and `free_section`) and one output place (`busy_section`). Transition `leave` has one input place (`busy_section`) and two output places (`train_out` and `free_section`).

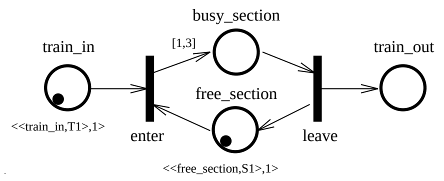

Figure 2. An interval timed coloured Petri net which models one section

Places are passive components, while transitions are the active components. Transitions cause state changes. A transition is called **enabled** if there are 'enough' tokens on each of its input places. In other words, a transition is enabled if all input places contain (at least) the specified number of tokens (further details will be given later). An enabled transition may **occur** (**fire**) at time x if all the tokens to be consumed have a timestamp not later than time x. The **enabling time** of a transition is the maximum timestamp of the tokens to be consumed. Because transitions are eager to fire, a transition with the smallest enabling time will fire first.

Firing a transition means consuming tokens from the input places and producing tokens on the output places. If, at any time, more than one transition is enabled, then any of the several enabled transitions may be 'the next' to fire. This leads to a non-deterministic choice if several transitions have the same enabling time.

Firing is an atomic action, thereby producing tokens with a timestamp of at least the firing time. The difference between the firing time and the timestamp of such a produced token is called the **firing delay**. This delay is specified by an **interval**, i.e. only delays between a given upper bound and a given lower bound are allowed. In other words, the delay of a token is 'sampled' from the corresponding delay interval. Note that the term 'sampled' may be confusing, because the modeller does not specify a probability distribution, merely an upper and lower bound.

Moreover, it is possible that the modeller specifies a delay interval which is too wide, because of a lack of detailed information. In this case, the actual delays (in the real system) only range over a part of the delay interval.

The number of tokens produced by the firing of a transition may depend upon the values of the consumed tokens. Moreover, the values and delays of the produced tokens may also depend upon the values of the consumed tokens. The relation between the *multi-set* of consumed tokens and the *multi-set* of produced tokens is described by the **transition function**. Function $F(\texttt{enter})$ specifies transition `enter` in the net shown in figure 2:

$$dom(F(\texttt{enter})) = \{\langle \texttt{train\_in}, t\rangle + \langle \texttt{free\_section}, s\rangle \mid t \in \mathcal{T} \quad \text{and} \quad s \in \mathcal{S}\}$$

For $t \in \mathcal{T}$ and $s \in \mathcal{S}$, we have:

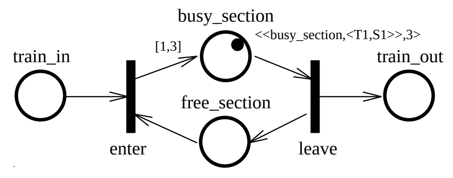

Figure 3. Transition `enter` has fired

$$F(\texttt{enter})(\langle \texttt{train\_in}, t\rangle + \langle \texttt{free\_section}, s\rangle) = \langle\langle \texttt{busy\_section}, \langle t, s\rangle\rangle, [1,3]\rangle$$

(Note that $\langle \texttt{train\_in}, t\rangle + \langle \texttt{free\_section}, s\rangle$ and $\langle\langle \texttt{busy\_section}, \langle t, s\rangle\rangle, [1,3]\rangle$ are multi-sets, see appendix A.1.) The domain of $F(\texttt{enter})$ describes the condition on which transition `enter` is enabled, i.e. `enter` is enabled if there is (at least) one token in place `train_in` and one token in `free_section`. This means that transition `enter` may occur if there is a train waiting to enter the section and the section is free. Note that, in this case, the enabling of a transition does not depend upon the values of the tokens consumed. The enabling time of transition `enter` depends upon the timestamps of the tokens to be consumed. If `enter` occurs, it consumes one token from place `train_in` and one token from `free_section` and it produces one token for place `busy_section`. The colour of the produced token is a pair $\langle t, s\rangle$, where t represents the train and s represents the section. The delay of this token is an arbitrary value between 1 and 3, e.g. 2, 2.55 or 4/3. The situation shown in figure 3 is the result of firing `enter` in the state shown in figure 2. In this case the delay of the token produced for `busy_section` was equal to 2.

Transition `leave` is specified as follows:

$$dom(F(\texttt{leave})) = \{\langle \texttt{busy\_section}, \langle t, s\rangle\rangle \mid t \in \mathcal{T} \quad \text{and} \quad s \in \mathcal{S}\}$$

For $t \in \mathcal{T}$ and $s \in \mathcal{S}$, we have:

$$F(\texttt{leave})(\langle \texttt{busy\_section}, \langle t, s\rangle\rangle) = \\ \langle\langle \texttt{train\_out}, t\rangle, [0,0]\rangle + \langle\langle \texttt{free\_section}, s\rangle, [0,0]\rangle$$

Transition `leave` is used to represent trains leaving the section. If `leave` occurs, it consumes one token from place `busy_section` and it produces two tokens (one for `train_out` and one for `free_section`) both with a delay equal to zero. If `leave` occurs in the state shown in figure 3, then the resulting state contains two tokens: $\langle\langle \texttt{train\_out}, T1\rangle, 3\rangle$ and $\langle\langle \texttt{free\_section}, S1\rangle, 3\rangle$.

3. The MTSRT Method

In the ITCPN model, a delay is described by an interval rather than a fixed value or some delay distribution. On the one hand, interval delays allow for the modelling of variable delays, on the other hand, it is not necessary to determine some artificial delay distribution (as opposed to stochastic delays). These delay intervals are used to specify bounds for durations. We will use these bounds to verify time constraints. This is very important when modelling time-critical systems, i.e. *real-time* systems with 'hard' deadlines. An acceptable behaviour of the system depends not only on the logical correctness of the results, but also on the time at which the results are produced. Therefore, we are interested in techniques to verify these deadlines and to calculate upper and lower bounds for all sorts of performance criteria. This is the reason we developed the **Modified Transition System Reduction Technique** (MTSRT), which was presented in [2] and [3]. Before giving a short description of this analysis method, we provide a brief survey of existing techniques which can be used to analyse the dynamic behaviour of timed and coloured Petri nets. The techniques may be subdivided into three classes: simulation, reachability analysis and Markovian analysis.

Simulation is a technique to analyse a system by conducting controlled experiments. Because simulation does not require difficult mathematical techniques, it is easy to understand for people with a non-technical background. Simulation is also a very powerful analysis technique, since it does not set additional restraints. However, sometimes simulation is expensive in terms of the computer time necessary to obtain reliable results. Another drawback is the fact that (in general) it is not possible to use simulation to *prove* that the system has the desired set of properties.

Recent developments in computer technology stimulate the use of simulation for the analysis of timed coloured Petri nets. The increased processing power allows for the simulation of large nets. Modern graphical screens are fast and have a high resolution. Therefore, it is possible to visualize a simulation graphically (i.e. animation).

Reachability analysis is a technique which constructs a reachability graph, sometimes referred to as reachability tree or occurrence graph (cf. Jensen [13], [15]). Such a reachability graph contains a node for each possible state and an arc for each possible state change. Reachability analysis is a very powerful method in the sense that it can be used to prove all kinds of properties. Another advantage is the fact that it does not set additional restraints. Obviously, the reachability graph needed to prove these properties may, even for small nets, become very large (and often infinite). If we want to inspect the reachability graph by means of a computer, we have to solve this problem. This is the reason several authors developed reduction techniques (Hubner et al. [12] and Valmari [26]). Unfortunately, it is not known how to apply these techniques to timed coloured Petri nets.

For timed (coloured) Petri nets with certain types of stochastic delays it is possible to translate the net into a *continuous time Markov chain*. This Markov chain can be used to calculate performance measures like the average number of tokens in a place and the average firing rate of a transition.

If all the delays are sampled from a negative exponential probability distribution, then it is easy to translate the timed Petri net into a continuous time Markov chain.

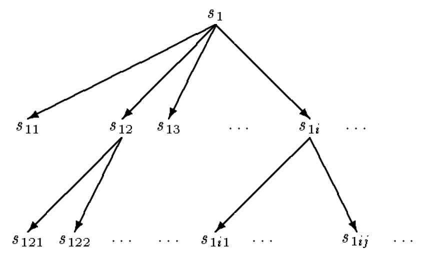

Figure 4. A reachability graph

Several authors attempted to increase the modelling power by allowing other kinds of delays, for example mixed deterministic and negative exponential distributed delays, and phase-type distributed delays (see Ajmone Marsan et al. [16]). Only a few stochastic Petri net models (and related analysis techniques) allow for coloured tokens, because the increased modelling power is offset by computational difficulties. This is the reason stochastic high-level Petri nets are often used in a simulation context only.

Besides the aforementioned techniques to analyse the behaviour of timed coloured Petri nets, there are several analysis techniques for Petri nets without 'colour' or explicit 'time'. As an example, we mention the generation of place and transition invariants, which may be used to verify properties which are time independent. For more information about the calculation of invariants in a coloured Petri net, see Jensen [13], [15].

The **Modified Transition System Reduction Technique** is a technique which generates the **reduced reachability graph** to answer all kinds of questions.

If we try to construct the reachability graph of an ITCPN in a straightforward manner we get into problems. The basic idea of a reachability graph is to organize all reachable markings in a graph, where each node represents a state and each arc represents an event transforming one state into another state. Consider for example the reachability graph shown in figure 4. Suppose that s_1 is the initial state of the ITCPN we want to consider. This state is connected to a number of states $s_{11}, s_{12}, s_{13}, ..$ reachable from s_1 by the firing of some transition, i.e. $s_1 \longrightarrow s_{1i}$. These states are called the 'successors' (or children) of the s_1. Repeating this process produces the graphical representation of the reachability graph, see figure 4. Such a reachability graph contains all relevant information about the dynamic behaviour of the system. If we are able to generate this graph, we can answer many questions about the behaviour of the system. However, for an ITCPN this graph is generally *infinite*! This is mainly caused by the fact that we use interval timing. Consider an enabled transition t. In general, there is an infinite number of allowed firing delays, all resulting in a different state. If transition t produces a token for a place with a delay x specified by the delay interval $[1,3]$, then every delay x between 1 and 3 is allowed. Moreover, every x leads to a different state. Since one firing already results in a 'fan-out' of reachable states, the reachability graph cannot be used to analyse the system.

To avoid this fan-out problem, we propose a reduction which aggregates states into **state classes**. Informally speaking, state classes are defined as the union of similar states having the same token distribution (marking) but different timestamps (within certain bounds).

A state s of an ITCPN is a multi-set of tuples $\langle\langle p,v\rangle,x\rangle$. Each tuple $\langle\langle p,v\rangle,x\rangle$ corresponds to one token in the net; p is the location of the token (i.e. the place where it resides), v is the value (colour) of the token and x is the timestamp of the token (i.e. the time it becomes available).

A state class $\overline{s}$ is also a multi-set of tuples $\langle\langle p,v\rangle,[y,z]\rangle$. Each tuple also corresponds to one token in the net. However, instead of a timestamp each token has a **time-interval**. Each state class corresponds to a set of states and vice versa. State class $\overline{s}$ corresponds to state s if and only if there is a bijection between the tokens in $\overline{s}$ and s such that $\langle\langle p,v\rangle,[y,z]\rangle$ is mapped onto $\langle\langle p,v\rangle,x\rangle$ with $x \in [y,z]$ (i.e. a token residing in the same place, having the same value and a timestamp which is within the time-interval $[y,z]$). We can think of these state classes as some kind of equivalence classes.

By using this reduction each token bears a time-interval instead of a timestamp. Therefore, we have to modify the firing rules described in section 2.

A transition is still enabled if there are enough tokens on each of its input places. However, the enabling time of a transition t is given by an interval! The lower bound of this interval is the **minimal enabling time** and upper bound of this interval is the **maximal enabling time** of t. These bounds are calculated by taking the maximum of the upper and lower bounds of the time-intervals of the tokens to be consumed respectively. We will use an example to clarify the modified firing rule.

Consider the net shown in figure 5. Initially, there is one token in place $p1$ with an interval of $[0,3]$, there is one token in $p2$ with an interval of $[2,5]$ and there is one token in $p3$ with an interval of $[4,6]$. Note that this state class $\overline{s}$ corresponds to an infinite number of states in the original reachability graph, for instance the state with a token in $p1$ with timestamp 2.4 and a token in $p2$ with timestamp π and a token in $p3$ with timestamp 31/6. Both transitions are enabled. If transition $t1$ fires first, then the tokens in $p1$ and $p2$ are consumed, if transition $t2$ fires first, then the tokens in $p2$ and $p3$ are consumed. The enabling time of $t1$ is between 2 ($ET_{min}(t1)$) and 5 ($ET_{max}(t1)$), the enabling time of $t2$ is between 4 ($ET_{min}(t2)$)

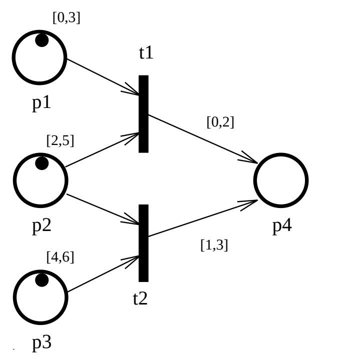

Figure 5. An example used to illustrate the modified firing rule

and 6 ($ET_{max}(t2)$). The transition with the smallest enabling time will fire first. Since the intervals associated to the enabling times of the transitions (i.e. $[2,5]$ and $[4,6]$) overlap it is not determined whether $t1$ or $t2$ fires first. However, the upper bound of the transition time ($MT_{max}(\overline{s})$) is equal to 5, i.e. a transition will fire before or at time 5. If $t1$ fires, it will be between 2 ($ET_{min}(t1)$) and 5 ($MT_{max}(\overline{s})$). If $t2$ fires, it will be between 4 ($ET_{min}(t2)$) and 5 ($MT_{max}(\overline{s})$). 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 in the intervals $[0,2]$ and $[1,3]$, i.e. it suffices to consider upper and lower bounds.

In Van der Aalst [2] and [3] a formal definition of these alternative semantics are given. If we use these semantics to construct a reachability graph, we obtain the reduced reachability graph which is finite for any practical application (see [2]). The alternative semantics have been introduced for computational reasons only. However, calculating the reduced reachability graph only makes sense if the reduced reachability graph can be used to deduce properties of the original reachability graph which represents the behaviour of the ITCPN. Therefore, we have to prove that there exists some meaningful relationship between the original reachability graph and the reduced reachability graph. Fortunately, the alternative semantics are 'sound' which means that any state reachable in the original reachability graph is also reachable in the reduced reachability graph. A formal proof is given in [2]

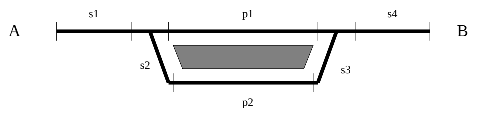

Figure 6. A simple railway station

and [3]. In these references it is also shown that the opposite is not true, i.e. the alternative semantics are not 'complete'.

Despite the non-completeness, the soundness property allows us to answer various questions. We can *prove* that a system has a desired set of properties by proving it for the modified transition system. For example, we can often use the reduced reachability graph to prove boundedness, absence of traps and siphons (deadlocks), etc. The reduced reachability graph may also be used to analyse the *performance* of the system modelled by an ITCPN. With performance we mean characteristics, such as: response times, occupation rates, transfer rates, throughput times, failure rates, etc. The MTSRT method can be used to calculate *bounds* for these performance measures. Although these bounds are sound (i.e. safe) they do not have to be as tight as possible, because of possible dependencies between tokens (non-completeness). However, experimentation shows that the calculated bounds are often of great value and far from trivial. Moreover, we are able to answer questions which cannot be answered by simulation or the method proposed by Berthomieu et al. [6].

We have modelled and analysed many examples using the approach presented in this paper, see Van der Aalst [1], [2] and Odijk [20]. To facilitate the analysis of real-life systems we have developed an analysis tool, called *IAT* ([2]). This tool also supports more traditional kinds of analysis such as the generation of place and transition invariants. IAT is part of the software package *ExSpect* (see ASPT [5], Van Hee et al. [11] and Van der Aalst [2], [4]).

4. Analysis of railway stations

4.1. Modelling a railway station

In this section we show how to model a railway station, i.e. a junction of railways, in terms of an interval timed coloured Petri net. Figure 6 shows a railway junction which is divided into a number of sections. Each of the sections `p1` and `p2` corresponds to a platform. There are two switches; one in section `s2` and one in `s3`. We assume that every train arrives via entry point `A` and leaves via exit point `B`. In this case there are two routes, one via platform `p1` (route `r1`) and the other one via platform `p2` (route `r2`).

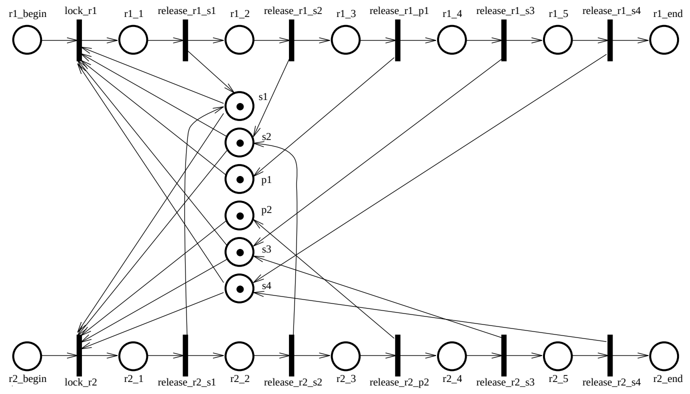

Figure 7. An ITCPN for a simple railway junction

When a train enters a section, it takes between 0.40 and 0.45 minutes to release this section. Stopping alongside a platform takes between 4 and 6 minutes. We can specify a route by a list of sections and durations:

```
r1:  s1[0.4,0.45]  s2[0.4,0.45]  p1[4,6]  s3[0.4,0.45]  s4[0.4,0.45]
r2:  s1[0.4,0.45]  s2[0.5,0.55]  p2[4,6]  s3[0.5,0.55]  s4[0.4,0.45]
```

Note that each duration is specified by an interval, i.e. an optimistic and a pessimistic estimate. Given such a specification of the possible routes inside the railway station, we have sufficient information to construct the corresponding ITCPN shown in figure 7. Each section corresponds to a place (`s1, s2, p1, p2, s3` and `s4`) and each route corresponds to a subnet which locks and releases sections.

We use the route-locking sectional-release principle. Trains for route `r1` enter via place `r1_begin`. If such a train arrives and the sections `s1, s2, p1, s3` and `s4` are free, then transition `lock_r1` fires. (Otherwise the train has to wait until the specified sections are released.) Transition `lock_r1` consumes a token from each place which represents a section on route `r1`. Transition `lock_r1` produces a token for place `r1_1` with a delay between 0.40 and 0.45 time units. Transition `release_r1_s1` releases section `s1` and produces a token for place `r1_2` with a delay between 0.40 and 0.45 time units, etc. Transition `release_r1_s4` releases the last section locked by the train and produces a token for `r1_end`. Note that each of the places `r1_begin, r1_1, r1_2, ..  ,r1_end` corresponds to a stage in process of using route `r1`. Route `r2` is modelled in a similar way (see figure 7).

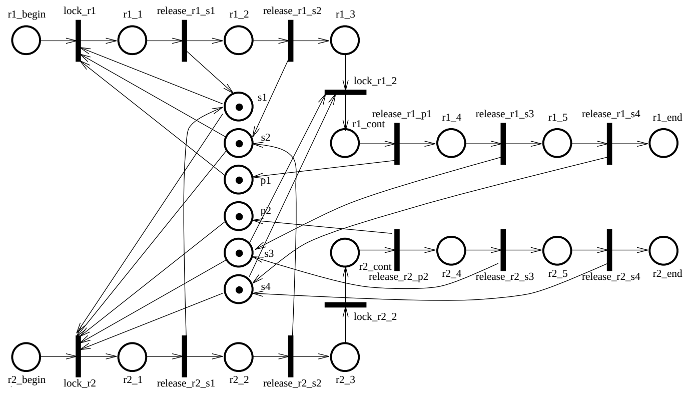

Figure 8. The modified ITCPN

To support the construction of such a net, we have developed a tool which automatically translates the specification of the possible routes into an ITCPN.

We have to modify the net shown in figure 7 if there are trains that claim only a part of the route. Consider for example a train locking the route from the entry point until the platform. When this train is ready to leave the platform, the second part of the route is locked. We can modify the net shown in figure 7 to model this principle, see figure 8. A train which follows route `r1` first locks sections `s1,s2` and `p1`, then just before this train leaves the platform `p1`, the sections `s3` and `s4` are locked.

It is also possible to model alternative routes, etc. Since the nets can be generated automatically the size and complexity of these nets is not a real problem. (Note that the size of the net is linear in the number of routes and sections.)

In the remainder of this section we will show how we can analyse such a net which represents railway junction. First, we will discuss the application of analysis techniques to derive structural properties of the net. Secondly we will apply the MTSRT to analyse the dynamic behaviour of a railway station. Finally, we focus on the simulation study that has been performed to analyse the railway station in Arnhem (The Netherlands).

4.2. Structural analysis

Several analysis methods have been developed to find and verify structural properties of classical Petri nets ([19], [24], [18]). Although some of these techniques have been extended to nets with 'colour', most of them are only feasible for uncoloured nets. Since the nets considered in this section are uncoloured, we can apply these methods without any problems.

Let us start with the calculation of *place* and *transition invariants* for the net shown in figure 7.

A place invariant (P-invariant) is a weighted token sum, i.e. a weight is associated with every token in the net. This weight is based on the location (place) and the value (colour) of the token. A place invariant holds if the weighted token sum of all tokens remains constant during the execution of the net. Consider for example the following place invariant:

`r1_begin + r1_1 + r1_2 + r1_3 + r1_4 + r1_5 + r1_end` $= C$

This invariant says that trains on route `r1` do not get 'lost', i.e. the total number of tokens in the places `r1_begin, r1_1, r1_2, r1_3, r1_4, r1_5` and `r1_end` cannot be changed by the firing of any transition. Another invariant is:

`s1 + r1_1 + r2_1` $= 1$

This invariant shows that section `s1` is 'safe', i.e. only one train is allowed to lock a section at the same time. Similar invariants hold for route `r2` and the other sections.

Transition invariants (T-invariants) are the duals of place invariants and the basic idea behind them is to find firing sequences with no effects, i.e. firing sequences which reproduce the initial state. The net shown in figure 7 has no transition invariants. However, if we add a transition `r1_wait` with input place `r1_end` and output place `r1_begin`, then we find the following transition invariant:

```
lock_r1 + release_r1_s1 + release_r1_s2 + release_r1_p1 +
   release_r1_s3 + release_r1_s4
```

This means that firing each of these transitions once results in the initial state. Many algorithms and tools have been developed to calculate invariants (see [2], [19], [24], [18]).

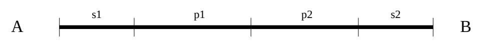

Figure 9. A railway station with a potential 'deadlock'

There are also some less trivial structural properties that can be investigated by applying Petri net based analysis techniques. Consider for example the station modelled in figure 9. Suppose there are two routes; one from `A` to `B` (`r1`) and one from `B` to `A` (`r2`). Trains using route `r1` stop alongside platform `p1` and trains using route `r2` stop alongside platform `p2`. Suppose two trains arrive at the same time from different directions; train 1 locks `s1` and `p1` and train 2 locks `s2` and `p2` (see section 4.1). Now both trains become blocked, i.e. a 'deadlock' occurs. For this simple railway station the existence of this deadlock is obvious, however for real stations it is hard to detect these potential deadlocks. Note that in a terminal station there may be many potential deadlocks.

Fortunately, we can find these deadlocks by applying Petri net based analysis techniques. First, we connect the entry and exit places by adding transitions (e.g. `r1_wait`). Then, we determine whether the net is *structurally live*. A net is structurally live if and only if there is an initial state s such that for any transition t and any state reachable from s there is firing sequence possible that fires transition t (see Murata [19]). There is a direct relation between the blocking of a train and this liveness property; a deadlock of trains is possible if and only if the net is *not* structurally live. There are special techniques to determine liveness. We can also construct the occurrence graph to find these deadlocks.

4.3. Applying the MTSRT method

We are also interested in the dynamic properties of an ITCPN which models a railway station. Interesting performance measures are:

- throughput (or waiting) times of trains,
- occupation rates of sections.

Consider for example the railway station shown in figure 10. For this railway station we define the following set of routes:

1. ADa: s11[0.4,0.45] s12[0.4,0.45] s13[0.4,0.45] s14[0.4,0.45] p11[4.0,6.0] p12[0.0,0.0] s15[0.4,0.45] s16[0.4,0.45]
2. AEa: s11[0.4,0.45] s12[0.4,0.45] s23[0.4,0.45] s24[0.4,0.45] p21[4.0,6.0] p22[0.0,0.0] s25[0.4,0.45] s26[0.4,0.45]
3. AEb: s11[0.4,0.45] s12[0.4,0.45] s13[0.4,0.45] s14[0.4,0.45] p11[0.4,0.45] s5[0.4,0.45] p22[0.4,0.45] s25[0.4,0.45] s26[0.4,0.45]
4. AFa: s11[0.4,0.45] s12[0.4,0.45] s23[0.4,0.45] s34[0.4,0.45] s35[0.4,0.45] p31[4.0,6.0] p32[0.0,0.0] s36[0.4,0.45] s37[0.4,0.45]
5. AFb: s11[0.4,0.45] s12[0.4,0.45] s23[0.4,0.45] s34[0.4,0.45] s35[0.4,0.45] p41[4.0,6.0] p42[0.0,0.0] s36[0.4,0.45] s37[0.4,0.45]
6. BDa: s21[0.4,0.45] s22[0.4,0.45] s23[0.4,0.45] s24[0.4,0.45] p21[4.0,6.0] s5[0.0,0.0] p12[0.0,0.0] s15[0.4,0.45] s16[0.4,0.45]
7. BEa: s21[0.4,0.45] s22[0.4,0.45] s23[0.4,0.45] s24[0.4,0.45] p21[4.0,6.0] p22[0.0,0.0] s25[0.4,0.45] s26[0.4,0.45]

8. BFa: s21[0.4,0.45] s22[0.4,0.45] s23[0.4,0.45] s34[0.4,0.45] s35[0.4,0.45] p31[4.0,6.0] p32[0.0,0.0] s36[0.4,0.45] s37[0.4,0.45]
9. BFb: s21[0.4,0.45] s22[0.4,0.45] s23[0.4,0.45] s34[0.4,0.45] s35[0.4,0.45] p41[4.0,6.0] p42[0.0,0.0] s36[0.4,0.45] s37[0.4,0.45]
10. CFa: s31[0.4,0.45] s32[0.4,0.45] s33[0.4,0.45] s34[0.4,0.45] s35[0.4,0.45] p31[4.0,6.0] p32[0.0,0.0] s36[0.4,0.45] s37[0.4,0.45]
11. CFb: s31[0.4,0.45] s32[0.4,0.45] s33[0.4,0.45] s34[0.4,0.45] s35[0.4,0.45] p41[4.0,6.0] p42[0.0,0.0] s36[0.4,0.45] s37[0.4,0.45]

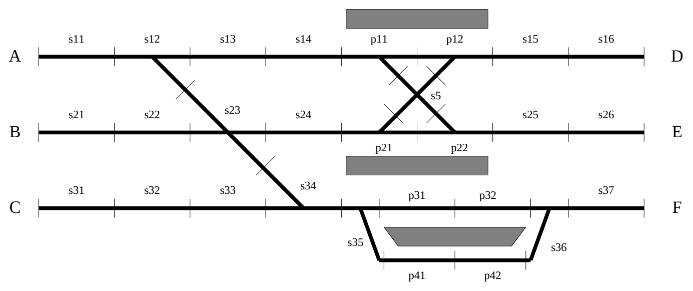

Figure 10. A railway station

Route `AFb` runs from entry point `A` to exit point `B` via platform `p41`. Trains using route `AEb` move from entry point `A` to exit point `E` via section `s5` without stopping. The time required to pass a section is assumed to be between 0.40 minutes and 0.45 minutes. Note that this interval may depend upon the characteristics of the route and the section.

We can automatically transform this description into an interval timed coloured Petri net as shown in section 4.1. The generated net contains 107 transitions and 146 places, 28 places are used to represent the sections (`s11, .. s37`) and the other places mark stages of trains using a specific route.

The initial state of this net is generated on the basis of a given timetable. Table 1 shows the first timetable we are going to analyse. The timetable describes arrivals of trains for a specific hour. Since most stations use an hourly schedule during daytime, it is reasonable to analyse one hour in isolation. Each place which marks the beginning of a route contains the appropriate number of tokens bearing a proper timestamp, e.g. place `ADa_begin` contains two tokens, one with timestamp 0 and the other with timestamp 30.

Since we have an ITCPN and an initial state, we are able to apply the MTSRT

Table 1. A timetable

train on route	arrival time
`ADa`	0
	30
`AEa`	35
`AFa`	25
`BDa`	40
`BEa`	20
	50
`BFa`	15
`CFa`	20
`CFb`	10

Table 2. The calculated upper and lower bounds for throughput and waiting times of trains

route	throughput time		waiting time	
	min	max	min	max
`ADa`	6.4	8.7	0.0	0.0
	6.4	8.7	0.0	0.0
`AEa`	6.4	12.5	0.0	3.8
`AFa`	6.8	21.6	0.0	12.45
`BDa`	7.0	15.3	0.6	6.6
`BEa`	6.4	9.2	0.0	0.5
	6.4	13.1	0.0	4.4
`BFa`	8.6	13.3	1.8	4.15
`CFa`	10.4	26.6	3.6	17.45
`CFb`	6.8	9.15	0.0	0.0

method described in section 3. The MTSRT method generates the reduced reachability graph which can be used to calculate bounds for all kinds of performance measures. Although the occupation of sections is also interesting, we focus on the throughput and waiting times of trains.

In this case the reduced reachability graph contains 1203 states. (Generating this graph takes about 15 seconds on a SUN-SPARC1 workstation.) By inspecting this graph we can deduce upper and lower bounds for the arrival times of tokens in each place. This information can be used to calculate bounds for throughput and waiting times of trains. Table 2 shows some of these results. The train which starts route `AFa` at time 25 has a throughput time between 6.8 and 21.6 minutes. This implies that the waiting time is between 0.0 and 12.45 minutes. This means that it may happen that a train is delayed for nearly 13 minutes. Obviously, this is not acceptable. (The train on route `CFa` may even have a delay of 17.45 minutes!)

There are several ways to deal with this problem. On the long term, we may invest in additional sections. On the short term we may be able to improve the station operating schedule by changing the routes through the station and changing the timetable.

Table 3. The improved timetable

train on route	arrival time
`ADa`	0
	30
`AEa`	15
`AFa`	45
`BDa`	50
`BEa`	0
	30
`BFa`	15
`CFa`	0
`CFb`	30

Table 4. The results for the improved timetable

route	throughput time		waiting time	
	min	max	min	max
`ADa`	6.4	8.7	0.0	0.0
	6.4	8.7	0.0	0.0
`AEa`	6.4	10.5	0.0	1.8
`AFa`	6.8	9.15	0.0	0.0
`BDa`	6.4	8.7	0.0	0.0
`BEa`	6.4	8.7	0.0	0.0
	6.4	8.7	0.0	0.0
`BFa`	6.8	10.5	0.0	1.35
`CFa`	6.8	9.15	0.0	0.0
`CFb`	6.8	9.15	0.0	0.0

Let us adapt the timetable to reduce waiting times. Observing the reduced reachability graph shows that there are a lot of 'conflicts' between trains with destination `F`. We can use this information to construct the improved timetable shown in table 3. The results for the new timetable are shown in table 4. Clearly, this new schedule is a major improvement. There are only two trains having a potential delay, the maximal delay of the train on route `AEa` is 1.8 minutes and the maximal delay of the train on route `BFa` is 1.35 minutes. Note that these figures are 'hard', i.e. it can be *proved* that given the description in terms of an ITCPN these trains have no or minor delays. This is important for a time-critical system like a railway station, because an operating schedule where a delay incurred by some train causes a cascade of delayed trains, should be avoided. By changing the delay intervals in the ITCPN the robustness of an operating schedule can be tested.

In this particular example the token values do not matter. However, the MTSRT method can be applied to arbitrary ITCPN's, i.e. also nets with coloured tokens. See Van der Aalst [2], [3] for examples.

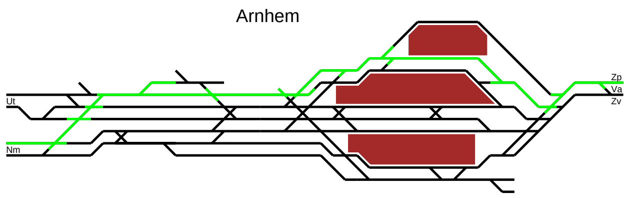

Figure 11. An animation of the railway station Arnhem

4.4. Simulation

Another approach to this problem was performed by the Dutch Railway Company (NS) in cooperation with Bakkenist, Delft University of Technology and Eindhoven University of Technology. This approach is also based of Petri nets, but uses simulation to analyse station operating schedules and infrastructures. The software package ExSpect has been used to model the dynamic behaviour of trains in a station in much more detail. Parameters of this model are lengths of platforms and sections, characteristics of trains (length, acceleration, maximum speed, etc.), track layout, alternative routes, preference structures, etc. The graphical interface can be used enter these parameters and the resulting Petri net can be simulated by ExSpect ([21], [22]). During the simulation it is possible to monitor train movements, conflicts, etc. At the end of each run statistics like average waiting times, occupation rates, etc. are reported. It is even possible to animate the railway station, see figure 11.

If we compare this simulation approach with the approach based on the ITCPN model, we see some striking differences. The simulation model is much closer to the actual situation than the ITCPN model described in section 4.1. However, there is a lot of data required to start up the simulation and the simulation runs cannot be used to prove properties!

5. Conclusion

In this paper we have used ITCPN's to model and analyse railway stations. The ITCPN model uses a new timing mechanism where time is associated with tokens and transitions determine a delay specified by an interval. Specifying each delay by an interval rather than a deterministic value or stochastic variable is promising, since it is possible to model uncertainty without having to bother about the delay distribution.

We have shown that the MTSRT method can been used to analyse throughput and waiting times of trains in railway stations. This MTSRT method constructs a reduced reachability graph. In such a graph a node corresponds to a set of (similar) states, instead of a single state. The reduced reachability graph can be used to *prove* certain properties or to calculate accurate bounds for all kinds of performance measures (e.g. throughput times, waiting times, occupation rates). The bounds calculated for these performance measures are always valid. Therefore, the proposed approach is extremely useful when evaluating the design of a time-critical system like a railway station.

Moreover, other Petri net based analysis techniques can be used to detect potential deadlocks of trains, etc.

We also used simulation to do a more detailed analysis of the dynamic behaviour of the railway station in Arnhem.

References

1. W.M.P. van der Aalst. Modelling and Analysis of Complex Logistic Systems. In H.J. Pels and J.C. Wortmann, editors, *Integration in Production Management Systems*, volume B-7 of *IFIP Transactions*, pages 277–292. Elsevier Science Publishers, Amsterdam, 1992.
2. W.M.P. van der Aalst. *Timed coloured Petri nets and their application to logistics.* PhD thesis, Eindhoven University of Technology, Eindhoven, 1992.
3. W.M.P. van der Aalst. Interval Timed Coloured Petri Nets and their Analysis. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 453–472. Springer-Verlag, New York, 1993.
4. W.M.P. van der Aalst and A.W. Waltmans. Modelling logistic systems with EXSPECT. In H.G. Sol and K.M. van Hee, editors, *Dynamic Modelling of Information Systems*, pages 269–288. Elsevier Science Publishers, Amsterdam, 1991.
5. ASPT. *ExSpect 4.1 User Manual.* Eindhoven, 1993.
6. B. Berthomieu and M. Diaz. Modelling and verification of time dependent systems using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, March 1991.
7. J. Bourachot. Computer-aided planning of traffic in large stations by means of the AFAIG model. *Rail International*, May:2–18, 1986.
8. G. Florin and S. Natkin. Evaluation based upon Stochastic Petri Nets of the Maximum Throughput of a Full Duplex Protocol. In C. Girault and W. Reisig, editors, *Application and theory of Petri nets: selected papers from the first and the second European workshop*, volume 52 of *Informatik Fachberichte*, pages 280–288, Berlin, 1982. Springer-Verlag, New York.
9. C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze. A unified high-level Petri net formalism for time-critical systems. *IEEE Transactions on Software Engineering*, 17(2):160–172, Feb 1991.
10. K.M. van Hee. *Information System Engineering: a Formal Approach.* Cambridge University Press, 1994.
11. K.M. van Hee, L.J. Somers, and M. Voorhoeve. Executable specifications for distributed information systems. In E.D. Falkenberg and P. Lindgreen, editors, *Proceedings of the IFIP TC 8 / WG 8.1 Working Conference on Information System Concepts: An In-depth Analysis*, pages 139–156, Namur, Belgium, 1989. Elsevier Science Publishers, Amsterdam.
12. P. Hubner, A.M. Jensen, L.O. Jepsen, and K. Jensen. Reachability trees for high level Petri nets. *Theoretical Computer Science*, 45:261–292, 1986.
13. K. Jensen. Coloured Petri Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 248–299. Springer-Verlag, New York, 1987.

14. K. Jensen. Coloured Petri Nets: A High Level Language for System Design and Analysis. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 342–416. Springer-Verlag, New York, 1990.
15. K. Jensen. *Coloured Petri Nets. Basic concepts, analysis methods and practical use.* EATCS monographs on Theoretical Computer Science. Springer-Verlag, New York, 1992.
16. M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. On Petri Nets with Stochastic Timing. In *Proceedings of the International Workshop on Timed Petri Nets*, pages 80–87, Torino, 1985. IEEE Computer Society Press.
17. M. Ajmone Marsan, G. Balbo, and G. Conte. A Class of Generalised Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, May 1984.
18. J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalised Petri Net. In C. Girault and W. Reisig, editors, *Application and theory of Petri nets : selected papers from the first and the second European workshop*, volume 52 of *Informatik Fachberichte*, pages 301–310, Berlin, 1982. Springer-Verlag, New York.
19. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
20. M.A. Odijk. ITPN analysis of ExSpect specifications with respect to production logistics. Master's thesis, Eindhoven University of Technology, Eindhoven, 1991.
21. M.A. Odijk. *A simulation tool to Evaluate the Performance of the Track Layout of Rail Junctions.* NS (internal report), 1993.
22. M.A. Odijk and W.M.P. van der Aalst. A Petri net based simulation tool to evaluate the performance of railway stations. In A. Guasch and M. Huber, editors, *Proceedings of the 1994 European Simulation Multiconference*, pages 207–211, Barcelona, June 1994. Society of Computer Simulation (SCS).
23. C. Ramchandani. *Performance Evaluation of Asynchronous Concurrent Systems by Timed Petri Nets.* PhD thesis, Massachusetts Institute of Technology, Cambridge, 1973.
24. W. Reisig. *Petri nets: an introduction.* Prentice-Hall, Englewood Cliffs, 1985.
25. J. Sifakis. Use of Petri Nets for performance evaluation. In H. Beilner and E. Gelenbe, editors, *Proceedings of the Third International Symposium IFIP W.G. 7.3., Measuring, modelling and evaluating computer systems (Bonn-Bad Godesberg, 1977)*, pages 75–93. Elsevier Science Publishers, Amsterdam, 1977.
26. A. Valmari. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets*, Bonn, June 1989.
27. W.M. Zuberek. Timed Petri Nets and Preliminary Performance Evaluation. In *Proceedings of the 7th annual Symposium on Computer Architecture*, volume 8(3) of *Quarterly Publication of ACM Special Interest Group on Computer Architecture*, pages 62–82, 1980.

Appendix

Formal Definition

In this section we define interval timed coloured Petri nets in mathematical terms, such as functions, multi-sets and relations. This definition was presented in [3].

A.1. Multi-sets

A *multi-set*, like a set, is a collection of elements over the same subset of some universe. However, unlike a set, a multi-set allows multiple occurrences of the same

element. Another word for multi-set is *bag*. Bag theory is a natural extension of set theory (Jensen [14]).

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

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

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

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

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

See Jensen [14], [15] for more details.

A.2. Definition of Interval Timed Coloured Petri Nets

The ITCPN model presented in this paper is analogous to the model described in [2]. However, in this paper we give a definition which is closer to the definition of Coloured Petri Nets (CPN), see Jensen [13], [14], [15].

Nearly all timed Petri net models use a continuous time domain, so do we.

Definition. TS is the **time set**, $TS = \{x \in \mathbb{R} \mid x \geq 0\}$, i.e. the set of all non-negative reals.

$INT = \{[y, z] \in TS \times TS \mid y \leq z\}$, represents the set of all closed intervals.

If $x \in TS$ and $[y, z] \in INT$, then $x \in [y, z]$ iff $y \leq x \leq z$.

We define an interval timed coloured Petri nets as follows:

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

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

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

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

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

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

(vi) F is the **transition function**. It is defined from T into functions. If $t \in T$, then:[3]

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

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

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

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

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

(vi) The transition function specifies each transition in the ITCPN. For a transition t, $F(t)$ specifies the relation between the multi-set of consumed tokens and the multi-set of produced tokens. The domain of $F(t)$ describes the condition on which transition t is enabled. Note that the produced tokens have a delay specified by an interval. In this paper, we require that both the multi-set of consumed tokens and the multi-set of produced tokens contain finitely many elements.

Apart from the interval timing and a transition function instead of incidence functions, this definition resembles the definition of a CP-matrix (see Jensen [13], [15]).

The formal semantics (i.e. the dynamic behaviour) of the ITCPN model are given in [2] and [3].

Notes

1. $\mathbb{N} = \{0, 1, 2, ..\}$
2. This notation has been adopted from Jensen [14].
3. $A \not\rightarrow B$ denotes the set of all partial functions from A to B.