Dynamic Process Creation in High-Level Petri Nets

by

K.M. van Hee    P.M.P. Rambags

90/2

January, 1990
This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology. Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review. Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
All rights reserved
Editors: prof.dr.M.Rem
        prof.dr.K.M. van Hee
Dynamic Process Creation
in
High-Level Petri Nets

K.M. van Hee    P.M.P. Rambags

Department of Mathematics and Computing Science
Eindhoven University of Technology

Abstract

All existing Petri Net models have a static topology, i.e. the number of components and their interaction structure are fixed during the course of the modeled system. In this article, we investigate to what extent presently known high-level Petri Nets support the modeling of processes with a dynamic topology, i.e. a changing interaction structure and number of components. We finally present the DTN model, a high-level Petri Net model with time component, where existing processes can create new ones.

Topics

Dynamic process creation, relationships between net theory and other approaches, high-level net models, timed nets.

1 Introduction

In literature, many kinds of Petri Nets have been described. The token selection mechanism of the first ones has been based on the availability of tokens only [eg. 23,28]. Next, Genrich [5] and Jensen [16] introduced tokens with a value and their selection depended no longer only on their availability, but also on their values. We have added a time component to each token in order to model real-time aspects [7], which resulted in concepts as action time, event time and transition time. Transitions are allowed to fire only if their action times are as low as possible. An individual transition can no longer independently (locally) determine whether it may fire (as in former models), the network of transitions and places should permit only transitions with lowest action time to fire.
All Petri Net models have a common property: They have a static topology, i.e. the number of components and their interaction structure are determined in advance. On the other hand, some models of discrete systems have a dynamic topology, i.e. new components can be created during the course of a system and the interaction structure may change. A well-known example is the Actor model [1,8,15].

Our main interest is in modeling distributed systems and therefore we are looking for formalisms with a high expressive power, i.e. formalisms that allow concise descriptions. In this article, we investigate to which extent dynamic process creation can be modeled with an existing high-level Petri Net model and what additional features facilitate the modeling of processes with a dynamic topology.

The article has been organized as follows: In Section 2 we review some high-level Petri Net models with a static topology. Section 3 has a short, informal description of the Actor model (for a formal description, see [8]) and we sketch a simple construction of a realization of the Actor system in a high-level Petri Net. Section 4 is the core of this article. Here we show, by construction, that it is possible to describe dynamic topology processes within a standard high-level Petri Net model. This construction, however, is not very elegant. Therefore we suggest some modifications on high-level nets to incorporate dynamic process creation and we elaborate our ideas into a formal definition of the so-called Dynamic Time Net (DTN) model. Figure 1.1 gives the evolution of this model.

We conclude in Section 5.

<table>
<thead>
<tr>
<th>Kind of Petri Net</th>
<th>Token selection criteria</th>
<th>At transition level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elementary Net</td>
<td>Availability</td>
<td></td>
</tr>
<tr>
<td>Predicate/Transition Net</td>
<td>+ value</td>
<td></td>
</tr>
<tr>
<td>Coloured Net Des</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real-Time Des</td>
<td>+ time stamp</td>
<td></td>
</tr>
<tr>
<td>Dynamic Time Net</td>
<td>+ version number [4]</td>
<td></td>
</tr>
<tr>
<td>(DTN future)</td>
<td>+ stochastic component</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(some tokens are more likely to be selected than others)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>At network level</td>
</tr>
</tbody>
</table>

Figure 1.1 Evolution of the DTN model

2 Static topology

All existing Petri Net models have a static topology: The number of components and their interaction structure are fixed during the system's course. We intend to explore the limits of such a static structure with respect to the modeling of processes with a dynamic topology. For this purpose, we use a high-level Petri Net model as basis. The two best-known ones are Predicate/Transition Nets [5] and Coloured Nets [16]. We
have chosen for another one, namely the Des model [7,9], which is a limited version of Coloured Nets. The most important motivations behind this choice are:

- The Des model has a strict separation of token consumption and token evaluation. The network takes care for the selection of tokens to be consumed by transitions without considering their values. Hence, token consumption proceeds as in the classical Place/Transition Nets [23].

- The transitions in the Des model behave like operators in the mathematical sense: Given their input parameters, they compute their output parameters by means of a mathematical function. Output parameters are of the bagtype (cf. notations, see subsection 4.2) and they are interpreted by the network as new tokens. As a consequence, transitions can easily be specified with formalisms like Z [6], VDM [18] or functional languages [13], and the computations of state transitions of a Des are easier to perform.

- It is important to have a tool that can check and simulate modeled nets. Huber, Jensen and Shapiro [14] have developed such a tool for Coloured Nets. It is, however, very difficult to build a tool that can handle all arc inscriptions in an efficient way and that is why their tool does not support all possible Coloured Nets. For instance, the net from Figure 2.1 is not supported by the tool. On the other hand, we have developed a tool for Des'ses that can efficiently handle all allowed situations. That is a direct consequence of the separation of token consumption and evaluation.

![Figure 2.1 A Coloured Net](image)

- The semantics of Petri Nets is usually given in terms of firing sequences or Occurrence Nets [24]. An Occurrence Net itself is again a Petri Net. One of our primary goals is to compare Petri Nets with totally different formalisms, therefore we need a more general concept. The semantics of the Des model is in terms of transition systems (see below).

A diversity of formalisms has been represented in terms of transition systems, for example temporal logic [20], models with a dynamic topology [1,8,15], trace theory based systems [19,21], process algebras [11,12,22], etc. This enables us to formally compare Des'ses with systems described in one of those formalisms. In Section 3, we compare Des'ses with Actor systems.

Please note that there are no significant differences between the modeling power of Coloured Nets and Des'ses. Each system modeled as a Coloured Net can be easily
transformed into a Des of about the same description length.

A transition system consists of a finite or countable set of states, some of which are called initial. The system starts in an initial state and then moves from one state to another. Actually, a transition system is a directed graph. Most formalisms incorporate, in some form, a transition system.

**Definition 2.1 Transition system**

A transition system is a triple \((S, L, T)\), where:

- \(S\) is a finite or countable set;
- \(L \subseteq S\);
- \(T \subseteq S \times S\).

\(S\) is called the **state space**, \(L\) the set of **initial states** and \(T\) the **transition relation**.

This definition of a transition system can be found in [3]. In literature, also other classes of transition systems have been described, for instance see [11,19,22,28]. They differ from ours in mainly two items, viz.:

- There is only one initial state;
- There is a set of actions or labels \(A\) such that \(T \subseteq S \times A \times S\), where \((s, a, s') \in T\) iff action \(a\) can make the system move from state \(s\) to state \(s'\).

Hence, there may be different transitions between two states, while in Definition 2.1 only the presence or absence of a transition can be indicated.

All these classes of transition systems can be transformed into each other, but we shall not elaborate that here.

A transition system may have several states which are never reachable, i.e. the system can never get there when it starts in an initial state. In some sense, these states are superfluous, yet we do not require all states be reachable. When specifying a transition system, one may not know which states are reachable and which are not. It may also be convenient to define the state space too large.

### 2.1 Standard Des model

In this subsection, we informally describe a mathematical model for discrete event systems, called the **Des model**. It is a restricted version of Coloured Nets [16]. Like all Petri Net models, it has a static topology. We shall not extensively discuss the motivations behind it and its properties, instead we refer to [7]. A formal description can also be found in [9].

Like all high-level Petri Net models, the Des model is an extension of the Elementary Net model [28]. We have developed a tool, called EXSPECT [9,10], to specify and
simulate a Des. Hence, specifications of systems described as Des are suitable for prototyping. In order to avoid ambiguities with respect to Definition 2.1, we shall use other names for the constituents of a Des than the usual ones.

A Des consists of two kinds of components: Processors and channels, which correspond to transitions and places in Petri nets. A processor is connected with several input and output channels. To each channel a type is associated and to each processor a function. The signature of the function of a processor is such that the types of its input parameters are identical to the types of the input channels and the types of its output parameters are bags over the types of the output channels. A channel may be shared by several processors as input or output channel. The channels may contain so-called tokens. A token has a value according to the type of the channel, in which it resides. More than one token of the same value may reside in a channel, so a channel contains a bag over its type.

For each processor, every input channel has a multiplicity, which corresponds to the number of input parameters of the processor function for that channel. At each moment a transition can occur, which means that the configuration of tokens in the channels may change. Such a transition happens instantaneously and is executed by the processors. A processor that has enough tokens in each input channel may pick as many tokens as it needs and produce a finite amount of new tokens for its output channels, according to its function. Several processors may produce tokens for the same output channel. A channel having always exactly one token can be seen as a memory of the system. We call such a channel a store. If a processor wants to use the store, it picks out the token and instantaneously places back a new token into the store. In fact, it replaces the token. A store can even be a database. In this case, its type is probably very complex. A marking of tokens distributed over channels is called a state.

The systems we consider have to be closed, i.e. the formalism does not explicitly support communications with an environment. Yet, in modeling practical systems it is useful to distinguish an open subsystem and an environment. Often the processors and channels in the open (sub-) system are fully specified while the constituents of the environment are left unknown.

There are several ways to complete an open system with an environment. One way is to use input channels with an infinite amount of tokens. Such channels represent input streams. Another way is to have a processor and a channel in the environment such that the channel has a finite amount of tokens and the processor repeatedly picks a token from the channel and produces two tokens identical to the input token, one for the channel in the environment from which it got the input token and one for the open subsystem. In this way, the environment behaves like a random generator of tokens. Other, more complex approaches, might support a relation between the produced output of the open subsystem and the next input to be generated.

To indicate the processors and channels and their input and output relations, we use a diagram technique where processors are represented by boxes, channels by circles and stores are marked with a dot. For input/output relations we use arrows and lines and for each input relation we mention its multiplicity, except for relations with multiplicity one, which is the default value. Processors of an environment (i.e., processors with an unknown specification) are marked with lines and channels of an environment are not drawn. See Figure 2.2 for an example.
The Des model actually resembles Coloured Nets very much. The major differences are:

- The selection of tokens differs. In our model, a processor has to accept all tokens, whereas in Coloured Nets a transition might refuse certain tokens because of their values. But we can simulate such a refusal by simply placing back undesirable tokens. (Of course, it can be simulated in a more sophisticated way.)

- Our processors are inherently deterministic, for they are functions. In Coloured Nets, transitions can act nondeterministically because of the concept of occurrence colours. However, a nondeterministic processor could be modeled with the aid of an extra channel having a distinct token for each possible action, see Figure 2.3.

---

**Figure 2.2**

![Diagram of Des model and Coloured Nets comparison](image)

**Figure 2.3** *Nondeterministic processor*
2.2 Real-time variant

In this subsection, we informally describe a slight extension of the standard Des model in order to describe real-time aspects of systems. We call the resulting model the Real-Time Des model. The tool EXSPECT has been based hereupon. For a formal description we refer to [7,9], but all concepts regarding the modeling of time can also be found in the DTN model (see Section 4).

A Real-Time Des is a standard Des in which each token has, besides its value, a time stamp. This time stamp denotes the availability time of the token and is interpreted as the earliest time the token may be consumed. When a processor performs an action (fires), it consumes some tokens and produces several new tokens. Now such an action has an action time, which is the maximum time stamp of all consumed tokens. Each processor can locally compute its action time. We require the time stamp of all produced tokens to be at least as large as the action time.

A processor can fire as soon as all input channels have enough tokens and no processor is able to perform an action at an earlier time. As a consequence, a processor can no longer locally (i.e., independent of other ones) determine whether it can fire, the network has to take care for that. (Note that the value of a token has still no influence on the firing.) This is done as follows.

An event is a simultaneous execution of some actions. Presently, such an event has an event time, which is the maximum of all action times in that event. Though we do not have any absolute clock in our model, we do have a notion of transition time, which is the time the system will jump to another state. The transition time is a mathematical function that assigns to each state a time stamp. (Please remember: A state is a marking of tokens distributed over channels.) It is the minimum event time of all possible events in that state. Hence, only actions with an action time equal to the transition time are allowable.

We do have parallelism in a Real-Time Des, however, if several processors can fire simultaneously then their action times must be equal and as least as possible.

By means of an input stream, a Real-Time Des can be initiated with infinitely many tokens. Each token has a time stamp, so we are able to determine its arrival time. Hence, it is possible to initiate a system with some tokens that will arrive in future.

An interesting property of a Real-Time Des is the establishment that the transition time cannot decrease when the system evolves. For a formal proof, we refer to [7].

The models we have discussed until now have a static topology, no new components can be created during the course of the system and the interaction structure is fixed. In the next section, we briefly introduce a model with a dynamic topology.

3 Dynamic topology

There exists a class of discrete event systems with a dynamic topology, i.e. new components can be created during the course of the system and the interaction structure may change. The Actor system [1,8,15] is one of the most well-known specimens of this class. In this section, we first very briefly introduce Actor systems. After that, we
show that any Actor system can be realized within the standard Des model.

3.1 Actors

An Actor system consists of actors that process communications. Actors have a sort of memory that informs them about the presence of other actors. Upon processing a communication, an actor:

- sends communications to known actors,
- replaces its behaviour (its internal state),
- produces new actors.

The dynamic behaviour of Actor systems is in all three concepts. This is obvious for the last concept. However, the second one implies that the set of known actors may change during the course of the system. That results, together with the first one, in a changing interaction structure. Please note that the actor population cannot shrink, as there is no option for deletion of actors.

This is a very global description of Actor systems. Of course, there are a lot of constraints and things to be worked out in more detail, e.g. communications and actors have a unique name, both communications and internal states contain values and names of known actors, actors may not send communications to unknown actors etc.. For more details, we refer to [1,8].

We present a formal Actor model in [8]. Agha [1] has already introduced a formal model, however, he uses recursive domain equations. This method has some disadvantages, for instance it is not immediately clear whether such an equation has solutions, and if so, which ones. A complex fixpoint theory is needed to find a minimal solution (e.g. see [2,25,26,27]). Therefore, we have chosen a more elementary approach. The semantics of our Actor model is in terms of transition systems.

3.2 Actor systems as Des (1)

In this subsection, we show that an arbitrary Actor system can easily be realized within the Des model. For a formal definition of the concept ‘realization’, see [8].

The idea is as follows. We construct a Des with one processor and two channels, as in Figure 3.1. Channel act is a store, having a table of Actor names with corresponding behaviours (internal states). Channel com contains communications. Processor p repeatedly takes a single communication from com. This communication is destined for some actor α in the table of act. Then p processes the communication: New communications are send to com, the behaviour of α is updated and possible new actors are added to the table.
A formal construction can be found in [8]. This paper also gives a formal proof that the thus constructed Des realizes the original Actor system. In this proof, two transition systems with a completely different state space structure are compared with each other. Please note: Both Des' ses and Actor systems specify transition systems.

This Des has no parallelism, since there is only one processor. However, it can perform each computation of the original Actor system, because each parallel computation can be imitated sequentially. An obvious conclusion is the establishment that systems with dynamic and static topologies have an equal computational power.

A disadvantage of the presented construction is that all structure is hidden in the table of act. This is like programming large systems in machine language: Possible, but not practical. Therefore we are looking for better constructions.

4 Dynamic process creation in models with a static topology

In this section, we first show that a standard high-level Petri Net suffices: We give a construction to describe processes with a dynamic topology in the standard Des model, without the disappearance of parallelism (as in the previous section). However, we have our doubts about the benefit of the resulting Des. We do not think that dynamic topology processes should be modeled in this way. Nevertheless, the construction has enabled us to derive some useful concepts for the modeling of such processes within a framework with a static topology. We finally elaborate these concepts into a formal definition of the so-called Dynamic Time Net (DTN) model.

4.1 Actor systems as Des (2)

This time we construct an equivalent Des for each Actor system, that is, their corresponding transition systems are equal, up to a renaming of the states. As a consequence, the resulting Des must exhibit parallelism, for an Actor system has an inherently parallel behaviour.

Let an Actor system be given. For each possible actor $\alpha$ in the Actor system, we use a processor with name $\alpha$ in the Des. Hence, we need an infinite chain of processors, since we do not know in advance how many actors will be created during the evolution of the Actor system.

Processor $\alpha$ has two input channels with multiplicity one. One of them, $c_\alpha$, is used to collect communications for $\alpha$ and the other one, $b_\alpha$, is a kind of pseudo-store for the behaviour of $\alpha$. It will contain at most one token. In case $b_\alpha$ is empty, the corresponding actor is not (yet) present in the Actor system, otherwise actor $\alpha$ has been created.
Each processor has every channel in the system as output channel, so it can send communications to every processor in the system and it can initiate another processor $\alpha'$ (it can create the corresponding actor) by sending a token to $b_{\alpha'}$. Of course, $b_{\alpha'}$ must then be empty, otherwise $\alpha'$ would have more than one internal state. This is accounted for by construction, since an Actor system does not create two different actors with an identical name.

Figure 4.1 gives a picture. A thick line represents an infinite number of output relations.

As in Subsection 3.2, we refer to [8] for a formal construction and an equivalence proof. We conclude that it is possible to model dynamic topology processes within standard high-level Petri nets. It is even possible to implement the above described Des, in spite of the infinite number of processors. The actor population will always be finite when the system starts in a finite one, because an actor produces only finitely many new actors. A processor creation can be implemented, for example, with a kind of new-instruction as in Pascal [17].

The above construction might be used to prototype Actor systems in Petri Nets. However, we do not recommend it as a standard method to model dynamic topology processes within Petri Nets. It has some disadvantages which we shall discuss in the next subsection. Yet, it enabled us to derive some useful concepts for the modeling of process creations in a framework with a static topology.

### 4.2 Dynamic Time Net model

In this subsection, we present a high-level Petri Net model with a time component, for the description of processes with a dynamic topology. We call it the Dynamic Time
Net (DTN) model. Like all Petri Net models, it has a static topology. Its semantics is given in terms of transition systems.

The model has been based upon the construction in the previous subsection. That construction, however, has some disadvantages:

- The specification has a lot of redundancy: Infinitely many channels have the same type and the standard Des model prescribes to specify a type for each channel. It would be preferable to mention that there is a class of channels with a certain type. A similar argument applies to processors;

- There are only two classes of channels and only one class of processors. We want to specify more classes, in order to represent more structure of a system in the topology of our network;

- Our standard diagramming technique is not appropriate anymore with regard to infinitely large structures as in Figure 4.1. It would be nice if we could depict such structures in a finite way.

Several useful concepts have already been mentioned in the above, e.g. the concept of processor and channel classes and the concept of infinitely many output relations. The latter allows a processor to initiate an unbounded number of other processes during its lifetime. The standard Des model forbids to have infinitely many input relations, so there is an asymmetry between input and output relations. To account for that, we define a processor class \( p \) and a channel class \( c \) to consist of an infinite number of constituents \( p.i \) and \( c.j \) respectively \( (i, j \in \mathbb{N}_0) \), where processor \( p.i \) has all \( c.j \) as output channel if processor class \( p \) has channel class \( c \) as output channel class and processor \( p.i \) has only \( c.i \) as input channel if processor class \( p \) has channel class \( c \) as input channel class.

With respect to the diagramming technique, please notice the resemblance between Figures 4.1 and 3.1. Why not use Figure 3.1 as a graphical representation of the standard Des from Figure 4.1? Now boxes do no longer stand for single processors, instead they represent classes of processors with an identical function. Circles denote classes of channels with identical types. Shared input channels can be modeled too, consider for example Figure 4.2. Here we have two processor classes \( p \) and \( q \) and two channel classes \( c \) and \( d \). The processor classes share channel class \( c \) as input and channel class \( d \) as output. Processor \( p.i \ (i \in \mathbb{N}_0) \) shares an input channel \( c.i \) with processor \( q.i \) and they have all \( d.j \ (j \in \mathbb{N}_0) \) as output channel.
Figure 4.2 Dynamic Des with corresponding standard Des

We introduce some notations in order to present the DTN formalism.

Notations
\( \mathbb{N}_0 \) is the set of natural numbers and for \( i \in \mathbb{N}_0 \), \( \mathbb{N}_i = \{ j \in \mathbb{N}_0 \mid j \geq i \} \). The symbol \( '\infty' \) stands for 'infinite'. For any ordered set \( S \) and \( s \in S : s < \infty \).

For \( A \) and \( B \) sets, \( A \rightarrow B \) denotes the set of all total functions from \( A \) to \( B \) and \( A \notightarrow B \) the set of all partial functions from \( A \) to \( B \). \( \mathcal{P}(A) \) denotes the set of all subsets of \( A \) and \( \mathcal{B}(A) \) denotes the set of all multisets (bags) over \( A \), i.e. the set of all total functions from \( A \) to \( \mathbb{N}_0 \cup \{\infty\} \). Please note: Infinitely many copies of the same element can appear in a bag and a bag can contain infinitely many different elements.

For \( b \in \mathcal{B}(A) \) and \( x \) some element: \( x \in b \) iff \( x \in A \) and \( b(x) > 0 \).

For \( x \in \mathcal{B}(A) \) and \( y \in \mathcal{B}(B) \):

- \( x \subseteq y \) iff \( \forall a \in x : a \in B \land x(a) \leq y(a) \).
- \( x = y \) iff \( x \subseteq y \land y \subseteq x \).
- \( x \cup y = \lambda a \in A \cup B : \) if \( a \in A \setminus B \) then \( x(a) \)
  else if \( a \in B \setminus A \) then \( y(a) \)
  else \( x(a) + y(a) \) fi fi.
- \( x \setminus y = \lambda a \in A : \) if \( a \in B \) then \( \max \{0, x(a) - y(a)\} \)
  else \( x(a) \) fi.
- \( x \cap y = x \setminus (x \setminus y) \).

Please notice that \( x \cap y = y \cap x \).

Sets can be viewed as bags. If \( S \) is a set, then the corresponding bag \( \tilde{S} \in \mathcal{B}(S) \) is
defined as \( s \in S : 1 \). So we can apply the operations above to sets and bags. For 
\( Y \) a set of bags or a set of sets, \( \cup Y \) denotes the union of all elements of \( Y \). If \( X \) is a 
singleton bag or set, then \( \sigma X \) is the element of \( X \).

\[ \square \]

We shall now formalize the DTN model. A token will have, besides its value, a time 
stamp denoting its availability time and a version number [4] indicating in which chan­
nel of a certain class it resides.

**Definition 4.1  Dynamic Time Net model**
A Dynamic Time Net (DTN) is a 5-tuple \( (R,V,I,O, TI) \) where \( TI \) is a totally ordered 
set, called the time set, and \( R, V, I \) and \( O \) are functions, such that:

- \( P := \text{dom}(R) \) is a set of processor class indices;
- \( C := \text{dom}(V) \) is a set of channel class indices;
- \( P \) and \( V \) are finite or countable;
- \( I \in P \to \mathcal{B}(C) \setminus \{\emptyset\} \), \( I(p) \) is the bag of input channel classes of processor class 
\( p \);
- \( O \in P \to \mathcal{P}(C) \), \( O(p) \) is the set of output channel classes of processor class \( p \);
- \( \forall p \in P : I(p) \) is finite;
- \( \forall c \in C : V(c) \) is finite or countable, it denotes the value set of channel class \( c \);
- \( Q := \{ (c,n,t,v) \mid c \in C \land n \in \mathbb{N}_0 \land t \in TI \land v \in V(c) \} \) is the set of tokens;
- \( \forall (c,n,t,v) \in Q \), \( n \) is the version number, \( t \) the time stamp, denoting the 
availability time and \( v \) the value;
- \( \forall p \in P : R(p) \subset \mathcal{B}(Q) \) is such that

\[
\text{dom} R(p) = \{ b \in \mathcal{B}(Q) \mid (\lambda c \in C : \sum_{n \in \mathbb{N}_0, t \in TI, v \in V(c)} b((c,n,t,v))) = I(p) \land \\
\forall (c,n,t,v),(c',n',t',v') \in b : n = n' \} \land \\
\forall b \in \text{dom} R(p) : R(p)(b) \text{ is finite} \land \\
\forall (c,n,t,v) \in R(p)(b) : c \in O(p) \land \\
t \geq \max \{ t' \in TI \mid \exists c' \in C, n' \in \mathbb{N}_0, v' \in V(c') : (c',n',t',v') \in b \}
\]

\( R(p) \) denotes the reaction function of processor class \( p \).

\[ \square \]

The requirements on \( \text{dom} R(p) \) state that \( R(p) \) may only be applied to tokenbags with 
the right number of tokens in each input channel class and all consumed tokens must 
have the same version number. The requirements about \( R(p)(b) \) indicate that a finite
amount of tokens is produced for only the output channel classes and the time stamps of the produced tokens are not less than any time stamp of the consumed tokens. To illustrate the definition above, we present an example.

**Example 4.1 Message duplication**
Consider a system with a class of users $u$ and a class of servers $s$. Users want messages to be duplicated. A user sends a message, together with a duplication number and his identification number, to some server of class $s$. Each individual server $s.i (i \in \mathbb{N}_0)$ performs the following task: If the duplication number is one, it returns the message to the user, otherwise it equally distributes its task between servers $s.(2i + 1)$ and $s.(2i + 2)$. All actions take an equal amount of time. In this way, a user receives the desired amount of messages in logarithmic time.

We construct a DTN $\langle R, V, I, O, TI \rangle$ for this system. We define $TI := \mathbb{N}_0$ and we assume each action to take one time unit.

$$P := \{u, s\}$$
$$C := \{c, c'\}$$

Users put their messages in channel class $c$ and servers return duplicates in channel class $c'$.

$$I := \{(u, \{c'\}), (s, \{c\})\}$$
$$O := \{(u, \{c\}), (s, \{c, c'\})\}$$

$$V := \{(c, M \times \mathbb{N}_1 \times \mathbb{N}_0), (c', M)\}$$

$M$ denotes the set of all messages and for $(m, d, i) \in V(c)$, $m$ is the message, $d$ the duplication number and $i$ the user's identification number.

See Figure 4.3 for a graphical representation.

For $n \in \mathbb{N}_0$, $t \in TI$ and $(m, d, i) \in V(c)$:

$$R(s)(\{(c, n, t, (m, d, i))\}) =$$

if $d = 1$

then $\{(c', i, t + 1, m)\}$

else $\{(c, 2n + 1, t + 1, (m, d \text{ div } 2, i)), (c, 2n + 2, t + 1, (m, d - (d \text{ div } 2), i))\}$

fl.

We do not describe $R$ for users, for they are part of the environment.
The remainder of this section is used to give the semantics of Dynamic Time Nets in terms of transition systems.

**Definition 4.2 State space, event set, event time**

Let a DTN be given. Then:

- \( S := \mathbb{B}(Q) \);
- \( E := \{ e \in P \rightarrow \mathcal{P}(S) \mid \cup \cup \text{rng}(e) \neq \emptyset, \cup \cup \text{rng}(e) \text{ is finite,} \)
  \[
  \forall p \in \text{dom}(e) : e(p) \subseteq \text{dom} R(p) \quad \text{and} \\
  \forall b, b' \in e(p) : b \neq b' \Rightarrow \text{nr}(b) \neq \text{nr}(b')
  \]
  where \( \text{nr}(b) = \sigma \{ n \in \mathbb{N}_0 \mid \exists c \in C, t \in TI, v \in V(c) : (c, n, t, v) \in b \} \)
- \( h \in E \rightarrow TI \) with for \( e \in E, \)
  \[
  h(e) = \max \{ t \in TI \mid \exists c \in C, n \in \mathbb{N}_0, v \in V(c) : (c, n, t, v) \in \cup \cup \text{rng}(e) \}.
  \]

\( S \) is called the *state space*, \( E \) the *event set* and function \( h \) assigns to each event an *event time*.

\( \square \)

An event \( e \) is an assignment of a set of tokenbags to each processor class \( p \) and \( \cup \cup \text{rng}(e) \) is the bag of all tokens involved. Each tokenbag \( b \in e(p) \) is destined for some processor \( p.n \) of class \( p \) (\( n \in \mathbb{N}_0 \)). Please note that \( n \) is uniquely determined by \( b \), since \( b \) can not be empty and all tokens in \( b \) have the same version number. For each input channel with multiplicity \( m \), \( b \) has exactly \( m \) tokens. The last requirement on \( E \) states that different bags in \( e(p) \) must be assigned to different processors, i.e. in an event, a processor may not fire more than once.

It should be possible to identify the current population of processes for a system with process creation. Therefore, we introduce a notion of *aliveness*:

**Definition 4.3 Processor aliveness**

Let \( s \in S, p \in P \) and \( n \in \mathbb{N}_0 \). Processor \( p.n \) is *alive* at state \( s \) iff

\[
\exists c \in I(p) : \exists t \in TI : \exists v \in V(c) : (c, n, t, v) \in s.
\]
In our model, processors may not only become alive (be initiated), they may also starve. In Example 4.1, a request for $2^n$ duplicates of some message at time $t$ would result in a population of $2^n$ alive processors at time $t + n + 1$. One time unit later, all these processors have responded and they are not alive anymore (if, meanwhile, no user has entered another request).

An event $e$ can happen at state $s$ if and only if the following two requirements are met:

- The tokenbag involved in $e$ must be present at $s$, i.e. $\cup \cup \text{rng}(e) \subseteq s$;
- No other event can occur at an earlier time.

For the second requirement, we introduce the concept of transition time, which gives for each state the time of the next transition. The event function assigns to each state all possible events.

**Definition 4.4 Transition time, event function**

The transition time $H$ and the event function $F$ of a DTN satisfy:

$$H \in S \rightarrow TI \cup \{\infty\}, \quad F \in S \rightarrow IP(E)$$

and for $s \in S$,

$$H(s) = \min \{h(e) \mid e \in E \land \cup \cup \text{rng}(e) \subseteq s\};$$

$$F(s) = \{e \in E \mid \cup \cup \text{rng}(e) \subseteq s \land h(e) = H(s)\}.$$  

Of course, the event function assigns tokens to alive processors only. We shall finally define the semantics of the DTN model in terms of transition systems.

**Definition 4.5 Semantics**

Let a DTN be given with reaction function $R$, state space $S$, event function $F$ and let $L$ be an arbitrary set of initial states, $L \subseteq S$. The corresponding transition system is

$$(S, L, T)$$

where

$$T = \{(s, t) \in S \times S \mid \exists e \in F(s) : t = (s \cup \cup \text{rng}(e)) \cup \bigcup_{p \in \text{dom}(e), \ b \in e(p)} R(p)(b)\}.$$
Some basic results with regard to time aspects can be found in [7]. For example, an important theorem concerns the transition time, which cannot decrease: For all \((s, t) \in T, H(s) \leq H(t)\). We are currently working on more theoretical results, especially with regard to dynamic aspects.

It is not difficult to obtain the original Des and Real-Time Des models from the DTN model. To abstract from dynamic aspects, it suffices to require all initial version numbers to equal some constant \(c (c \in \mathbb{N}_0)\) and to forbid the production of tokens with other version numbers. The resulting model is basically identical to the Real-Time Des model. If, furthermore, the time set \(TI\) is a singleton, then we obtain the standard Des model.

Any Actor system can be very easily described as DTN. Of course, as we indicated in Subsection 4.1, a single processor class suffices. But the DTN model supports the modeling of more structure. Consider, for example, an Actor system with two different sorts of actors (such as factorial actors and customers in [1]). It would be obvious to define a processor class for each sort of actors.

We have presented a model for distributed systems that allows a concise description of time aspects and dynamic aspects, with the ability to maintain certain structural features of the modeled system.

5 Conclusion

We do not have to use models with a dynamic topology in order to describe systems with process creations. Petri Net based models serve as well, because the token mechanism is inherently dynamic.

We have used a construction of an equivalent Des for each Actor system as basis for our DTN model. The latter is a high-level Petri Net model with time component, where existing processes can create new ones during the course of the system. Of course, as we indicated in Section 4, it is possible to model dynamic topology processes with ordinary high-level nets, but that makes the description of such systems much longer than in the DTN formalism.

In further research, we shall mainly be engaged in two topics. First, we intend to describe a lot of examples of systems with a dynamic topology, coming from literature and practice, as DTN and we probably have to extend the tool EXSPECT in order to prototype them. Next, we are working on a theoretical basis for the DTN formalism with regard to invariants, time aspects, dynamic aspects and composition/decomposition.

Acknowledgement

We would like to thank Kurt Jensen for several valuable suggestions.
References


Revised by A.B. Mickel and J.F. Miner.

[18] **Jones C.B.**: *Systematic software development using VDM.*


[21] **Mazurkiewicz A.**: *Concurrent Program Schemes and Their Interpretation.*

[22] **Milner R.**: *A calculus of communicating systems.*

[23] **Reisig W.**: *Place/Transition systems.*


[25] **Scott D.**: *Data Types as Lattices.*


[27] **Tarski A.**: *A Lattice-Theoretical Fixpoint Theorem and its Applications.*

In this series appeared:

<table>
<thead>
<tr>
<th>No.</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>85/01</td>
<td>R.H. Mak</td>
<td>The formal specification and derivation of CMOS-circuits.</td>
</tr>
<tr>
<td>85/02</td>
<td>W.M.C.J. van Overveld</td>
<td>On arithmetic operations with M-out-of-N-codes.</td>
</tr>
<tr>
<td>85/03</td>
<td>W.J.M. Lemmens</td>
<td>Use of a computer for evaluation of flow films.</td>
</tr>
<tr>
<td>85/04</td>
<td>T. Verheoff</td>
<td>Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate.</td>
</tr>
<tr>
<td></td>
<td>H.M.L.J. Schols</td>
<td></td>
</tr>
<tr>
<td>86/01</td>
<td>R. Koymans</td>
<td>Specifying message passing and real-time systems.</td>
</tr>
<tr>
<td>86/02</td>
<td>G.A. Bussing</td>
<td>ELISA, A language for formal specification of information systems.</td>
</tr>
<tr>
<td></td>
<td>K.M. van Hee</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M. Voorhoeve</td>
<td></td>
</tr>
<tr>
<td>86/03</td>
<td>Rob Hoogerwoord</td>
<td>Some reflections on the implementation of trace structures.</td>
</tr>
<tr>
<td>86/04</td>
<td>G.J. Houben</td>
<td>The partition of an information system in several systems.</td>
</tr>
<tr>
<td></td>
<td>J. Paredaens</td>
<td></td>
</tr>
<tr>
<td></td>
<td>K.M. van Hee</td>
<td></td>
</tr>
<tr>
<td>86/05</td>
<td>J.L.G. Dietz</td>
<td>A framework for the conceptual modeling of discrete dynamic systems.</td>
</tr>
<tr>
<td></td>
<td>K.M. van Hee</td>
<td></td>
</tr>
<tr>
<td>86/06</td>
<td>Tom Verheoff</td>
<td>Nondeterminism and divergence created by concealment in CSP.</td>
</tr>
<tr>
<td>86/07</td>
<td>R. Gerth</td>
<td>On proving communication closedness of distributed layers.</td>
</tr>
<tr>
<td></td>
<td>L. Shira</td>
<td></td>
</tr>
<tr>
<td>86/08</td>
<td>R. Koymans</td>
<td>Compositional semantics for real-time distributed computing (Inf.&amp;Control 1987).</td>
</tr>
<tr>
<td></td>
<td>R.K. Shyamasundar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W.P. de Roever</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R. Gerth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S. Arun Kumar</td>
<td></td>
</tr>
<tr>
<td>86/09</td>
<td>C. Huizing</td>
<td>Full abstraction of a real-time denotational semantics for an OCCAM-like language.</td>
</tr>
<tr>
<td></td>
<td>R. Gerth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W.P. de Roever</td>
<td></td>
</tr>
<tr>
<td>86/10</td>
<td>J. Hooman</td>
<td>A compositional proof theory for real-time distributed message passing.</td>
</tr>
<tr>
<td>86/11</td>
<td>W.P. de Roever</td>
<td>Questions to Robin Milner - A responder’s commentary (IFIP86).</td>
</tr>
<tr>
<td>86/12</td>
<td>A. Boucher</td>
<td>A timed failures model for extended communicating processes.</td>
</tr>
<tr>
<td></td>
<td>R. Gerth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W.P. de Roever</td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>86/14</td>
<td>R. Koymans</td>
<td>Specifying passing systems requires extending temporal logic.</td>
</tr>
<tr>
<td>87/01</td>
<td>R. Gerth</td>
<td>On the existence of sound and complete axiomatizations of the monitor concept.</td>
</tr>
<tr>
<td>87/02</td>
<td>Simon J. Klaver, Chris F.M. Verberne</td>
<td>Federatieve Databases.</td>
</tr>
<tr>
<td>87/03</td>
<td>G.J. Houben, J.Paredaens</td>
<td>A formal approach to distributed information systems.</td>
</tr>
<tr>
<td>87/04</td>
<td>T. Verhoeff</td>
<td>Delay-insensitive codes - An overview.</td>
</tr>
<tr>
<td>87/05</td>
<td>R. Kuiper</td>
<td>Enforcing non-determinism via linear time temporal logic specification.</td>
</tr>
<tr>
<td>87/06</td>
<td>R. Koymans</td>
<td>Temporele logica specificatie van message passing en real-time systemen (in Dutch).</td>
</tr>
<tr>
<td>87/07</td>
<td>R. Koymans</td>
<td>Specifying message passing and real-time systems with real-time temporal logic.</td>
</tr>
<tr>
<td>87/08</td>
<td>H.M.J.L. Schols</td>
<td>The maximum number of states after projection.</td>
</tr>
<tr>
<td>87/10</td>
<td>T. Verhoeff</td>
<td>Three families of maximally nondeterministic automata.</td>
</tr>
<tr>
<td>87/11</td>
<td>P. Lemmens</td>
<td>Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.</td>
</tr>
<tr>
<td>87/12</td>
<td>K.M. van Hee and A. Lapinski</td>
<td>OR and AI approaches to decision support systems.</td>
</tr>
<tr>
<td>87/13</td>
<td>J.C.S.P. van der Woude</td>
<td>Playing with patterns - searching for strings.</td>
</tr>
<tr>
<td>87/14</td>
<td>J. Hooman</td>
<td>A compositional proof system for an occam-like real-time language.</td>
</tr>
<tr>
<td>87/16</td>
<td>H.M.M. ten Eikelder, J.C.F. Wilmont</td>
<td>Normal forms for a class of formulas.</td>
</tr>
<tr>
<td>87/17</td>
<td>K.M. van Hee, G.J. Houben, J.L.G. Dietz</td>
<td>Modelling of discrete dynamic systems framework and examples.</td>
</tr>
</tbody>
</table>
87/18  C.W.A.M. van Overveld  An integer algorithm for rendering curved surfaces.

87/19  A.J. Seebregts  Optimalisering van file allocatie in gedistribueerde database systemen.

87/20  G.J. Houben  The $R^2$-Algebra: An extension of an algebra for nested relations.
        J. Paredaens

87/21  R. Gerth  Fully abstract denotational semantics for concurrent PROLOG.
        M. Codish
        Y. Lichtenstein
        E. Shapiro

88/01  T. Verheoef  A Parallel Program That Generates the Möbius Sequence.

88/02  K.M. van Hee  Executable Specification for Information Systems.
        G.J. Houben
        L.J. Somers
        M. Voorhoeve

88/03  T. Verheoef  Settling a Question about Pythagorean Triples.

88/04  G.J. Houben  The Nested Relational Algebra: A Tool to Handle Structured Information.
        J. Paredaens
        D. Tahon

88/05  K.M. van Hee  Executable Specifications for Information Systems.
        G.J. Houben
        L.J. Somers
        M. Voorhoeve

88/06  H.M.J.L. Schols  Notes on Delay-Insensitive Communication.

88/07  C. Huizing  Modelling Statecharts behaviour in a fully abstract way.
        R. Gerth
        W.P. de Roever

88/08  K.M. van Hee  A Formal model for System Specification.
        G.J. Houben
        L.J. Somers
        M. Voorhoeve

88/09  A.T.M. Aerts  A Tutorial for Data Modelling.
        K.M. van Hee

88/10  J.C. Ebergen  A Formal Approach to Designing Delay Insensitive Circuits.

88/11  G.J. Houben  A graphical interface formalism: specifying nested relational databases.
        J. Paredaens

88/12  A.E. Eiben  Abstract theory of planning.

88/13  A. Bijlsma  A unified approach to sequences, bags, and trees.
88/14 H.M.M. ten Eikelder
R.H. Mak
Language theory of a lambda-calculus with recursive types.

88/15 R. Bos
C. Hemerik
An introduction to the category theoretic solution of recursive domain equations.

88/16 C. Hemerik
J.P. Katoen
Bottom-up tree acceptors.

88/17 K.M. van Hee
G.J. Houben
L.J. Somers
M. Voorhoeve
Executable specifications for discrete event systems.

88/18 K.M. van Hee
P.M.P. Rambags
Discrete event systems: concepts and basic results.

88/19 D.K. Hammer
K.M. van Hee
Fasering en documentatie in software engineering.

88/20 K.M. van Hee
L. Somers
M. Voorhoeve
EXSPECT, the functional part.

89/1 E.Zs.Lepoeter-Molnar
Reconstruction of a 3-D surface from its normal vectors.

89/2 R.H. Mak
P. Struik
A systolic design for dynamic programming.

89/3 H.M.M. Ten Eikelder
C. Hemerik
Some category theoretical properties related to a model for a polymorphic lambda-calculus.

89/4 J. Zwiers
W.P. de Roever
Compositionality and modularity in process specification and design: A trace-state based approach.

89/5 Wei Chen
T. Verhoeff
J.T. Udding
Networks of Communicating Processes and their (De-)Composition.

89/6 T. Verhoeff
Characterizations of Delay-Insensitive Communication Protocols.

89/7 P. Struik
A systematic design of a parallel program for Dirichlet convolution.

89/8 E.H.L. Aarts
A.E. Eiben
K.M. van Hee
A general theory of genetic algorithms.

89/9 K.M. van Hee
P.M.P. Rambags
Discrete event systems: Dynamic versus static topology.

89/10 S. Ramesh
A new efficient implementation of CSP with output guards.

89/11 S. Ramesh
Algebraic specification and implementation of infinite processes.
<table>
<thead>
<tr>
<th>Year/Volume</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>89/12</td>
<td>A.T.M. Aerts, K.M. van Hee</td>
<td>A concise formal framework for data modeling.</td>
</tr>
<tr>
<td>89/14</td>
<td>H.C. Haesen</td>
<td>ELDA, data manipulatie taal.</td>
</tr>
<tr>
<td>89/15</td>
<td>J.S.C.P. van der Woude</td>
<td>Optimal segmentations.</td>
</tr>
<tr>
<td>89/16</td>
<td>A.T.M. Aerts, K.M. van Hee</td>
<td>Towards a framework for comparing data models.</td>
</tr>
<tr>
<td>89/17</td>
<td>M.J. van Diepen, K.M. van Hee</td>
<td>A formal semantics for Z and the link between Z and the relational algebra.</td>
</tr>
<tr>
<td>90/2</td>
<td>K.M. van Hee, P.M.P. Rambags</td>
<td>Dynamic process creation in high-level Petri nets, pp. 19.</td>
</tr>
</tbody>
</table>