A Proposal For Relative Time Petri Nets

Joseph Kuehn, Charles Lakos, Robert Esser
The University of Adelaide,
School of Computer Science,
SA 5005, Australia

Abstract

Petri nets are a graph-based modelling formalism which has been widely used for the formal specification and analysis of concurrent systems. A common analysis technique is that of state space exploration (or reachability analysis). Here, every possible reachable state of the system is generated and desirable properties are evaluated for each state. This approach has the great advantage of conceptual simplicity, but the great disadvantage of being susceptible to state space explosion, where the number of states is simply too large for exhaustive exploration. Many reduction techniques have been suggested to ameliorate the problem of state space explosion. In the case of timed systems, the state space is infinite, unless analysis is restricted to a bounded time period. In this paper, we present a Petri net formalism based on the notion of relative time (as opposed to the traditional approach of dealing with absolute time). The goal is to derive a finite state space for timed systems which have repeating patterns of behaviour, even though time continues to advance indefinitely.

1 Introduction

Petri nets [8] have been widely used for formally describing systems with a high level of concurrency. Having both a formal mathematical definition and an intuitive graphical notation, they are often able to capture a clear and concise representation of a system, while still providing access to a rich set of analysis techniques for exploring the dynamic behaviour of the model.

However, even for very small systems there are often (a great) many different *states* the system can reach, resulting in what is known as the *state space explosion problem* [11]. This problem is acute for concurrent models due to the increased number of possible orderings of the events (known as *interleavings*).

Moreover, even if the Petri net model is of finite size, it may give rise to an infinite sized state space, such as in Figure 1. In this case each *firing* of the transition t requires the removal of one token from the place p, but then produces two more tokens in p as well. Thus each firing of t strictly increases the number of tokens in p, and so there will be an infinite number of states corresponding to the number of tokens in p. This particular problem can be solved with a *covering* graph, but other situations are not so straightforward.

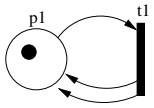

Figure 1. A simple Petri net with an infinite state space.

The addition of some form of *global clock* to a net can have a similar effect on the state space [5]. In this case, as the value of time monotonically increases, the state space continues to expand.

While the original Petri net notation did not explicitly include time as a feature of the net, a wide variety of applications rely heavily on a quantitative measure of time as part of their specification. Hence, a number of extensions to the basic Petri net notation have been proposed which explicitly relate to some form of global timing [10, 14, 16].

One popular approach in timed Petri nets is to add *time-stamp* to each token that can be used to determine when the token is accessible [15]. Typically time-stamps are based on some form of absolute global clock, which is monotonically inceasing. Since the time-stamps only ever increase, this results in an infinite state space for any non-deadlocking system. By contrast, we consider instead a delay *relative* to the time the preceding transition fired (or zero initially). This process allows us to generate a state space where any equivalences are automatically exposed, instead of requiring a separate state space reduction step.

In addition, the delays in many practical problems are neither fixed nor evenly distributed, making the expression

Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)
0-7695-2435-4/05 $20.00 © 2005 IEEE

of such delays within the net problematic at best. For this reason, we will adopt the use of *interval semantics*, similar to those introduced by [7, 12, 13]. An interval allows us to specify a range of times that may be valid for a particular task, without forcing the modeller to artificially constrain the exact distribution of those values (which may not be known).

The paper commences with foundational concepts in Section 2, including a traditional definition for Petri nets. Relative time Petri nets are then introduced in Section 3, while in Section 4 we consider properties of these nets. A comparison with other approaches, together with the conclusions and further work are found in Section 5.

2 Foundations

We begin with a few necessary formal definitions. For convenience we denote the set of natural numbers $\{0, 1, 2, \dots\}$ by $\mathbb{N}$, and the set of integers $\{\dots, -2, -1, 0, 1, 2, \dots\}$ by $\mathbb{Z}$.

2.1 Intervals of Uncertainty

Definition 2.1 *Let $\phi(\lambda, t) \in [0, 1]$ be the probability that property λ holds by time t. An* interval of uncertainty *for a property λ is a pair of timing bounds $[\boldsymbol{lo}_\lambda, \boldsymbol{hi}_\lambda]$ such that:*

1. $\boldsymbol{lo}_\lambda \leq \boldsymbol{hi}_\lambda$;
2. $\forall k < \boldsymbol{lo}_\lambda : \phi(\lambda, k) = 0$;
3. $\forall k \geq \boldsymbol{hi}_\lambda : \phi(\lambda, k) = 1$;
4. $\forall k' \geq k : \phi(\lambda, k') \geq \phi(\lambda, k)$.

Our interval thus encapsulates the lower and upper timing bounds of a property, before which it does not hold, and after which it does. The probabilities associated with the times between the lower and upper bounds remain undefined, provided they are monotonically increasing towards the upper bound. While this may not be as 'precise' as specifying the actual distribution function for each delay, it is sufficient to prove best and worst case scenarios of a model's behaviour.

2.2 Multi-sets

Definition 2.2 *A* multi-set *[2, 6], also known as a* bag*, is a mapping from a set S to the naturals $\mathbb{N}$. This may also be represented as a formal sum: $A = \sum_{s \in S} A(s)`s$. We define S_{MS} as the set of all multi-sets over S.*

A multi-set is essentially an unordered collection which allows for multiplicity. Each mapping (s, n) in a multi-set A corresponds to the element $s \in S$ appearing n times in A, written $n`s$. The usual set operations can be extended to multi-sets in a straightforward manner.

Definition 2.3 *Let $A, B, C \in S_{\mathrm{MS}}$, then:*

1. *An element $s \in S$ is* present *in A (written $s \in A$) if $A(s) > 0$.*
2. *The* size *of A (written $\sharp A$) is equal to $\sum_{s \in S} A(s)$.*
3. *A is a* subset *of B (written $A \subseteq B$) if $\forall s \in S : A(s) \leq B(s)$.*
4. *C is the* sum *of two multi-sets A and B (written $C = A + B$) if $\forall s \in S : C(s) = A(s) + B(s)$.*
5. *C is the* difference *of A and B (written $C = A - B$) if $B \subseteq A$ and $\forall s \in S : C(s) = A(s) - B(s)$.*

It is often useful to refer to a multi-set more generally as an indexed collection of individual elements from S. In particular, if we wish to compare two multi-sets over the same domain where neither is a proper subset of the other, then the formal notation can become cumbersome. For this reason we define a short hand notation for such an indexed collection of elements in a multi-set.

Definition 2.4 *Let $A \in S_{\mathrm{MS}}$, then we write A as $\{a_i\}_{i=1}^{n}$, where:*

1. $n = \sharp A$
2. $\forall 1 \leq i \leq n : a_i \in A$
3. $\forall s \in S :$ *if $sub(s) = \{m | a_m = s\}$ is the set of indices of the sequence which map to s, then $\sharp sub(s) = A(s)$.*

In other words we choose some sequence for the elements present in A such that each of the elements s appears in that sequence $A(s)$ times. If an ordering relation exists over the set S we may choose to use that relation to order the elements of our sequence, but this is not assumed for the general case.

2.3 Petri nets

We begin with the usual definitions for Petri nets.

Definition 2.5 *A* Petri net *is a tuple (P, T, A, M_0), where:*

1. *P is the set of* places *in the net; T is the set of* transitions *in the net; and $P \cap T = \emptyset$.*
2. *$A \in ((P \times T) \cup (T \times P))_{\mathrm{MS}}$ is the multi-set of* arcs *of the net.*
3. *M_0 is the* initial marking*, which is a mapping $M : P \to \mathbb{N}$.*

Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)
0-7695-2435-4/05 $20.00 © 2005 IEEE

A Petri net can be thought of as a directed graph. Its nodes are *places* and *transitions*. Places contain some quantity of *tokens*, and transitions represent the effect of various *events* on the net. The definition of *arcs* is slightly unusual, in assuming that there is one arc for each token consumed or generated. This is more appropriate when we come to associate intervals of uncertainty with the tokens consumed and generated.

A *marking* for a Petri net is a mapping which indicates how many tokens are currently within each of the places of the net. We denote the set of all possible markings as $\mathbb{M}$. Some examples of simple Petri nets are in Figures 2 and 3.

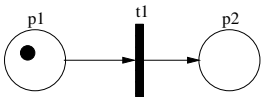

Figure 2. A Two Place, One Transition Petri net.

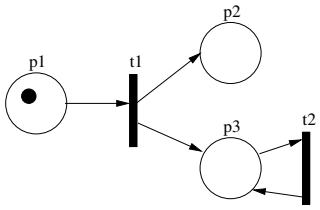

Figure 3. A Three Place, Two Transition Petri net.

Arcs are the directed edges within the net. There are edges which connect places to transitions, known as *input arcs*, and edges which connect transitions to places, known as *output arcs*. We say a transition is *enabled* when the current marking contains a token in each of its input places for each of the input arcs connecting that place to that transition. When an enabled transition is *fired*, it *consumes* one token from each place for each input arc it has from that place, and *produces* one token in each place for each output arc it has to that place. We say the resulting marking produced is *reachable* from the previous marking by firing that transition.

Definition 2.6 *Let (P, T, A, M_0) be a Petri net. The transition $t \in T$ is* enabled *for a given marking $M \in \mathbb{M}$, written $M[t\rangle$, iff $\forall p \in P : M(p) \geq A(p, t)$.*

If $t \in T$ is enabled for a given marking $M \in \mathbb{M}$ then t may fire *to produce a marking $M' \in \mathbb{M}$, written $M[t\rangle M'$, such that $\forall p \in P : M'(p) = M(p) - A(p, t) + A(t, p)$.*

The (reachable) *state space* for a Petri net is then found by recursively finding all of the new markings that may be produced from firing enabled transitions in previously reachable markings, starting from the initial marking M_0.

2.4 State Spaces

A *state space*, also called a *reachability graph*, is a data structure that represents the complete set of dynamic behaviours a system may exhibit. If a state space contains all of the dynamic behaviours that can occur for a specific model, then proving a property holds over every state in the state space is equivalent to proving that the property always holds for the system being modelled.

Definition 2.7 *A* state space *[11] of a model is a tuple (S, E, Δ, S_I) where:*

1. *S is the set of* states *of the model.*
2. *E is the set of* events *which can occur in the model.*
3. *Δ is the set of (semantic)* transitions *of the model, with $\Delta \subseteq S \times E \times S$.*
4. *$S_I \subseteq S$ are the* initial states *in which the model may begin.*

A state space embodies a directed graph with nodes given by the states S and arcs given by the transitions Δ.

Definition 2.8 *Let $s_2, s_1 \in S$. We say s_2 is* directly reachable *from s_1 if there exists an event $e \in E$ such that $(s_1, e, s_2) \in \Delta$. This may be written as either $s_1 \rightarrow_e s_2$ or simply $s_1 \rightarrow s_2$.*

Let $s_{dest}, s_{src} \in S$. We say s_{dest} is reachable *from s_{src} if there exists a sequence of events $\{e_i\}_{i=1}^{n-1}$ and a sequence of states $\{s_i\}_{i=1}^{n}$ such that:*

1. *$s_{src} = s_1$ and $s_{dest} = s_n$.*
2. *$\forall 1 \leq i \leq n-1 : (s_i, e_i, s_{i+1}) \in \Delta$.*

This may be written as $s_{src} \rightarrow^* s_{dest}$.

We are normally concerned with proving that a property holds for all reachable states of our model.

Definition 2.9 *The state space for a Petri net (P, T, A, M_0) is the tuple (S, E, Δ, S_I), where:*

1. *$S = \mathbb{M}$.*
2. *$E = T$.*
3. *$(M_1, t, M_2) \in \Delta$ iff $M_1[t\rangle M_2$.*
4. *$S_I = \{M_0\}$.*

Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)
0-7695-2435-4/05 $20.00 © 2005 IEEE

The set of possible states that a Petri net may assume is the set of reachable markings for that net, with the initial marking M_0 being the only initial state in the state space. Then, there is an edge from the state M_1 to the state M_2 with label t iff t is enabled in M_1, and the resulting marking produced by firing t in M_1 is exactly M_2.

The state spaces for the models in Figures 2 and 3 are shown in Figures 4 and 5 respectively. The initial state is designated by an arrow without a source.

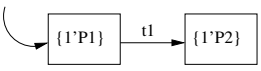

Figure 4. The state space for Figure 2

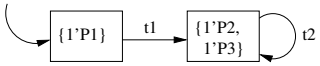

Figure 5. The state space for Figure 3

3 Relative Time Petri Nets

3.1 Structure of Relative Time Petri-nets

We now propose our approach to adding time to Petri nets, which we will call *relative time Petri nets* (RTPNs). As mentioned previously we also add a time-stamp to each token for denoting its accessibility, however our time-stamp values are a delay *relative* to the time the preceding transition fired. By eliminating the requirement that the time-stamps be monotonically increasing, we are thus able to represent the state spaces of cyclical timed systems in a finite form.

Definition 3.1 *A* relative time Petri net *is a tuple* (P, T, A, M_0) *where:*

1. P *is the set of* places *in the net;* T *is the set of* transitions *in the net; and* $P \cap T = \emptyset$.
2. $A \in (((P \times T) \cup (T \times P)) \times (\mathbb{N} \times \mathbb{N}))_{\mathrm{MS}}$ *is the multi-set of* arcs *of the net.*
3. M_0 *is the* initial relative time marking*; which is a mapping* $M : P \to \mathbb{Z}_{\mathrm{MS}}$.

Some examples of relative time Petri nets are shown in Figures 6, 7, and 8. We are thus extending the previous definition of Petri-nets in three different ways.

Firstly, we extend the markings so that each token is marked with a time-stamp $d \in \mathbb{Z}$. This value represents the amount of time that the token has been accessible since the last transition firing. So tokens with a negative valued time-stamp are not accessible yet. This delay is expressed *relative* to the time at which the last transition fired (transitions still fire instantaneously). We denote the set of all relative time markings as $\mathbb{M}^*$.

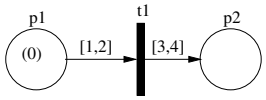

Figure 6. A Two Place, One Transition RTPN.

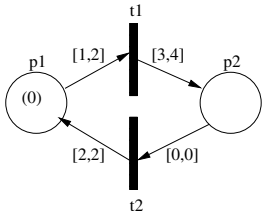

Figure 7. A Two Place, Two Transition RTPN.

Secondly, we annotate each arc in the net with an *interval of uncertainty*. This interval represents the range of possible delays each input and output of a transition may have in the model. Input arc delays represent how long the required input tokens of a transition must previously have been accessible before that transition can be enabled [7]. Output arc delays represent how long it will be until the produced tokens of that transition will first be accessible [12]. We provide both kinds of arc delays for the sake of flexibility and symmetry.

Lastly, we modify the transition binding and firing rules to take into account this extra information in our model. However, it will be helpful to define a few additional relations between relative time markings first.

Definition 3.2 *Let* $M_1, M_2 \in \mathbb{M}^*$. *We say* M_1 *is a* timed subset *of* M_2*, written* $M_1 \ll M_2$*, iff* $\forall p \in P : M_1(p) = \{x_i\}_{i=1}^n$*, and* $\exists \{y_i\}_{i=1}^n \subseteq M_2(p)$ *such that* $x_i \leq y_i \; \forall \; i$.

If a marking M_1 is a timed subset of a marking M_2, it means that M_2 contains at least as many tokens as M_1, and those tokens have been available for at least as long as the

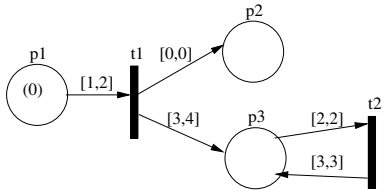

Figure 8. A Three Place, Two Transition RTPN.

Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)
0-7695-2435-4/05 $20.00 © 2005 IEEE

tokens in M_1. In practical terms this means any transition enabled in M_1 will also be enabled in M_2, since M_2 has at least as many tokens as old as M_1 does.

Notice that for any timed multi-set M it is clear that $\emptyset \ll M$, and $M \ll M$.

Definition 3.3 *Let $M_3, M_2, M_1 \in \mathbb{M}^*$, and $M_1 \ll M_3$. Then we say M_2 is a* timed difference *between M_3 and M_1, written $M_2 = M_3 \backslash M_1$, iff $M_2 \subseteq M_3$, $\sharp M_2 = \sharp M_3 - \sharp M_1$ and $M_1 \ll (M_3 - M_2)$.*

Let $M_3, M_2, M_1 \in \mathbb{M}^*$, and $M_1 \ll M_3$. Then we say M_2 is the minimal timed difference *between M_3 and M_1, written $M_2 = M_3 \backslash_{min} M_1$, iff $M_2 = M_3 \backslash M_1$, and $\forall M_4 \in \mathbb{M}^*$ st $M_4 = M_3 \backslash M_1$, then $M_4 \ll M_2$.*

Notice that there may be multiple valid values for the timed difference between two markings. For example if $A = \{-3, 1\}$ and $B = \{0, 1, 2\}$ then the multi-sets $\{0\}$, $\{1\}$ and $\{2\}$ are all valid timed differences between A and B. However, the minimal timed difference between A and B is $\{2\}$, since $\{0\} \ll \{1\} \ll \{2\}$. In practice we can easily calculate the minimal difference between any two finite markings $A \ll B$ by iteratively removing a pair of corresponding elements $a_i \leq b_i$ from each multi-set. That is we calculate the (unique) sequences $\{A_i\}_{i=0}^n$ and $\{B_i\}_{i=0}^n$ such that $A_0 = A$, $A_n = \emptyset$, $B_0 = B$, $A_i = A_{i-1} - \{max(A_{i-1})\}$ and $B_i = B_{i-1} - \{min\{b \in B_i | b \geq max(A_{i-1})\}\}$. In this case the resulting set B_n is the minimal timed difference.

The purpose of the timed difference function is to allow us to make a clear distinction between the possible transitions defined in the net, and the changes that occur to the marking when those transitions are fired. The $\ll$ operator determines when a transition is enabled, and the $\backslash$ operator calculates which tokens to remove as a consequence of that firing (a non-trivial questions in timed nets since two tokens in the same place are no longer indistinguishable).

The minimal timed difference will become important later on, when we consider the semantics of relative time Petri nets. In general, $M_3 \backslash_{min} M_1$ can be thought of as removing the subset of M_3 which is 'closest in time' to M_1, leaving behind the oldest tokens possible.

Definition 3.4 *Let $M, M' \in \mathbb{M}^*$. Then M' is equal to M* aged by δ, *written $M' = \mathbf{age}(M, \delta)$, iff $\forall p \in P, d \in \mathbb{Z}$: $M'(p, d + \delta) = M(p, d)$.*

Since we wish to retain the ability for transitions to fire instantaneously, we will need some way to designate the passage of time in the model. We do this by 'aging' the marking of the net prior to the next transition becoming enabled, and then firing. It will become useful later on to notice that $\forall M \in \mathbb{M}^*, \epsilon \geq 0 : M \ll \mathbf{age}(M, \epsilon)$.

For example, let us consider a marking which has three tokens with delays of -7, -5 and -3 respectively. Say we need two available tokens for our next transition to be enabled (assuming for simplicity that the associated input arcs have intervals $[0, 0]$), so we age the marking by 5 time units. Hence, in the new marking we will have tokens with delays of -2, 0 and 2 respectively. The first token has a negative delay in the aged marking because it is still unavailable for two more units of time, while the last token has a time-stamp of two because it was only unavailable for the first three time units. If our transition only needed to consume a single token, then either of the 0 or the 2 time-stamped tokens could be chosen, since they are both available in the current marking.

We recall that our input and output arcs have each been annotated with an interval of uncertainty, which describes a range of times at which a transition can fire and a range of different delays the produced tokens may have. To allow us to identify each of the different possible cases we formally define an *event*.

Definition 3.5 *An* event *is a tuple (t, M_{cons}, M_{prod}), where:*

1. *$t \in T$; $M_{cons}, M_{prod} \in \mathbb{M}^*$ are (partial) relative time markings.*
2. *$\forall p \in P$: let $X = \{(p, t, [a_i, b_i])\}_{i=1}^n \subseteq A$ be the complete multi-set of input arcs for t, then $M_{cons}(p) = \{x_i\}_{i=1}^n$ where $a_i \leq x_i \leq b_i \; \forall i$.*
3. *$\forall p \in P$: let $X = \{(t, p, [a_i, b_i])\}_{i=1}^n \subseteq A$ be the complete multi-set of output arcs for t, then $M_{prod}(p) = \{x_i\}_{i=1}^n$ where $-a_i \geq x_i \geq -b_i \; \forall i$.*

Each event represents a single selection from each of the input and output intervals that are defined for a given transition. The set of all events for a particular transition thus represent the complete set of occurrences of that transition in the system being modelled.

This additional step is important, because it is ambiguous simply to talk about a transition t being enabled for a relative time marking M. It is not clear whether that means there are *some* combinations of input delays that are satisfied by M, or whether *every* combination of the input delays are satisfied by M. To avoid this ambiguity we will treat each case as a separate event, so that we can clearly distinguish which of the possible occurrences of t are enabled for a particular marking.

For example, in Figure 6 the transition $t1$ has an input delay of $[1, 2]$ and an output delay of $[3, 4]$. Thus there are four possible events which correspond to $t1$, which are (informally) $(t1, \{1\}, \{-3\})$, $(t1, \{1\}, \{-4\})$, $(t1, \{2\}, \{-3\})$ and $(t1, \{2\}, \{-4\})$; each of which represents a valid occurrence of $t1$ according to our model.

We can now define our enabling and firing rules.

Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)
0-7695-2435-4/05 $20.00 © 2005 IEEE

Definition 3.6 *An event $E = (t, M_{cons}, M_{prod})$ is* enabled *for a marking $M \in \mathbb{M}^*$ after delay δ, written $M[\delta, E\rangle$, iff $M_{cons} \ll \mathbf{age}(M, \delta)$.*

If the event (t, M_{cons}, M_{prod}) is enabled for the marking $M \in \mathbb{M}^*$ after a delay of δ then the marking $M' \in \mathbb{M}^*$ produced by firing *that event, written $M[\delta, E\rangle M'$, is: $M' = \mathbf{age}(M, \delta)\backslash_{min} M_{cons} + M_{prod}$.*

The requirement for enabling says that after the marking has been aged by δ the input requirements for t are satisfied. The firing rule states that the specific tokens that we consume must be chosen as those which are the minimal timed difference to M_{cons}. In practice they will be those applicable tokens which have been available for the shortest amount of time, that is, a LIFO-like behaviour. This is an important requirement for maintaining the *diamond rule* [4] for our state space. Without this requirement the choice of ordering for a particular interleaving might result in different behaviours from the net.

In the case of concurrently enabled transitions (including auto-concurrency), after the initial choice of delay for aging the net there may then be many additional events that occur immediately (ie. with zero delays). As with most timed systems, exploration may be performed with an initial aging of the current marking followed by exhausting all of the possible events that can occur from the resulting marking.

We note that since this is a relative time marking, it is the tokens which do *not* participate in the transition that age rather than the ones which are produced. This is the reason why our token time-stamps may go above zero, as those tokens have now been available for some time *before* the firing of the preceding transition.

We can construct a state space for a particular relative time Petri net in a similar way as we do for normal Petri nets. Beginning with the initial marking M_0 we find the resulting marking produced by firing each enabled transition, and then repeat the process of firing transitions until we have explored all of the markings reachable from M_0.

3.2 Behaviour of Relative Time Petri Nets

We now provide the formal definitions for constructing a state space for relative time Petri nets, using the same approach as we did for Section 2.4.

Definition 3.7 *The state space for a relative time Petri net (P, T, A, M_0) is the tuple (S, E, Δ, S_I), where:*

1. $S = \mathbb{M}^*$.
2. $E = \mathbb{N} \times T$.
3. $(M_1, (\delta, t), M_2) \in \Delta$ *iff* $\exists M_{cons}, M_{prod} \in \mathbb{M}^* : M_1[\delta, (t, M_{cons}, M_{prod})\rangle M_2$.
4. $S_I = \{M_0\}$.

So, we extend the previous definition for an untimed Petri net state space by including the amount of time the previous marking was aged in order to enable (t, M_{cons}, M_{prod}). Our reachable state space is now those states which can be reached by a sequence of transitions, and a sequence of delays after which there are corresponding events for that transition enabled in the current marking.

Note that we do not include the particular choices for M_{cons} and M_{prod} as labels on the edges in the figures. This abstraction allows us to combine the arcs corresponding to different events for the same transition and delay, provided they produce the same marking when fired. We note that once we know the current marking, the choice of delay used to age the current marking and the destination marking reached, it is trivial to compute which set of events for t all produce that destination marking when fired.

Figures 9, 10, and 11 provide (partial) state spaces for the previous relative time Petri nets in Figures 6, 7 and 8 respectively. We have also removed some redundant edges from the diagram, as discussed below. Notice that in these examples there is a corresponding outgoing arc labelled with a delay for each choice from the input arc intervals. In general, transitions which consume tokens from multiple places will not however have an exponential set of outgoing arcs, as it will tend to be the token which becomes available last that will dominate the enabling of that transition.

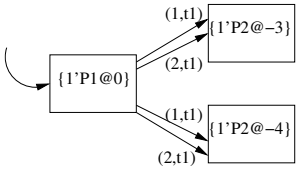

Figure 9. The state space for Figure 6

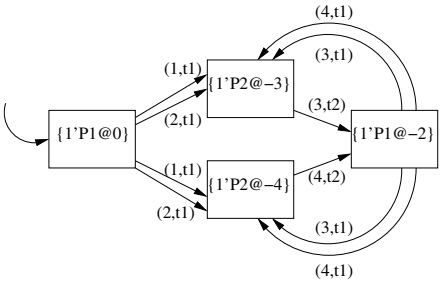

Figure 10. The state space for Figure 7

We note that our original goal was to provide an extension to the Petri net formalism which allowed for the expression of quantitative timing delays without automatically resulting in an infinitely large state space. However, in the

Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)
0-7695-2435-4/05 $20.00 © 2005 IEEE

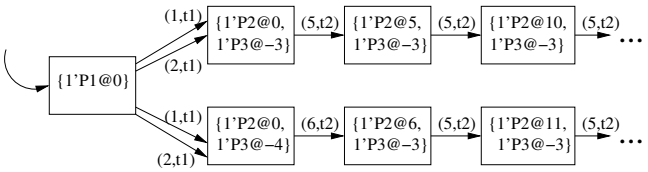

Figure 11. The partial state space for Figure 8

state space of Figure 11 we see that this is not the case. This is because one of the tokens in the model will never be consumed by a transition. If the token is never consumed then the inevitable consequence of aging the token is that its time-stamp will continue to increase monotonically without limit.

Also, though it is common in many other formalisms to do so, we have not yet introduced any requirement that the delay be chosen so that it is minimal. In other words, if a particular transition t has some events enabled for a marking M after a delay δ, then by our definition it will also have some events enabled for the marking M after a delay of $\delta + \epsilon \; \forall \; \epsilon \geq 0$. Hence we could conceivably have an infinite number of edges for each transition with labels $(k, t) \; \forall \; k \geq \delta$. For brevity however, this possibility has been omitted from the previous figures.

In the next section we provide techniques for eliminating these problems whilst still retaining a complete set of the dynamic behaviours of the model. As part of this process we provide definitions for the 'earliest' choice of delay, although we note that the 'truncation' technique is alone sufficient to limit the number of states in the state space to a finite number.

4 Properties of Relative Time Petri Nets

4.1 Equivalence Reduction by Truncation

We now consider what happens when a token has been available for binding long enough that it satisfies the complete range of all of the guards on the available input arcs. In particular, we will prove that aging a token beyond a certain delay does not change which events are enabled for a given state. This allows us to stop tracking each token's delay past a particular upper bound, without losing any of the behaviours of the model.

Definition 4.1 *A* time cap *for a relative time Petri net (P, T, A, M_0), is a mapping $\omega : P \rightarrow \mathbb{N}$ such that $\forall \; p \in P : \omega(p) = max\{b \mid \exists \; a \in \mathbb{N}, t \in T : [a, b] \in A(p, t)\}$.*

The time cap for a net defines an upper bound for each of the places on that net. In particular, the value of this upper bound is greater than or equal to any of the values in an input arc interval leading from that place. So if a token in place p has been available for at least $\omega(p)$ units of time, it can participate in firing any event related to that place. We notice also that the time cap ω is in fact a structural property of the model as it makes no reference to any particular marking of the net.

Definition 4.2 *Let $M \in \mathbb{M}^*$ be a marking for a net (P, T, A, M_0), with time cap ω. We define $M|_\omega \in \mathbb{M}^*$ as the* truncation *of M, iff $\forall p \in P, d \in \mathbb{Z}$:*

$$M|_\omega(p, d) = \begin{cases} M(p, d) & d < \omega(p) \\ \sum_{i=\omega(p)}^{\infty} M(p, i) & d = \omega(p) \\ 0 & d > \omega(p) \end{cases}$$

That is, each time-stamp in the truncation of M is 'capped' once it reaches $\omega(p)$. This is the reason why the number of tokens with time-stamp greater than $\omega(p)$ is zero, as they are all counted towards the tokens with time-stamp exactly $\omega(p)$ instead. We note that $M|_\omega \ll M$, as the values in the truncation will either be equal to (uncapped) or less than (capped) those in M.

Theorem 4.3 *Let $M \in \mathbb{M}^*$ be a marking for the net (P, T, A, M_0). Then every event enabled for M is enabled for $M|_\omega$, and vice versa.*

PROOF:

$\Leftarrow$ Let (t, M_{cons}, M_{prod}) be enabled for $M|_\omega$ after delay δ. Since $M|_\omega \ll M$ and $M_{cons} \ll \mathbf{age}(M|_\omega, \delta) \ll \mathbf{age}(M, \delta)$, then $M_{cons} \ll \mathbf{age}(M, \delta)$ since the time-stamps in M are greater than those in $M|_\omega$. Hence, (t, M_{cons}, M_{prod}) is enabled in M after delay δ.

$\Rightarrow$ Let (t, M_{cons}, M_{prod}) be enabled for M after delay δ. Then $M_{cons} \ll \mathbf{age}(M, \delta) \Rightarrow M_{cons}|_\omega \ll \mathbf{age}(M|_\omega, \delta)$. But $M_{cons} = M_{cons}|_\omega$, as the tokens in M_{cons} must be within the input intervals for t. Therefore $M_{cons} \ll \mathbf{age}(M|_\omega, \delta)$, so (t, M_{cons}, M_{prod}) is enabled in $M|_\omega$ after delay δ.

□

Given that the events enabled at M are exactly the same as those enabled at $M|_\omega$, the state space formed by truncating all markings will be an equivalence-reduced version of the non-truncated state space. In this way, we can achieve a finite state space even though some token time-stamps would otherwise increase without limit.

We also observe that there is a *minimum* possible time-stamp that a token in a marking will ever attain. If we define $\Omega(p)$ as the maximum upper bound on the output arcs leading into p, then every token in p will have a time-stamp greater than or equal to $-\Omega(p)$. This is because the aging function of our relative time net always increases the token's

Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)
0-7695-2435-4/05 $20.00 © 2005 IEEE

time-stamp. Thus, we only need to track the time-stamp of each token within the range of $[-\Omega(p), \omega(p)]$, for each place p, to retain a complete set of behaviours of the net.

Definition 4.4 *If the event* (t, M_{cons}, M_{prod}) *is enabled for the marking* $M \in \mathbb{M}^*$ *after a delay of* δ*, then the* truncated *marking* $M' \in \mathbb{M}^*$ *produced by* firing *that event is:* $M' = \mathbf{age}(M, \delta)|_\omega \backslash_{min} M_{cons} + M_{prod}$.

We can now revisit one of our previous examples in Figure 8 which produced an infinite state space in Figure 11 under the original 'unbounded' semantics. Using truncation we can instead generate a state space with a finite number of states, shown in Figure 12, which still retains all of the behaviours of the original model. We note that there may still be an infinite number of edges with strictly increasing delays specified, but we may abstract them out of the figure since they no longer refer to distinct states.

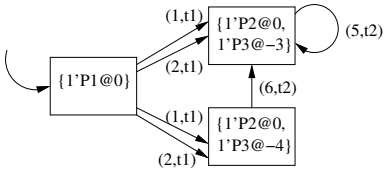

Figure 12. The truncated state space for Figure 8

Clearly a relative time Petri net with all arc intervals $[0, 0]$ will have an isomorphic state space under truncation to an untimed Petri net, as all of the markings will simply map onto a 0 time-stamp.

Henceforth we assume that we are always referring to the truncated markings of a relative time net.

4.2 Eager Transition Semantics

Previously in this paper we have not attempted to constrain events to firing at the earliest opportunity, ie. for minimal δ. We now provide definitions for limiting the state space under *eager* transition semantics. That is, we only explore the dynamic behaviours available in each state up to the earliest time at which we can guarantee an event will occur.

Definition 4.5 *An event* (t, M_{cons}, M_{prod}) *is* certain *iff:* $\forall p \in P$ *: let* $\{(p, t, [a_i, b_i])\}_{i=1}^n \subseteq A$ *be the complete multiset of input arcs for t, then* $M_{cons}(p) = \{b_i\}_{i=1}^n$.

The earliest certain events *for a marking* $M \in \mathbb{M}^*$*, is the set of certain events which are enabled in* M *for a minimal delay* δ_0*. That is, for any other certain event enabled in* M *after a delay* δ_1*, then* $\delta_1 \geq \delta_0$.

The earliest certain event time *for a marking* M*, is the delay* δ *after which its* earliest certain events *are enabled.*

That is, a 'certain' event is one which satisfies the upper bound of all of its input intervals, so that there is no longer any uncertainty over whether this transition may be fired once this event becomes enabled. Similarly, the set of earliest certain events are those which can be enabled with the smallest possible delay from the present marking, and so can be used as the 'upper limit' on the delay for a particular state when considering eager transition semantics.

4.3 Absolute Time

We now describe a general procedure for generating the corresponding absolute time state space from a relative time state space.

Definition 4.6 *Let* (S, E, Δ, S_I) *be a state space. A* trace *is a sequence of (strictly) alternating states and events, written* $[s_0, e_1, s_1, e_2, s_2, \dots]$*, where* $\forall i : (s_i, e_{i+1}, s_{i+1}) \in \Delta$.

We define the set of all relative time traces as $\mathbb{T}_R$.

In the case of relative time Petri nets, we know that each of the members of E in a trace also contains a delay. This is the key to building absolute state spaces from our relative states.

Definition 4.7 *The* aggregate delay function *is a mapping* $\mathbf{clock} : \mathbb{T}_R \times \mathbb{N} \rightarrow \mathbb{N}$*, where* $\forall \xi = [s_0, (\delta_1, t_1), s_1, \dots], n \leq |\xi|/2 : \mathbf{clock}(\xi, n) = \sum_{i=1}^n \delta_i$

Recall that the annotation of a delay on an edge in a relative time state space represents the amount of time that the marking was aged from the previous marking, before the firing of the specified transition. Hence, the sum of all of the delays that have occurred prior to a particular state in a trace represents the total 'global clock' time that has passed since the initial marking. We can now simply combine the clock time with the relative value on each of the tokens in our current marking to immediately obtain an equivalent absolute marking.

Definition 4.8 *Let* $\xi^* = [s_0, (\delta_1, t_1), s_1, \dots]$ *be a relative time trace. Then the corresponding* absolute time *trace is defined as:*

$\mathbf{absolute}(\xi^*) =$
$[s_0, (\mathbf{clock}(\xi^*, 1), t_1),$
$\mathbf{age}(s_1, \mathbf{clock}(\xi^*, 1)), (\mathbf{clock}(\xi^*, 2), t_2),$
$\mathbf{age}(s_2, \mathbf{clock}(\xi^*, 2)), \dots]$

We can then construct an absolute time state space for our relative time Petri net by combining all of the absolute traces generated. This demonstrates that no information is lost when using a relative time representation. See Figure 13 for an example, which is based on the relative time Petri net from Figure 7 which had a relative state space represented in Figure 10.

Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)
0-7695-2435-4/05 $20.00 © 2005 IEEE

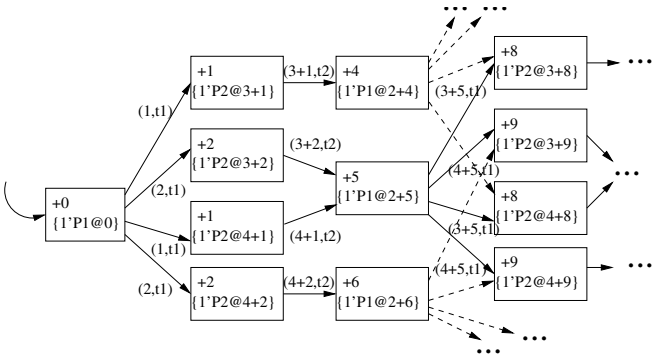

Figure 13. A portion of the absolute state space for Figure 7

5 Discussion and Conclusions

In this paper, we have proposed an approach to the definition of timed Petri nets using relative time instead of absolute time. Tokens in a marking have an associated time-stamp which indicates the time at which the token becomes accessible *relative* to the time that the previous transition occurred. The use of relative time means that tokens which are not consumed by transitions may have a time-stamp which becomes monotonically increasingly. We have defined a *time cap* beyond which there is no point in increasing the time-stamps. The recognition of the time cap allows us to produce an equivalence-reduced state space for our relative time Petri nets. In this way, we are able to capture repeated patterns of behaviour in a timed system as a finite state space. We have also shown that it is possible to recover an absolute time state space from the relative time version.

5.1 Related Work

We note a great deal of similarity between our approach and *interval timed coloured Petri nets* (ITCPN) [13]. This is predominately due to our decision to time-stamp the tokens within each marking and also to adopt interval semantics to represent abstract delay distributions. However, despite their similar structure, ITCPN events play a slightly different role than ours do. Without input delays it is much easier to maintain well-formedness under interleaving, so that the 'input requirement' specified by ITCPN events is actually the binding of tokens to be consumed. However, we choose to limit our events to represent a strictly contained instance of each of the involved intervals, and instead calculate the appropriate binding based on the current marking and the minimal difference function.

We note that there is little difference between the two approaches in terms of the overall complexity of the enabling condition and firing rules (when considered together). For example, we apply our aging function to the tokens not consumed, while ITCPNs must apply their scaling function to the produced tokens of the transition which fired.

Other work that is also based on interval semantics includes [7], as explored by [1]. In particular, their approach also uses relative time semantics instead of a global clock. However, we have extended our marking with the current delay information for each state, while they create an auxiliary structure to contain this information instead. They call this a *firing interval set*.

A firing interval set expresses a collection of delays for each of the transitions enabled in the current marking, which are in essence the labels for each of the possible outgoing transitions from the current state. When they fire an enabled transition, say $t1$, they modify the current firing interval set in three ways. Firstly they remove an entry for $t1$, and each of the entries corresponding to transitions which were in conflict with $t1$, as the input tokens they will have needed were consumed by $t1$ instead. They then decrement each of the delays for the remaining transition entries, to signify the passage of time. Lastly, they add new entries for each new enabling that was not possible in the previous marking, or was disabled by the consumption of tokens by $t1$ and then re-enabled by the production of tokens by $t1$.

Our firing rule is similar to this process, as we age the current marking, remove the consumed tokens and then add the produced tokens with their selected delays. We then calculate for the new state which events are enabled for the new marking after certain delays, but this is in principle no more work than in [1], as they must also recalculate all of the enablings in order to update the firing interval set. We say 'in principle' because we have chosen to assign delays to the arcs of the net, allowing a larger collection of possible behaviours to potentially be expressed for each transition in a relative time Petri net.

We also note that despite requiring delays before the firing of transitions, they have a different interpretation of the interleaving problem discussed in Section 3.1. Because their approach does not differentiate between the *times* at which multiple tokens arrive in the same place, they cannot talk about which tokens arrived *earliest* to a particular place. They must say instead that there has been *at least* one token in that place since the earliest arrival. Provided a new token is produced at that place moments before a token is consumed, the contribution that place makes to each of the other binding delays in the firing interval set remains unchanged. In our model we differentiate between the specific times at which the tokens arrive in each place. Hence if firing one transition consumes the 'older' token in the place, in the new state previously conflicting transitions may have their delays increased while they wait for 'younger' tokens instead.

Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)
0-7695-2435-4/05 $20.00 © 2005 IEEE

Other work on incorporating relative time into reachability analysis has been carried out particularly in the area of asynchronus circuit design [3, 9]. The construction of *lazy transition systems* is used during synthesis to prune a previously explored untimed state space of those areas which cannot be reached due to the timing constraints of the system. In this case the goal is to optimise the construction process by exploiting any inherent timing characteristics which may, in practice, simplify the actual operation of the system. In this case the amount of timing information retained for the system depends upon the discovery of useful causal relations between the system's components, rather then a permanent part of the initial specification of the system.

5.2 Future Work

The next step in this research is to gain practical experience in the application of our proposals to non-trivial case studies. Another high priority is to identify the kind of system properties that can be specified for relative time Petri nets, and then to implement analysis algorithms that can evaluate these properties on the finite relative time state spaces that we have generated.

We also plan to consider the extension of this work from integral time to other granularities, even to real-valued time. In this regard, we anticipate adopting the notion of *state classes* along the lines considered in [13].

References

[1] B. Berthomieu and M. Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. Software Engineering.*, 17(3):259–273, 1991.

[2] W. D. Blizard. Multiset Theory. *Notre Dame Journal of Formal Logic*, 30(1):36–66, 1989.

[3] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Taubin, and A. Yakovlev. Lazy transition systems: application to timing optimization of asynchronous circuits. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 324–331, New York, NY, USA, 1998. ACM Press.

[4] H. J. Hoogeboom and G. Rozenberg. Diamond Properties of State Spaces of Elementary Net Systems. *Fundamenta Informaticae*, XIV:287–300, 1991.

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

[6] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art Of Computer Programming*. Addison-Wesley, 1969.

[7] P. Merlin and D. Farber. Recoverability of Communication Protocols. *IEEE Trans. Comm.*, 24(4):1036–1043, 1976.

[8] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[9] M. A. Peña, J. Cortadella, E. Pastor, and A. Kondratyev. Formal verification of safety properties in timed circuits. In *ASYNC*, pages 2–11, 2000.

[10] J. Sifakis. Use of Petri nets for Performance Evaluation. *3rd Intl. Symposium on Modeling and Evaluation, IFIP*, pages 75–93, 1977.

[11] A. Valmari. The State Explosion Problem. *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, 1491:429–528, 1998.

[12] W. 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, pages 453–472. Springer-Verlag, Berlin, 1993.

[13] W. M. P. van der Aalst. Timed Coloured Petri nets and their Application to Logistics. *PhD-Thesis, Eindhoven University of Technology*, 1992.

[14] W. M. P. van der Aalst, K. M. van Hee, and H. A. Reijers. Analysis of Discrete-Time Stochastic Petri Nets. *Statistica Neerlandica*, 54(2):237–255, 2000.

[15] K. M. van Hee, L. J. Somers, and M. Voorhoeve. Executable Specifications for Distributed Information Systems. In W. . Working Conference on Information System Concepts: An In-depth Analysis, Namur, Belgium, E. D. Falkenberg, and P. Lindgreen, editors, *Proceedings of the IFIP TC 8, Elsevier Science Publishers, Amsterdam*, pages 139–156, 1989.

[16] W. M. Zuberek. Performance Evaluation of Concurrent Systems Using Timed Petri Nets. *13th Annual ACM Computer Science Conference*, pages 326–329, March 1985.

Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)
0-7695-2435-4/05 $20.00 © 2005 IEEE