TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Discrete event systems : concepts and basic results

Citation for published version (APA):
Hee, van, K. M., & Rambags, P. M. P. (1988). *Discrete event systems : concepts and basic results*. (Computing science notes; Vol. 8818). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/1988

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 19. Apr. 2025

Discrete event systems: concepts and basic results

by

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

88/18

December, 1988

COMPUTING SCIENCE NOTES

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

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
All rights reserved
editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

DISCRETE EVENT SYSTEMS: CONCEPTS AND BASIC RESULTS

by

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

1. Introduction

The systems we consider are discrete, which means that their behavior can be described by sequences of successive states. Many frameworks are developed to describe discrete systems, for instance Markov-chains [Freedman 71], automata and Turing machines [Lewis, Papadimitriou 81], trace structures [Rem 83, Mazurkiewicz 84], process algebra [Bergstra, Klop 84] and Petri nets [Petri 80].

In this paper we introduce a framework that is closely related to Petri nets. We call a system that fits into our framework a *discrete event system* (des). Before we define this framework, we give a more general notion of a system. We start with some notations.

Notations

$\mathbb{N}_0$ is the set of natural numbers including zero. $\mathbb{N}_1$ is the set of positive natural numbers.
For S a set and T a binary relation over S, T^* is the transitive closure of T. If A and B are sets, then $A \rightarrow B$ denotes the set of all total functions from A to B and $A \nrightarrow B$ the set of all partial functions from A to B. For notational clarity we often write the function application $f(x)$ by f_x. We denote function restriction by $\restriction$. For f a partial function from A to B and $R \subseteq B$, $f \restriction R$ is the range-projection of f to R, i.e. $f \restriction R = \{(x,y) \in f \mid y \in R\}$. Note that $\forall x \in \mathrm{dom}(f) \ : \ x \in \mathrm{dom}(f \restriction R)$ iff $f(x) \in \mathrm{rng}(f \restriction R)$.

Most frameworks incorporate, in some form, the following notion of a system, here called *basic system.* A basic system consists of a state space S, a set $L \subseteq S$ of initial states and a binary relation T over S, called the *transition relation.* If the system is in some state $s \in S$ it may move to a state $t \in S$ if and only if $< s,t > \in \ T$.

Definition 1 *Basic system.*
A basic system is a triple $< S,L,T >$ where S is a finite or countable set, $L \subseteq S$ and $T \subseteq S \times S$.
S is called the *state space*, L the set of *initial states* and T the *transition relation.* Elements of $S_0 \ := \ \{s \in S \mid \neg \exists t \in S \ : < s,t > \in \ T\}$ are called *terminal states.*

Note that a basic system is a directed graph. Now we introduce the conepts *path* and *process.* A *path* is a sequence of states such that any successive pair belongs to the transition relation. A path starts in an initial state and either it is infinite or it ends in a terminal state. The *process* of a basic system is the set of all paths. A finite initial segment of a path is called a *trace.*
A path may be considered as a (partial) function from $\mathbb{N}$ to S. The term *behavior* is used here as a synonym for process.

Definition 2 *Process.*
Let $< S,L,T >$ be a basic system with set of terminal states S_0. The process of Π of it satisfies:

$$\begin{aligned} \Pi = \{\sigma \in \mathbb{N}_0 \rightarrow S \mid\ & 0 \in \mathrm{dom}(\sigma) \wedge \sigma_0 \in L \\ & \wedge \forall n \in \mathrm{dom}(\sigma) \ : \ (n > 0 \rightarrow n-1 \in \mathrm{dom}(\sigma) \wedge < \sigma_{n-1}, \sigma_n > \in T) \\ & \wedge \forall n \in \mathrm{dom}(\sigma) \ : \ (\forall n' \in \mathrm{dom}(\sigma) \ : \ n' \leq n) \rightarrow \sigma_n \in S_0 \\ \} \ . & \end{aligned}$$

An element of Π is called a *path* and for all $\sigma \in \Pi$ and $n \in \mathbb{N}_0$:

$\sigma \restriction \{k \in \mathbb{N} \mid 0 \leq k < n\}$ is called a *trace* .

The set of all traces is called the *trace set.*

Note that a basic system may have several states which are never reachable, i.e. no path contains such a state. In some sense these states are superfluous, yet we do not require all states be reachable. When specifying a basic 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. For any basic system we define its *reachability variant*:

Definition 3 *Reachability variant.*
Let $< S, L, T >$ be a basic system with process Π. Then its reachability variant $< \tilde{S}, \tilde{L}, \tilde{T} >$ satisfies:

$$\tilde{S} = \{s \in S \mid \exists \sigma \in \Pi \ : \ \exists i \in \mathrm{dom}(\sigma) \ : \ \sigma(i) = s\}$$
$$\tilde{L} = L$$
$$\tilde{T} = T\,.$$

Note that the reachability variant of the reachability variant is the reachability variant itself.

A major problem of the description of systems is the complexity of the state space, in particular when we deal with concurrent systems. The states of these systems can be described by vectors, not necessary of fixed length, and the dimension of a realistic state space can be astronomically. In these cases it is impossible or practically infeasible to specify the transition relation by enumeration; Therefore different frameworks, each for specific domains of application, have been developed. These frameworks enable us to specify state space and transition relation in some closed form.

The concept of a basic system enables us to introduce some important notions.
A basic system is called *deterministic* if each state has at most one successor, i.e. the transition relation is a (partial) function. Other systems are called *nondeterministic.* Often nondeterministic systems are confused with *stochastic systems.* In the latter a *transition probability* is given instead of a transition relation.
A transition probability over a countable set S is a function W such that:

- $\mathrm{dom}(W) = S \times S$
- $\forall \sigma, \tau \in S \ : \ W(\sigma, \tau) \geq 0$
- $\forall \sigma \in S \ : \ (\sum \tau \in S \ : \ W(\sigma, \tau)) \leq 1\,.$

Of course such a system also defines a basic system, namely $< S, L, \{< s, t > \in S \times S \mid W(s,t) > 0\} >$, where L is the set of initial states. The difference is that in a stochastic system each trace has a probability of occurrence.

Nondeterminism is used to model unpredictable influences, sometimes within the system, for instance due to physical phenomena or external influences such as impulses from the environment of the system.
Up to now we considered a system in isolation, i.e. without any interaction with an

environment. Such systems are called *closed.* Above we have seen that nondeterminism might be interpreted as an external influence , however, often it is more natural to model the environment in a more explicit way, for instance by a 4-tuple $< \Sigma, \Lambda, A, M >$ where Σ is the *internal state space*, $\Lambda \subseteq \Sigma$ is the set of *internal initial states*, A is the *action space* and M the *transition function*, i.e. $M \in \Sigma \times A \to \Sigma$.
The actions represent the external influences. It is easy to transform such a system into a basic system $< S, L, T >$ by

$$S = \Sigma \times A\,, \quad L = \Lambda \times A \text{ and}$$
$$T = \{<< s, a >, < t, b >> \mid s, t \in \Sigma \wedge a, b \in A \wedge t = M(s, a)\}\ .$$

This system is nondeterministic and closed, while the underlying system is deterministic. Systems that explicitly model interaction with an environment are called *open.*
Finally, we distinguish *terminating systems* and *perpetual systems.* Each algorithm specifies a discrete system, mostly nondeterministic and preferably with finite lifetime, because a computation should end. Each time we use the algorithm, we create a new discrete system again. We may say that such a discrete system is an incarnation of the algorithm. Computations are typical terminating systems, however, operating systems and most logic systems are perpetual. The distinction between the two classes is seen from their processes: Terminating systems have only finite paths, while perpetual systems have only infinite paths. In practice, many systems are mixtures: They have both finite and infinite paths. From the point of view of a perpetual system, a finite path means failure and in a system that should terminate, an infinite path means failure.

In Section 2 we introduce some relationships between basic systems. In Section 3 we consider our framework: Des. In Section 4 we define the realtime variant of a des. We always use the symbols S, L and T as in Definition 1.

2. Relationships between basic systems

In systems engineering we compare systems. We may use expressions as: 'System A is more powerfull than system B', 'system A simulates system B' or 'system A is in fact the same as system B'.
For instance if we are developing a new system, we compare several designs with each other and also with the existing system.
In this section we introduce some concepts to make these comparisons more precise. We only consider basic systems here. Consequently, if we like to compare two systems that are for instance described in two different frameworks, we have to find representations on the level of basic systems first.
We sometimes consider several basic systems simultaneously and we will distinguish them by uppercase letters of the beginning alphabet. Their initial states, state space and transition relation are distinguished by subscripts.
We map states of a system A to states of a system B by a function $f \in S_A \rightarrow S_B$. Hence, not all states of system A have to have a correspondent in system B and each state of system B can have zero, one or more correspondents in system A.

For a subset C of the state space of a basic system we define the *C-projected transition relation.*

Definition 1 *C-projected transition relation.*
Let C be a subset of the state space of a basic system $< S, L, T >$ then the C-projected transition relation satisfies:

$$
\begin{aligned}
T^C = \{ < x, y > \in\ & C \times C \mid < x, y > \in T \\
& \vee\ \exists n \in I\!N_1\ :\ \exists v_1, ..., v_n \in S \backslash C\ : \\
& \quad < x, v_1 > \in\ T \wedge\ < v_n, y > \in\ T \\
& \wedge\ \forall i \in \{1, ..., n-1\}\ :< v_i, v_{i+1} > \in\ T \\
\}\ .&
\end{aligned}
$$

C is called the set of *visible states* and T^C is considered as the *visible transition relation.* Note that $T \cap (C \times C) \subseteq T^C$, for all $C \subseteq S$.

The following lemma states that if a given transition relation is first projected to a set C and next to a set $D \subseteq C$, the result is the same if the transition relation was directly projected to D. We omit the proof.

Lemma 1.
Let $< S, L, T >$ be a basic system and let $C \subseteq S$. Then

$$\forall D \subseteq C\ :\ T^D = (T^C)^D\ .$$

Someone from the outside who looks at a basic system with visible states C actually sees another basic system, namely a basic system with states C and transition relation

T^C. This gives rise to a definition of *projection of basic systems.*

Definition 2 *Projection of basic systems.*
Let $A =< S, L, T >$ be a basic system. For any $C \subseteq S$, the *projection of A to C*, $A\restriction C$, will be

$$< C, (L \cap C) \cup \{t \in C \mid \exists s \in L \backslash C \ : < s,t > \in T$$
$$\vee \exists n \in I\!N_1 \ : \ \exists v_1, ..., v_n \in S \backslash C \ :$$
$$< s, v_1 > \in T \wedge \ < v_n, t > \in T$$
$$\wedge \forall i \in \{1, ..., n-1\} \ : < v_i, v_{i+1} > \in T\},$$
$$T^C > \ .$$

Hence, all initial states of $A\restriction C$ either are visible initial states of A or are the first visible state that can be reached when A is started at an invisible initial state.

Now we define the concept of *weak realization.*

Definition 3 *Weak realization.*
A weakly realizes $B$with respect to function f iff

- $f \in S_A \not\rightarrow S_B$;

- $\forall < s,t > \ T_A^{\mathrm{dom}(f)} \ : < f(s), f(t) > \in T_B^{\mathrm{rng}(f)}$.

$\mathrm{dom}(f)$ and $\mathrm{rng}(f)$ represent the visible states of system A and system B, respectively. Hence, every visible state of B has at least one representant in A and A weakly realizes B if every visible transition of A corresponds to a visible transition of B.
The next relationship, called *realization*, is stronger:

Definition 4 *Realization.*
A realizes B with respect to function f iff

- A weakly realizes B with respect to f;

- f is surjective.

Hence, every state of system B should have at least one representant in system A. Note that the surjectivity of f implies $T_B^{\mathrm{rng}(f)} = T_B$.

From Definition 4 it follows that A weakly realizes B w.r.t. f iff A realizes $B\restriction \mathrm{rng}(f)$ w.r.t. f.

The following lemma states that range-projection doesn't destroy the weakly realization-relation.

Lemma 2.
Let A weakly realize B w.r.t. f.
Then for every $R \subseteq \mathrm{rng}(f)$, A weakly realizes B w.r.t. $f \restriction R$.

Proof.
Let $< s,t > \in T_A^{\mathrm{dom}(f \restriction R)}$. We have to show $< f \restriction R(s), f \restriction R(t) > \in T_B^{\mathrm{rng}(f \restriction R)}$.
Note that $s,t \in \mathrm{dom}(f \restriction R)$. So $f \restriction R(s) = f(s)$ and $f \restriction R(t) = f(t)$. Since $\mathrm{dom}(f \restriction R) \subseteq \mathrm{dom}(f)$ we have, by Lemma 1, $< s,t > \in (T_A^{\mathrm{dom}(f)})^{\mathrm{dom}(f \restriction R)}$. There are two cases:

1. $< s,t > \in T_A^{\mathrm{dom}(f)}$. Since A weakly realizes B w.r.t. f, $< f(s), f(t) > \in T_B^{\mathrm{rng}(f)}$.
2. $\exists n \in I\!N \ : \ \exists v_1, ..., v_n \in S_A \backslash \mathrm{dom}(f) \ : < s, v_1 > \in T_A^{\mathrm{dom}(f)} \wedge < v_n, t > \in T_A^{\mathrm{dom}(f)} \wedge \forall i \in \{1..n-1\} \ : < v_i, v_{i+1} > \in T_A^{\mathrm{dom}(f)}$. Since A weakly realizes B w.r.t. f, $< f(s), f(v_1) > \in T_B^{\mathrm{rng}(f)} \wedge \ < f(v_n), f(t) > \in T_B^{\mathrm{rng}(f)} \wedge \forall i \in \{1..n-1\} \ : < f(v_i), f(v_{i+1}) > \in T_B^{\mathrm{rng}(f)}$. For all $i \in \{1, ..., n\} \ : \ v_i \notin \mathrm{dom}(f \restriction R)$ implies $f(v_i) \notin \mathrm{rng}(f \restriction R)$, so by Definition 1, $< f(s), f(t) > \in (T_B^{\mathrm{rng}(f)})^{\mathrm{rng}(f \restriction R)}$.
   Now use Lemma 1 again.

□

Note that A weakly realizes B w.r.t. f does not necessary imply, for $g \subseteq f$, A weakly realizes B w.r.t. g. We give a counterexample:
Let $I_A = S_A = \{a,b,c,d\}$, $T_A = \{< a,b >, < b,c >, < c,d >, < d,a >\}$ and let $I_B = S_B = \{x,y\}$, $T_B = \{< x,y >, < y,x >\}$.
Functions $f, g \in S_A \not\rightarrow S_B$ are defined by $f := \{(a,x),(b,y),(c,x),(d,y)\}$ and $g := \{(a,x),(b,y),(c,x)\}$. Now A weakly realizes B w.r.t. f, $g \subseteq f$ but A does not weakly realize B w.r.t. g.

Suppose A weakly realizes B w.r.t. f and B weakly realizes C w.r.t. g. If g does not introduce new visible states in B, then A weakly realizes C w.r.t. $g \circ f$.

Lemma 3.
Let A weakly realize B w.r.t. function f and let B weakly realize C w.r.t. function g. If $\mathrm{dom}(g) \subseteq \mathrm{rng}(f)$, then A weakly realizes C w.r.t. $g \circ f$.

Proof.
Let $< s,t > \in T_A^{\mathrm{dom}(g \circ f)}$. We have to show $< g \circ f(s), g \circ f(t) > \in T_C^{\mathrm{rng}(g \circ f)}$. Let $f' := f \restriction \mathrm{dom}(g)$. Then $\mathrm{rng}(f') = \mathrm{dom}(g)$ and $g \circ f = g \circ f'$. $\mathrm{Dom}(g \circ f) = \mathrm{dom}(g \circ f') = \mathrm{dom}(f')$, so $< s,t > \in T_A^{\mathrm{dom}(f')}$.
By Lemma 2, $< f'(s), f'(t) > \in T_B^{\mathrm{rng}(f')} = T_B^{\mathrm{dom}(g)}$. Since B weakly realizes C w.r.t. g, $< g \circ f'(s), g \circ f'(t) > \in T_C^{\mathrm{rng}(g)}$. We have $g \circ f' = g \circ f$ and $\mathrm{rng}(g) = \mathrm{rng}(g \circ f)$, so the lemma has been proved.
□

The realization relation over the set of basic systems is transitive:

Lemma 4.
Let A realize B w.r.t. function f and let B realize C w.r.t. function g.
Then A realizes C w.r.t. $g \circ f$.

Proof.
Since f and g are surjective, $\mathrm{dom}(g) \subseteq \mathrm{dom}(f)$ and $g \circ f$ is also surjective. Now use Lemma 3.
□

Lemma 5 indicates the meaning of realization in terms of paths. We omit the proof.

Lemma 5.
Let A realize B w.r.t. function f. For each path σ of system A a path τ of system B exists such that: $\forall i \in \mathrm{dom}(\sigma) \ : \ \sigma_i \in \mathrm{dom}(f) \rightarrow \tau_j = f(\sigma_i)$ where $j = \#\{k \in I\!N_0 \mid k < i \wedge \sigma_k \in \mathrm{dom}(f)\}$.

Note that Lemma 5 can easily be generalized to traces. A very simple example of realization is expressed in Lemma 6:

Lemma 6.
Let A and B be basic systems such that $S_A = S_B$ and $T_A \subseteq T_B$.
Then A realizes B with respect to the identity function.

The proof is trivial.

Note that if system A realizes system B, then every path of system A corresponds to a path of system B, however, system B may have also paths that do not correspond to a path of system A. An example of the realization relation is given in Section 3. The realization relation is used in case system B is a specification of some system and system A is an implementation: In the implementation not all freedom given by the specification, is used.
The next relationship is stronger than realization. It is called *simulation.*

Definition 5 *Simulation.*
A simulates B with respect to function f iff

- A realizes B with respect to f;
- $\forall < x, y > \in T_B \ : \forall \sigma \in \mathrm{dom}(f) \ : \ f(\sigma) = x \rightarrow$

$$\exists \tau \in \mathrm{dom}(f) \ : < \sigma, \tau > \in T_A^{\mathrm{dom}(f)} \wedge f(\tau) = y.$$

Note that for all $x \in S_B$ a $\sigma \in S_A$ exists such that $f(\sigma) = x$, by the surjectivity of f.

The next lemma translates the concept of simulation to paths. We omit the proof.

Lemma 7.
Let A simulate B w.r.t. f.
Then for all paths τ of system B there is a path σ of system A such that

$$\forall i \in \mathrm{dom}(\tau) \; : \exists j \in \mathrm{dom}(\sigma) \; : \; f(\sigma_j) = \tau_i \wedge$$
$$i = \#\{k \in I\!N_0 \mid k < j \wedge \sigma_k \in \mathrm{dom}(f)\} \, .$$

Lemma 7 can easily be generalized to traces.

If system A simulates system B, then there exists also a simulation relation between system A and the basic system derived from B when B is projected to its own visible states. This is expressed in the following lemma:

Lemma 8.
Let A simulate B with respect to function f.
Then for all $R \subseteq \mathrm{rng}(f)$, A simulates $B \restriction R$ w.r.t. $f \restriction R$.

Proof.

We have - f is surjective;
- A weakly realizes B w.r.t. f;
- $\forall < x, y > \in T_B \; : \; \forall \sigma \in \mathrm{dom}(f) \; :$

$$f(\sigma) = x \rightarrow \exists \tau \in \mathrm{dom}(f) \; : \; f(\tau) = y \wedge \; < \sigma, \tau > \in T_A^{\mathrm{dom}(f)} \, . \qquad [*]$$

We have to show

(1) A weakly realizes $B \restriction R$ w.r.t. $f \restriction R$;
(2) $f \restriction R$ is surjective w.r.t. system $B \restriction R$;
(3) $\forall < x, y > \in T_{B \restriction R} \; : \; \forall \sigma \in \mathrm{dom}(f \restriction R) \; :$

$$f \restriction R(\sigma) = x \rightarrow \exists \tau \in \mathrm{dom}(f \restriction R) \; : \; f \restriction R(\tau) = y \wedge \; < \sigma, \tau > \in T_A^{\mathrm{dom}(f \restriction R)} \, .$$

From Lemma 2 it follows that A weakly realizes B w.r.t. $f \restriction R$. Since $T_{B \restriction R} = T_B^R$ we have (1). (2) follows from the surjectivity of f. What remains to prove is (3).
Let $< x, y > \in T_{B \restriction R} = T_B^R$, $\sigma \in \mathrm{dom}(f \restriction R)$ and suppose $f \restriction R(\sigma) = x$. Then also, by definition, $f(\sigma) = x$. We have $< x, y > \in T_B$ or

$$\exists n \in I\!N \; : \exists v_1, ..., v_n \in S_B \backslash R \; : < x, v_1 > \in T_B \wedge$$
$$< v_n, y > \in T_B \wedge \forall i \in \{1..n-1\} \; : < v_i, v_{i+1} > \in T_B \, .$$

From reapplying [*] we get

$$\exists \tau \in \mathrm{dom}(f \restriction R) \; : f(\tau) = y \wedge (< \sigma, \tau > \in T_A^{\mathrm{dom}(f)}$$
$$\vee \; \exists n \in I\!N_1 \; : \; \exists \tau_1, ..., \tau_n \in S_A \backslash \mathrm{dom}(f \restriction R) \; :$$
$$< \sigma, \tau_1 > \in T_A^{\mathrm{dom}(f)} \wedge \; < \tau_n, \tau > \in T_A^{\mathrm{dom}(f)}$$
$$\wedge \; \forall i \in \{1..n-1\} \; : < \tau_i, \tau_{i+1} > \in T_A^{\mathrm{dom}(f)}) \, .$$

(Note that $f(\tau_i) \notin R$ iff $\tau_i \notin \mathrm{dom}(f \restriction R)$.) So by Definition 1,

$$\exists \tau \in \mathrm{dom}(f \restriction R) \ : \ f(\tau) = y \wedge < \sigma, \tau > \in (T_A^{\mathrm{dom}(f)})^{\mathrm{dom}(f \restriction R)} .$$

From Lemma 1 the rest follows.
□

The simulation relation is also *transitive*:

Lemma 9.
Let A simulate B with respect to f and let B simulate C with respect to g.
Then A simulates C with respect to $g \circ f$.

Proof.
From Lemma 4 we know that A realizes C w.r.t. $g \circ f$. What remains to prove is that if $< x, y > \in T_C$ and $\sigma \in \mathrm{dom}(g \circ f)$ such that $g \circ f(\sigma) = x$, a $\tau \in \mathrm{dom}(g \circ f)$ exists such that $< \sigma, \tau > \in T_A^{\mathrm{dom}(g \circ f)}$ and $g \circ f(\tau) = y$. Since $g \circ f(\sigma) = x$ there is a $w \in S_B$ such that $f(\sigma) = w$ and $g(w) = x$. Hence, since B simulates C w.r.t. g, there is a $z \in \mathrm{dom}(g)$ such that $g(z) = y$ and $< w, z > \in T_B^{\mathrm{dom}(g)}$. Let f' be $f \restriction \mathrm{dom}(g)$. Then by Lemma 8, A simulates $B \restriction \mathrm{dom}(g)$ w.r.t. f'. Hence, there is
a $\tau \in \mathrm{dom}(f') \ : < \sigma, \tau > \in T_A^{\mathrm{dom}(f')} \wedge f'(\tau) = z$. Since $g(z) = y$, $g \circ f'(\tau) = y$. By the definition of f', $g \circ f = g \circ f'$ and $\mathrm{dom}(g \circ f) = \mathrm{dom}(f')$, so we have $< \sigma, \tau > \in T_A^{\mathrm{dom}(g \circ f)}$.
□

Suppose system A simulates system B with function f. Assume that we can observe system A by sequences of states, transformed by f. Hence, if a state does not belong to $\mathrm{dom}(f)$ we do not see it, otherwise we see its image under f. If we are also able to observe all states of system B, then we cannot decide from the behaviors which system is A or B. In some sense B and the transformed A are exchangeable. In the next lemma we give a sufficient condition for two systems to simulate each other.

Lemma 10.
Let A simulate B with function f such that: f is injective and f is total.
Then B simulates A with function f^{-1}.

Proof.
Note that $T_A^{\mathrm{dom}(f)} = T_A$ since f is total and that f^{-1} exists and that $T_B^{\mathrm{dom}(f^{-1})} = T_B$. First we show that system B realizes system A with f^{-1}. Let $< x, y > \in T_B$, then we have to prove $< f^{-1}(x), f^{-1}(y) > \in T_A$. Since A simulates B there are $\sigma, \tau \in S_A$ such that $< \sigma, \tau > \in T_A$ and $f(\sigma) = x$ and $f(\tau) = y$. Hence, $< f^{-1}(x), f^{-1}(y) > \in T_A$.
Secondly we show that system B simulates system A. Let $< \sigma, \tau > \in T_A$. Then, since A realizes B, $< f(\sigma), f(\tau) > \in T_B$. Hence, $\forall x \in S_B \ : \ f^{-1}(x) = \sigma \rightarrow \exists y \in S_B \ : \ f^{-1}(y) = \tau \wedge < x, y > \in T_B$.
□

Note that function f must be bijective to satisfy the conditions of Lemma 10. Finally, we define the strongest relationship: *Equivalence*, and we relate it to simulation.

Definition 6 *Equivalence.*
Two basic systems A and B are equivalent if there exists a function $f \in S_A \rightarrow S_B$ such that f is bijective and

$$\forall \sigma, \tau \in S_A \ : < \sigma, \tau > \in T_A \Leftrightarrow < f(\sigma), f(\tau) > \in T_B \ .$$

Notation: $A \cong B$.

The use of the term 'equivalent' is justified by the following property. The proof is trivial.

Lemma 11.
The relationship $\cong$ is an equivalence relation.

Simulation with a bijective function is the same as equivalent.

Lemma 12.
Let A simulate B with function f such that f is bijective.
Then $A \cong B$.

Proof.
Note that $T_A^{\mathrm{dom}(f)} = T_A$. Let $< \sigma, \tau > \in T_A$. By the realization property we have $< f(\sigma), f(\tau) > \in T_B$. Let $< x, y > \in T_B$, then $< f^{-1}(x), f^{-1}(y) > \in T_A$ by the simulation property.
□

We conclude this section with a relationship between a basic system and its *historical variant.* The latter is a basic system the states of which are traces of the original one. We will show that the historical variant simulates the reachability variant of the original one and that the historical variant of the historical variant does not contain new information, i.e. they are equivalent.

Definition 7 *Historical variant.*
For a basic system $< S_A, L_A, T_A >$ with trace set TR, the historical variant $< S_B, L_B, T_B >$ satisfies:

$$S_B = TR;$$
$$L_B = \{\tau \in TR \mid \tau \text{ is a singleton}\};$$
$$T_B = \{< \sigma_0, ..., \sigma_m >, < \tau_0, ..., \tau_n > \in TR \times TR \mid$$
$$n = m + 1 \wedge \forall i \in \{1..m\} \ : \ \sigma_i = \tau_i\} \ .$$

The historical variant simulates the reachability variant of the original system:

Lemma 13.
Let A be a basic system with reachability variant $\tilde{A}$ and historical variant B.
Then B simulates $\tilde{A}$ with function f defined by $\forall n \in I\!N_0 \ : \ \forall < \sigma_0, ..., \sigma_n > \in S_B \ : \ f(< \sigma_0, ..., \sigma_n >) = \sigma_n$.

Proof.
Note that f is total and, by the definition of the reachability variant, f is surjective. Hence, $T_B^{\mathrm{dom}(f)} = T_B$ and $T_{\tilde{A}}^{\mathrm{rng}(f)} = T_{\tilde{A}} = T_A$.
Let $< \sigma, \tau > \in T_B$. Then by the definition of T_B, $< f(\sigma), f(\tau) > \in T_A = T_{\tilde{A}}$. So B realizes $\tilde{A}$ w.r.t. f.
Let $< x, y > \in T_{\tilde{A}}$ and $< \sigma_0, ..., \sigma_n > \in S_B$ such that $f(< \sigma_0, ..., \sigma_n >) = x$, then $\sigma_n = x$ and $<< \sigma_0, ..., \sigma_{n-1}, x >, < \sigma_0, ..., \sigma_{n-1}, x, y >> \in T_B$ and $f(< \sigma_0, ..., \sigma_{n-1}, x, y >) = y$.
□

Lemma 14.
Let A be a basic system, B its historical variant and let C be the historical variant of B.
Then $B \cong C$.

Proof.
Consider the same f as in Lemma 13:

$$f \in S_C \to S_B \text{ and}$$

$$\forall n \in \mathbb{N}_0 : \forall << \sigma_0 >, < \sigma_0, \sigma_1 >, ..., < \sigma_0, ..., \sigma_n >> \in S_C :$$

$$f(<< \sigma_0 >, < \sigma_0, \sigma_1 >, ..., < \sigma_0, ..., \sigma_n >>) =< \sigma_0, ..., \sigma_n > .$$

This function is total, surjective but also injective. Furthermore, the reachability variant of B is B itself. Hence, by Lemma 13 and Lemma 12, $B \cong C$.
□

We did not consider the initial states in our definitions of realization and simulation. We will confine ourselves to remarking that it is obvious to require that the function which maps states of a basic system to states of another basic system is such that initial states correspond only to initial states.
In the next section we will introduce our framework, called des, to model discrete event systems.

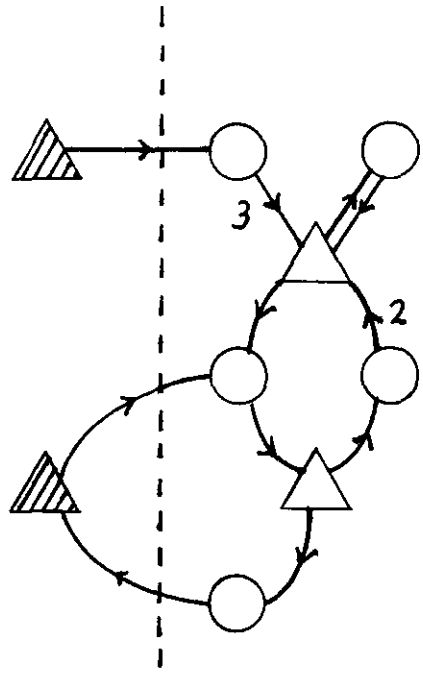
3. Model of discrete event systems

In this section we introduce a mathematical model for discrete event systems, called the des-model. It is an extension of the Petri net-model. At the end of this section we summarize some differences. We start with an informal treatment.

A des consists of two kinds of components: *Processors* and *channels*, which correspond to transitions and places in Petri nets. A processor is connected to 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 the input parameters are identical to the types of the input channels and the types of the output parameters of the function correspond to the types of the output channels. A channel may be shared by several processors as input or output channel. At each moment the channels may contain so-called *triggers* (tokens in Petri nets). A trigger has a value that belongs to the type of the channel. There may be more than one trigger of the same value in a channel. So a channel contains a bag over its type (cf. notations).
For each processor, every input channel has a multiplicity, which gives the number of triggers the processor needs from that channel to operate. At each moment there may be a *transition*, which means that the configuration of triggers in the channels may change. Such a transition happens instantaneously and is *executed* by the processors. Each processor that has enough triggers in every input channel may pick as many triggers as it needs for each input channel and produce a finite amount of new triggers for its output channels, according to its function. Several processors may produce triggers for the same output channel. A channel in which always exactly one trigger is present 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 trigger and instantaneously places back a new trigger into the store. In fact, it replaces the trigger. A store can even be a database. In this case its type is very complex. A configuration of triggers distributed over channels is called a *state*.

The systems we consider may be part of another, much larger system. In this case communications take place from our system to its environment and vice versa. We regard the environment as a des too, i.e. as a system with processors and channels, but we do not know the specification of those processors and the channel structure. Processors with an unknown specification are called *black box-processors*. A black box-processor which only produces triggers for our well-known system may be modeled as a single channel containing an infinite amount of triggers. Such a channel represents an *input stream*. It will never get empty, since a processor consumes only a finite amount of triggers in a finite time interval.
To indicate the processors and channels and their input- and output relations, we use a diagram technique where processors are represented by triangles and channels by circles. To indicate input/output relations we use arrows and for each input channel we mention its multiplicity, except for input channels with multiplicity one, which is the default value.

Environment



Well-known system

Figure 3.1.

Now we formalize the des-framework. We start with some additional notations.

Notations

The symbol 'ω' stands for 'infinite'. $\omega + \omega = \omega$ and for any $n \in \mathbb{N}_0$, $\omega + n = \omega$ and $\omega - n = \omega$. If X is a set of numbers, then $X^\omega = X \cup \{\omega\}$. If Y is a set of sets, then $\bigcup Y$ denotes the union of all elements of Y. If A is a set, then $\mathbb{P}(A)$ denotes the set of all finite subsets of A and $\mathbb{B}(A)$ denotes the set of all multisets (bags) over A, i.e. the set of all functions from A to $\mathbb{N}_0^\omega$. Note that infinite many copies of the same element can appear in a bag and that a bag can contain infinite many different elements. For $x \in \mathbb{B}(A)$ and b some element: $b \in x$ iff $b \in A$ and $x(b) > 0$. For s a set and $x \in \mathbb{B}(A)$:

- $s \subseteq x$ iff $\forall a \in s : a \in x$.
- $x \subseteq s$ iff $\forall a \in A : x(a) = 0 \vee (x(a) = 1 \wedge a \in s)$.
- $x = s$ iff $x \subseteq s \wedge s \subseteq x$.

So we can compare sets and multisets.

- $x \cup s = \lambda a \in A \cup s :$ if $a \in A \backslash s$ then $x(a)$ else if $a \in s \backslash A$ then 1 else $x(a) + 1$ fi fi.
- $x \backslash s = \lambda a \in A :$ if $a \in s$ then $\max(0, x(a) - 1)$ else $x(a)$ fi.
- $s \backslash x = \{a \in s \mid a \notin x\}$.
- $x \cap s = x \backslash (x \backslash s)$.

For $x \in \mathbb{B}(A)$ and $y \in \mathbb{B}(B)$, where B is also a set:

- $x \subseteq y$ iff $\forall a \in x : a \in B \wedge x(a) \leq y(a)$.

- $x = y$ iff $x \subseteq y \wedge y \subseteq x$.
- $x \cup y = \lambda a \in A \cup B$: if $a \in A \backslash B$ then $x(a)$
  else if $a \in B \backslash A$ then $y(a)$
  else $x(a) + y(a)$ fi fi.
- $x \backslash y = \lambda a \in A$: if $a \in B$ then $\max(0, x(a) - y(a))$
  else $x(a)$ fi.
- $x \cap y = x \backslash (x \backslash y)$.

For a bag-valued function f with finite domain $A = \{a_1, ..., a_n\}$, mrng(f) denotes the multisetrange of f, defined as mrng(f) $= f(a_1) \cup ... \cup f(a_n)$. We also write $\bigcup_{a \in A} f(a)$ for mrng(f).
If x is a bag over A, then $\# x$ is the number of elements in x, i.e. $\# x = \sum a \in A : x(a)$. x is *infinite* iff $\# x = \omega$, otherwise x is *finite*. $\mathbb{B}_n(A)$ denotes the set of all bags over A with exactly n elements, i.e. $\mathbb{B}_n(A) = \{b \in \mathbb{B}(A) \mid \# b = n\}$. For an ordered set D and $d_1, d_2 \in D$: d_1 min d_2 is the minimum of d_1 and d_2 and d_1 max d_2 is the maximum of d_1 and d_2.

Definition 1 *Discrete event system.*
A discrete event system (des) is a four-tuple $< R, C, I, O >$ where R is a function-valued function, C and O are set-valued functions and I is a bag-valued function, such that:

- dom(I) = dom(O) = dom(R), finite or countable sets.
- $\forall i \in \text{dom}(R)$: $I_i \in \mathbb{B}(\text{dom}(C)) \wedge I_i \neq \emptyset \wedge I_i$ is finite
  $\wedge\ O_i \subseteq \text{dom}(C)$.
- $\forall i \in \text{dom}(R)$: $R_i \in \bigcup_{k \in \text{dom}(C)} \mathbb{B}_{I_i(k)}(\{< c, w > \mid c = k \wedge w \in C_c\})$
  $\rightarrow \mathbb{B}(\{< c, w > \mid c \in O_i \wedge w \in C_c\})$.
- $\forall i \in \text{dom}(C)$: C_i is finite or countable.

dom(R) is called the set of *processor indices*, denoted by P,
dom(C) is called the set of *channel indices*, denoted by K
and for all $i \in P$, $k \in K$:

- I_i is called the bag of input channels of i, where
  $I_i(k)$ is the multiplicity of channel k.
- O_i is called the set of output channels of i.
- R_i is called the reaction function of i.
- C_k is called the type of channel k.

We will use these symbols strictly for the concepts defined. If we consider different des'ses we distinguish them by means of subscripts.

Definition 2 *Trigger set, state space, event set.*
Let a des be given. Then

$$Q := \{< k, w > \mid k \in K \wedge w \in C_k\}$$
$$S := I\!B(Q)$$
$$E \subset P \not\to I\!B(Q) \text{ such that } \forall e \in E : \mathrm{dom}(e) \neq \emptyset$$
$$\text{and } \forall p \in \mathrm{dom}(e) : e_p \in \mathrm{dom}(R_p)\,.$$

Q is called the *trigger set*, S is called the *state space* and E is called the *event set.*

Note that an event is an assignment of a bag of triggers to a processor such that for each input channel k with multiplicity m exactly m triggers are chosen.

Definition3 *Event function.*
The event function F of a des satisfies:

$$F \in S \to I\!P(E) \text{ and}$$
$$\forall s \in S : F(s) = \{e \in E \mid mrng(e) \subseteq s\}\,.$$

Hence $e \in F(s)$ only if s contains enough triggers to supply all the processors of $\mathrm{dom}(e)$. Note that $e \in F(\mathrm{mrng}(e))$, for all events $e \in E$. It is easy to verify that $\forall s, t \in S : s \subseteq t \to F(s) \subseteq F(t)$.
Next we define the transition function, which assigns to a state s and an event $e \in F(s)$ a new state.

Definition 4 *Transition function, transition relation.*
The transition function T of a des satsifies:

$$T \in S \times E \not\to S \text{ such that}$$
$$\forall s \in S : \forall e \in F(s) : < s, e > \in \mathrm{dom}(T) \wedge$$
$$T(s,e) = s \backslash \mathrm{mrng}(e) \cup \bigcup_{p \in \mathrm{dom}(e)} R_p(e_p)\,.$$

The transition relation of a des is:

$$\{< s, t > \in S \times S \mid \exists e \in F(s) : T(s,e) = t\}\,.$$

Elements of the transition relation are called *transitions.*

We also use the symbol T to denote the transition relation. It is easy to verify that the graph $< S, L, T >$ where L is a collection initial states, S the state space and T the transition relation of a des, forms a basic system. If we speak of path, trace or process of a des we mean the path, trace or process of the basic system induced by the des.
The next lemma states that if two events change a state in the same way, then they also change any state containing more triggers in the same way.

Lemma 1.
Let e_1 and e_2 be events and σ and τ be states, such that $e_1, e_2 \in F(\sigma)$ and $\sigma \subseteq \tau$. Then $e_1, e_2 \in F(\tau)$ and

$$T(\sigma, e_1) = T(\sigma, e_2) \text{ iff } T(\tau, e_1) = T(\tau, e_2) .$$

Proof.
That $e_1, e_2 \in F(\tau)$ is trivial.
Since $e_1, e_2 \in F(\sigma)$ we have, by Definition 3, $\text{mrng}(e_i) \subseteq \sigma$ $(i \in \{1,2\})$. Let $x = \tau \backslash \sigma$. Then $T(\tau, e_i) = x \cup T(\sigma, e_i)$ by Definition 4 and $T(\sigma, e_i) = T(\tau, e_i) \backslash x$, since

$$((\sigma \cup x) \backslash \text{mrng}(e)) \backslash x = \sigma \backslash \text{mrng}(e) .$$

□

Note that $T(\sigma, e_1) = T(\sigma, e_2)$ does not imply that $e_1 = e_2$: A transition may be caused by different events. We give an example:

Example.
Consider two processors p and q with input channels $a_1, ..., a_m$ for p and $b_1, ..., b_n$ for q. Let further the input channels be also output channels and let p and q have some common output channels $c_1, ..., c_l$ such that all channels are different. Choose the reactor function of p and q such that for every $t_1 \in \text{dom}(R_p)$ and $t_2 \in \text{dom}(R_q)$: $R_p(t_1) = t_1 \cup G$ and $R_q(t_2) = t_2 \cup G$, where G is some bag over $\{< k, w > \mid k \in \{c_1, ..., c_l\} \wedge w \in C_k\}$. In Figure 3.2, $m = 2$ and $n = l = 1$.

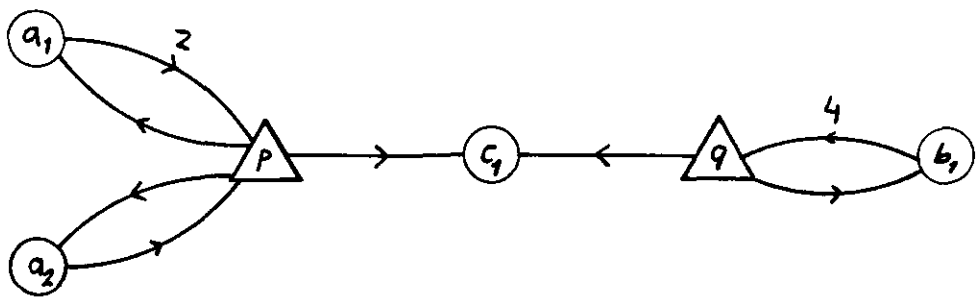


Figure 3.2.

Suppose we have two events e_1 and e_2 which are equal except for triggers assigned to p and q, such that e_1 triggers p but does not trigger q (i.e. $p \in \text{dom}(e_1)$ and $q \notin \text{dom}(e_1)$) and e_2 triggers q but does not trigger p. Furthermore, suppose we have a state σ such that both e_1 and e_2 can be executed in σ. Then it is obvious that both transitions are equal: $T(\sigma, e_1) = T(\sigma, e_2)$.

It is usefull to introduce an equivalence relation on the set of events. Intuitively two events are equivalent if their effects are indistinguishable.

Definition 5 *Equivalence of events.*
Two events e_1, e_2 are equivalent (notation: $e_1 \sim e_2$) iff

$$\exists \sigma \in S \ : \ e_1, e_2 \in F(\sigma) \wedge T(\sigma, e_1) = T(\sigma, e_2) \ .$$

The next lemma justifies the use of the term 'equivalent'.

Lemma 2.
The relation $\sim$ over E is an equivalence relation.

Proof.
Reflexivity and symmetry are trivial, so transitivity remains. Let $e_1 \sim e_2$ and $e_2 \sim e_3$. Then there are states σ and τ such that $e_1, e_2 \in F(\sigma)$ and $e_2, e_3 \in F(\tau)$ and $T(\sigma, e_1) = T(\sigma, e_2)$ and $T(\tau, e_2) = T(\tau, e_3)$. By Lemma 1 we have $T(\sigma \cup \tau, e_1) = T(\sigma \cup \tau, e_2) = T(\sigma \cup \tau, e_3)$, so $e_1 \sim e_3$.
□

We are able to formulate a criterion to verify if two events are equivalent.

Lemma 3.
Let e_1 and e_2 be events. Then

$$T(\mathrm{mrng}(e_1) \cup \mathrm{mrng}(e_2), e_1) = T(\mathrm{mrng}(e_1) \cup \mathrm{mrng}(e_2), e_2) \ \text{ iff } \ e_1 \sim e_2 \ .$$

Proof.
The only-if-part is trivial.
Suppose $e_1 \sim e_2$. Then there is a state τ such that $e_1, e_2 \in F(\tau)$ and $T(\tau, e_1) = T(\tau, e_2)$. Then $\mathrm{mrng}(e_i) \subseteq \tau$, $i \in \{1, 2\}$, hence $\mathrm{mrng}(e_1) \cup \mathrm{mrng}(e_2) \subseteq \tau$. Call $\sigma = \mathrm{mrng}(e_1) \cup \mathrm{mrng}(e_2)$. Since $e_i \in F(\mathrm{mrng}(e_i))$, we have by Lemma 1, $T(\sigma, e_1) = T(\sigma, e_2)$.
□

The next corollary is a direct consequence of Lemma 1 and Lemma 3:

Corollary 4.
Let e_1 and e_2 be events. Then

$$\begin{aligned} e_1 \sim e_2 \ \text{ iff } \ \forall \sigma \in S \ &: \mathrm{mrng}(e_1) \cup \mathrm{mrng}(e_2) \subseteq \sigma \\ &\rightarrow T(\sigma, e_1) = T(\sigma, e_2) \ . \end{aligned}$$

Note that we have true parallellism in our model: Processors may execute simultaneously. However, it is always possible to split up an event which triggers more than one processor into other events, such that the successive execution of these events, in any order, ends in the same state as the original compound event. Especially, we are able to split up any event into other events which trigger exactly one processor. This property is a consequence of the following theorem.

Theorem 5.
Let a des be given and let $s \in S$ and $e \in F(s)$ such that $|\mathrm{dom}(e)| \geq 2$. Let further

$D_1, D_2 \subset \mathrm{dom}(e)$ such that $|D_1| \geq 1$, $|D_2| \geq 1$, $D_1 \cap D_2 = \emptyset$ and $D_1 \cup D_2 = \mathrm{dom}(e)$. Then $e\lceil D_1 \in F(s)$, $e\lceil D_2 \in F(T(s, e\lceil D_1))$ and $T(T(s, e\lceil D_1), e\lceil D_2) = T(s, e)$.

Proof.
By Definition 2 and Definition 3, $e\lceil D_1 \in E$ and $e\lceil D_1 \in F(s)$. Furthermore, $T(s, e\lceil D_1) = s \backslash \mathrm{mrng}(e\lceil D_1) \cup \bigcup_{p \in D_1} R_p(e_p)$. Since $\mathrm{mrng}(e\lceil D_1) \cup \mathrm{mrng}(e\lceil D_2) = \mathrm{mrng}(e) \subseteq s$, also $\mathrm{mrng}(e\lceil D_2) \subseteq s \backslash \mathrm{mrng}(e\lceil D_1)$. Hence, by Definition 3, $e\lceil D_2 \in F(s \backslash \mathrm{mrng}(e\lceil D_1))$ and by Lemma 1, $e\lceil D_2 \in F((s \backslash \mathrm{mrng}(e\lceil D_1)) \cup \bigcup_{p \in D_1} R_p(e_p)) = F(T(s, e\lceil D_1))$. By Definition 4,

$$
\begin{aligned}
& T(T(s, e\lceil D_1), e\lceil D_2)) = \\
& (((s \backslash \mathrm{mrng}(e\lceil D_1)) \cup \bigcup_{p \in D_1} R_p(e_p)) \backslash \mathrm{mrng}(e\lceil D_2)) \cup \bigcup_{p \in D_2} R_p(e_p) = \\
& (s \backslash (\mathrm{mrng}(e\lceil D_1) \cup \mathrm{mrng}(e\lceil D_2))) \cup \bigcup_{p \in D_1 \cup D_2} R_p(e_p) = \\
& (s \backslash \mathrm{mrng}(e)) \cup \bigcup_{p \in \mathrm{dom}(e)} R_p(e_p) = T(s, e) \; .
\end{aligned}
$$

□

A special case of Theorem 5 is when D_1 is a singleton. By repeated application of Theorem 5 with D_1 being a singleton all the time we may split up an event into a sequence of events where each event triggers exactly one processor and the order of triggering is irrelevant. This property is called *trigger serializability*. It gives some freedom for implementation of des'ses. If we restrict ourselves to transitions with exactly one triggered processor, then we obtain a basic system that not only realizes the original one, but also has an even stronger relationship: The transitive closu of their transition relations are equal.

Lemma 6.
Let $< R, C, I, O >$ be a des with basic system $< L, S, T >$, so L is the set of initial states. Consider the event function $\tilde{F}$ derived from F by:

$$\forall \sigma \in S \; : \; \tilde{F}(\sigma) = \{e \in F(\sigma) \mid |\mathrm{dom}(e)| = 1\} \; .$$

Let $\tilde{T} = \{< \sigma, \tau > \in S \times S \mid \exists e \in \tilde{F}(\sigma) \; : \; T(\sigma, e) = \tau\}$. Then

- $< L, S, \tilde{T} >$ realizes $< L, S, T >$ with the identity function.
- The transitive closu of $\tilde{T}$ and T are equal.

Proof.
The first assertion immediately follows from Section 2, Lemma 6. For the second assertion, note that $\tilde{T} \subseteq T$ implies $\tilde{T}^* \subseteq T^*$. To prove $T^* \subseteq \tilde{T}^*$, it satisfies to show $T \subseteq \tilde{T}^*$. Assume $< \sigma, \tau > \in T$ and let $e \in F(\sigma)$ satisfy $T(\sigma, e) = \tau$. Let

$\{p_1, ..., p_m\} = \text{dom}(e)$. We define events $e_1, ..., e_m$ by $e_i = \{(p_i, e(p_i)\}$ and states v_i by $v_i = T(...T(T(\sigma, e_1), e_2), ..., e_i)$, $i \in \{1, ..., m\}$.
Then $< \sigma, v_1 > \in \tilde{T}$ and for all $i \in \{1, ..., m-1\}$: $< v_i, v_{i+1} > \in \tilde{T}$ and since we have trigger serializability, $v_m = \tau$. Hence, $< \sigma, \tau > \in \tilde{T}^*$.
□

In our model many basic ideas of Petri nets can be found. For example, a transition can fire only if all its input places have enough tokens. But our model has some featu which are not common in Petri nets. Recall that our tokens are called triggers and that our places are called channels. Now we summarize some differences between our model and the Petri net model [Peterson 81]:

- Triggers have values and channels have types;
- A channel may contain infinitely many triggers;
- By firing, a processor takes a predefined number of triggers from each input channel and produces a non-predefined number of triggers for each output channel;
- The kind and number of produced output triggers depend on the kind of input triggers;
- Processors may fire simulataneously.

Upto now we did not consider time aspects, we had only order of events. In the next section we introduce another framework, called rtdes, to model real-time aspects of discrete event systems, too. This framework is closely related to des, it has more structure and some extra restrictions on events. In a des each triggers has a value and in a rtdes each trigger has a timestamp in addition.

4. Real-time model

In this section we introduce a new model for discrete event systems, in which time plays an essential role. This model is closely related to des, in fact each system that fits into this model, also fits into the des-model. We call the model rtdes because it describes a possible real-time behavior of a des. We start with an informal description.

Consider a des in which each trigger has, beside its value, a *time stamp.* This time stamp denotes the arrival time of the trigger and is interpreted as the earliest time the trigger may be used. When a processor performs an action, it consumes several triggers and produces several new triggers. Now such an action has an *action time*, which is the maximum time stamp of all consumed triggers. We require that the time stamp of all produced triggers is at least as large as the action time. Processors are eager to execute, which means that as soon as all input channels have enough triggers and no processor is able to perform an action at an earlier time, it performs its task. Though we do not have any absolute clock in our model, each processor can easily compute the action time by simply taking the maximum value of all time stamps of its input triggers. By means of an input stream an rtdes can be initialized with infinitely many triggers. However, since each trigger has a time stamp, we are also able to determine its arrival time. Hence, it is possible to initialize a system with some triggers which will arrive in the future.

Definition 1 *Real-time discrete event system (rtdes).*
An rtdes is a des $< R, C, I, O >$ with the properties:

- A set-valued function V and an ordered set D exist such that $\mathrm{dom}(V) = K$ and $\forall k \in K \ : \ C_k = V_k \times D$.
- $\forall p \in P \ : \ \forall b \in \mathrm{dom}(R_p) \ : \ \forall x \in b \ : \ \forall y \in R_p(b) \ : \ \mathrm{time}(x) \leq \mathrm{time}(y)$, where, for any $k \in K$, $v \in V_k$ and $d \in D$, $\mathrm{time}(< k, < v, d >>) = d$.

Hence, the time stamps of the produced triggers are at least as large as the time stamps of the consumed triggers. The state space, trigger set, event set and event function of an rtdes are the same as of the corresponding des. It is usual to take $D \subseteq \mathbb{R}^+$, the positive real numbers. If we consider clocked systems, the set D is a set of equidistant points in time, i.e. $D = \{n \cdot \delta \mid n \in \mathbb{N}_1\}$ for some $\delta \in \mathbb{R}^+$.
We assign to each event e an *event time* $h(e)$, which is the maximum time stamp of each trigger in $\mathrm{mrng}(e)$.

Definition 2 *Event time.*
For an rtdes the function h assigns to each event an event time, such that for $e \in E$:

$$h(e) = \max\{\mathrm{time}(x) \mid x \in \mathrm{mrng}(e)\} \ .$$

The events that may be used in a transition are more restricted than in a des, since an event may happen only if no other event can happen at an earlier time. Consequently, for each state we may define a transition time: It is the time the system will leave the state.

Definition 3 *Transition time.*
For an rtdes the function H assigns to each state a transition time, such that for $s \in S$:

$$H(s) = \min\{h(e) \mid e \in F(s)\} .$$

Now we are able to define the event function for an rtdes. It is called the real-time event function.

Definition 4 *Real-time event function.*
The real-time event function FT of an rtdes satisfies:

$$FT \in S \rightarrow I\!P(E)$$

and

$$\forall s \in S \ : \ FT(s) = \{e \in E \mid e \in F(s) \wedge h(e) = H(s)\} .$$

This expresses that only events with lowest event time are allowable. We have parallellism in an rtdes, however, several processors may execute simultaneously only if their action times are equal. This is expressed by the following lemma.

Lemma 1.
Let s be a state of an rtdes and $e \in FT(s)$.
Then $\forall p \in \mathrm{dom}(e) \ : \ h(e\lceil\{p\}) = H(s)$.

Proof.
Let $\{p_1, ..., p_n\} = \mathrm{dom}(e)$. By Definition 3, $h(e \lceil \{p\}) \geq H(s)$.
Assume, to get a contradiction, $h(e\lceil\{p\}) > H(s)$. Then, by Definition 4 and Definition 2,

$$H(s) = h(e) = h(e\lceil\{p_1\}) \,\underline{\max}\, ... \,\underline{\max}\, h(e\lceil\{p_n\}) \geq h(\lceil\{p\}) > H(s) ,$$

which is a contradiction.
□

Now we define the real-time transition function and the real-time transition relation.

Definition 5 *Real-time transition function, real-time transition relation.*
The real-time transition function TR of an rtdes satisfies: $TR \in S \times E \not\to S$ such that

$$\forall s \in S \ : \forall e \in FT(s) \ : < s, e > \in \mathrm{dom}(TR) \wedge$$
$$TR(s,e) = s \backslash \mathrm{mrng}(e) \cup \bigcup_{p \in \mathrm{dom}(e)} R_p(e_p) \ .$$

The real-time transition relation of an rtdes is:

$$\{< s,t > \in S \times S \mid \exists e \in FT(s) \ : \ TR(s,e) = t\} \ .$$

We also use the symbols TR to denote the real-time transition relation. If L is a collection initial states of an rtdes, then the graph $< S, L, TR >$ forms a basic system. Note that the only difference between the real-time transition relation TR of an rtdes and the transition relation T of the corresponding des is the requirement that the event should be an element of $FT(s)$ instead of $F(s)$. Hence, $TR \subseteq T$. Consequently, the process of the rtdes is a subset of the process of the des and the rtdes realizes the des with the identity function.

A major feature of an rtdes is that the transition times of successive states on a path are ascending. First we introduce two lemmas which are helpful in proving this property. Lemma 2 states that the transition time cannot decrease when several triggers are deleted from a state.

Lemma 2.
Let s and t be states of an rtdes.
If $s \subseteq t$, then $H(s) \geq H(t)$.

Proof.
Note that $s \subseteq t$ implies $F(s) \subseteq F(t)$. By Definition3,

$$H(s) = \ \min\{h(e) \mid e \in F(s)\} \geq \min\{h(e) \mid e \in F(s) \cup (F(t) \backslash F(s))\} \ =$$
$$= \ \min\{h(e) \mid e \in F(t)\} = H(t) \ .$$

□

The transition time does not change when we add to a state several triggers which have a time stamp that is at least as large as the current transition time. This is expressed in Lemma 3.

Lemma 3.
Let s and t be states of an rtdes.
If $\forall x \in t \ : \ \mathrm{time}(x) \geq H(s)$, then $H(s) = H(s \cup t)$.

Proof.
Note that $F(s) \subseteq F(s \cup t)$. We show $\forall e \in F(s \cup t) \backslash F(s) \ : \ h(e) \geq H(s)$. Let $e \in$

$F(s \cup t) \backslash F(s)$. Then $\text{mrng}(e) \subseteq s \cup t$ and $\neg(\text{mrng}(e) \subseteq s)$. Hence, $\exists x \in \text{mrng}(e) : x \in t \wedge x \notin s$. Since $x \in t$, $\text{time}(x) \geq H(s)$ and by Definition 2, $h(e) \geq H(s)$.
Hence, $H(s) \leq \min\{h(e) \mid e \in F(s \cup t) \backslash F(s)\}$. Consequently,

$$\begin{aligned} H(s) &= H(s) \underline{\min} \ \min\{h(e) \mid e \in F(s \cup t) \backslash F(s)\} \\ &= \min\{h(e) \mid e \in F(s)\} \underline{\min} \ \min\{h(e) \mid e \in F(s \cup t) \backslash F(s)\} \\ &= \min\{h(e) \mid e \in F(s \cup t)\} = H(s \cup t) \ . \end{aligned}$$

□

It is a direct consequence of the following theorem that the transition times of successive states on a path are ascending.

Theorem 4.
For an rtdes we have

$$\forall < s,t > \in TR : H(s) \leq H(t) \ .$$

Proof.
$< s,t > \in TR$ implies $\exists e \in FT(s) : t = s \backslash \text{mrng}(e) \cup \bigcup_{p \in \text{dom}(e)} R_p(e_p)$. Let $r = \bigcup_{p \in \text{dom}(e)} R_p(e_p)$.
By Lemma 1, the action times of all processors in $\text{dom}(e)$ is equal to $H(s)$. By the definition of the reaction function (Definition 1) all produced triggers must have a time stamp that is at least as large as $H(s)$, so r satisfies the condition of Lemma 3. Hence, $H(s) = H(s \cup r)$. By Lemma 2, $H(s \cup r) \leq H(s \backslash \text{mrng}(e) \cup r) = H(t)$.
□

Note that it might happen that the transition time remains constant after a while. Three possible causes for this are: The system may be initialized with infinitely many triggers having a time stamp not larger than a certain value, a processor may produce infinitely many triggers or a processor may produce triggers which have a time stamp equal to the action time. The following theorem gives sufficient conditions such that the number of transitions in each finite time interval is finite.

Theorem 5.
Let an rtdes be given with additional properties: There is an $\epsilon > 0$ such that

$$\forall p \in P : \forall b \in \text{dom}(R_p) : \forall x \in b : \forall y \in R_p(b) : \text{time}(y) - \text{time}(x) \geq \epsilon$$

and $R_p(b)$ is finite.
Let also a collection initial states L be given such that

$$\forall l \in L : \forall t \in D : \{x \in l \mid \text{time}(x) \leq t\} \text{ is finite} \ .$$

Then for all paths σ and all $t \in D$:

$$\{i \in I\!N \mid H(\sigma_i) \leq t\} \text{ is finite .}$$

Proof.
We prove the property for $t = n \cdot \epsilon + H(\sigma_0)$, $n \in I\!N_0$, using induction. For $n = 0$ the property holds. Assume it holds for $t = n\epsilon$. We show that the number of triggers with time stamp in $TI = (H(\sigma_0) + n\epsilon, H(\sigma_0) + (n+1)\epsilon]$ is finite. By the induction hypothesis, there are only finitely many transitions with transition time smaller or equal to $H(\sigma_0) + n\epsilon$ and the number of produced triggers per transition is finite. Furthermore, the initial state σ_0 is such that only finitely many triggers have an arrival time in time-interval TI. Transitions in TI do not produce triggers with time stamps within TI because of the delay ϵ. Transitions after $H(\sigma_0) + (n+1)\epsilon$ do not produce such triggers either, according to Theorem 4. Hence, the number of transitions in TI must be finite, too. Consequently, the number of transitions in $[H(\sigma_0), H(\sigma_0) + (n+1)\epsilon]$ is finite.
□

Suppose we specify a des and we implement it as an rtdes. In this rtdes, the values of produced triggers are independent of the time stamps of the consumed triggers. Furthermore, the time stamp of each produced trigger is computed by means of a delay depending only on the value of the produced trigger and the values of the consumed triggers. This delay is added to the action time to obtain the time stamp of the produced trigger. We will show that this rtdes realizes the original des.

Theorem 6.
Let a des $< R, C, I, O >$ with basic system $< S, L, T >$ be given.
We define a real-time state space $\tilde{S} := I\!B(\{< k, < v, d >> \mid k \in K \wedge v \in C_k \wedge d \in I\!R^+\})$ and a function $f : \tilde{S} \rightarrow S$ with

$$f(\tilde{s}) = \{<< k, v >, n > \mid k \in K \wedge v \in C_k \wedge n = \sum d \in I\!R^+ \ : \ \tilde{s}(< k, < v, d >>)\} .$$

Let also a delay-function dl be given such that

$$\text{dom}(dl) = P$$

and

$$\forall p \in P \ : \ dl_p \in \text{dom}(R_p) \rightarrow \tilde{S}$$

and

$$\forall b \in \text{dom}(R_p) \ : \ f(dl_p(b)) = R_p(b)$$

and

$$\forall x \in dl_p(b) \ : \ \text{time}(x) > 0 .$$

Now we define the rtdes $< \tilde{R}, \tilde{C}, \tilde{I}, \tilde{O} >$ with basic system $< \tilde{S}, \tilde{L}, TR >$ as follows:

$$
\begin{aligned}
\tilde{L} &= \{<< k, < v, d >>, n > \mid k \in K \wedge v \in C_k \wedge d \in \mathbb{R}^+ \wedge n \in \mathbb{N}_0 \wedge << k, v >, n > \in L\}, \\
\tilde{I} &= I, \\
\tilde{O} &= O, \\
\tilde{C} &= \lambda k \in K : C_k \times \mathbb{R}^+ \text{ and for } p \in P,\ \tilde{b} \in \mathrm{dom}(\tilde{R}_p) : \\
\tilde{R}_p(\tilde{b}) &= \{<< k, < v, d + H(\tilde{b}) >>, n > \mid k \in K \wedge v \in C_k \wedge d \in \mathbb{R}^+ \wedge n \in \mathbb{N}_0 \\
&\qquad \wedge << k, < v, d >>, n > \in dl_p(f(\tilde{b}))\}.
\end{aligned}
$$

(Note that $\mathrm{dom}(\tilde{R}_p)$ is defined as a function from $\tilde{I}$ and $\tilde{C}$.)
Then $< \tilde{S}, \tilde{L}, TR >$ realizes $< S, L, T >$ with respect to f.

Proof.
From the deifnition of f and $\tilde{R}$ it follows that for $\tilde{b}, \tilde{c} \in \tilde{S}$, $f(\tilde{b} \cup \tilde{c}) = f(\tilde{b}) \cup f(\tilde{c})$, $f(\tilde{b} \backslash \tilde{c}) = f(\tilde{b}) \backslash f(\tilde{c})$ and for $p \in P$, $f(\tilde{R}_p(\tilde{b})) = R_p(f(\tilde{b}))$.
f is surjective, but also total, hence, $TR^{\mathrm{dom}(f)} = TR$ and $T^{\mathrm{rng}(f)} = T$. Suppose $< \tilde{s}, \tilde{t} > \in TR$. Then

$$
\exists \tilde{e} \in FT(\tilde{s}) : \tilde{t} = \tilde{s} \backslash \mathrm{mrng}(\tilde{e}) \cup \bigcup_{p \in \mathrm{dom}(\tilde{e})} \tilde{R}_p(\tilde{e}_p).
$$

We define $e := \lambda p \in \mathrm{dom}(\tilde{e}) : f(\tilde{e}_p)$. Then

$$
\begin{aligned}
f(\tilde{t}) &= f(\tilde{s}) \backslash f(\mathrm{mrng}(\tilde{e})) \cup \bigcup_{p \in \mathrm{dom}(e)} f(\tilde{R}_p(\tilde{e}_p)) \\
&= f(\tilde{s}) \backslash \mathrm{mrng}(e) \cup \bigcup_{p \in \mathrm{dom}(e)} R_p(e_p).
\end{aligned}
$$

Hence, $< f(\tilde{s}), f(\tilde{t}) > \in T$.
□

Finally, we remark that a des or an rtdes may have starvation of triggers, i.e. some trigger is never consumed. We distinguish two kinds of starvation:

I. A trigger in an input channel of a processor p is not consumed since p never gets enough triggers in each input channel;

II. Though p gets enough triggers in each input channel, always other triggers are consumed.

An rtdes satisfying the conditions of Theorem 5 does not suffer from starvation of the second kind. It is the responsability of the designer to avoid starvation of the first kind.

References

Bergstra, J.A. and J.W. Klop: The algebra of recursively defined processes and the algebra of regular processes.
Proceedings 11th ICALP, Antwerpen, 1984, Springer LNCS 172, 1984.

Freedman, D.: Markov chains.
In: Holden Day Series in Probability and Statistics, 1971, ed. by E.L. Lehmann.

Lewis, H.R. and C.H. Papadimitriou: Elements of the theory of computation.
Englewood Cliffs, Prentice Hall, 1981.

Mazurkiewicz, A.: Traces, histories, graphs: Instances of a process monoid.
Math. Found. Comp. Science, Springer LNCS 176, 1984.

Peterson, J.L.: Petri net theory and the modeling of systems.
Englewood Cliffs, Prentice Hall, 1981.

Petri, C.: Introduction to general net theory. Advanced course of general net theory of processes and systems.
Springer LNCS, 1980.

Rem, M.: Partially ordered computations, with applications to VLSI design.
Proc. Found. of Comp. Science IV_2, MC Tract 159, 1983.

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS-circuits
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films
85/04	T. Verhoeff H.M.J.L. Schols	Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate
86/01	R. Koymans	Specifying message passing and real-time systems
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specifications of information systems
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several parallel systems
86/05	Jan L.G. Dietz Kees M. van Hee	A framework for the conceptual modeling of discrete dynamic systems
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987)
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86)
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes

86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)
86/14	R. Koymans	Specifying passing systems requires extending temporal logic
87/01	R. Gerth	On the existence of sound and complete axiomatizations of the monitor concept
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems
87/04	T.Verhoeff	Delay-insensitive codes - An overview
87/05	R.Kuiper	Enforcing non-determinism via linear time temporal logic specification.
87/06	R.Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch).
87/07	R.Koymans	Specifying message passing and real-time systems with real-time temporal logic.
87/08	H.M.J.L. Schols	The maximum number of states after projection.
87/09	J. Kalisvaart L.R.A. Kessener W.J.M. Lemmens M.L.P. van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics.
87/10	T.Verhoeff	Three families of maximally nondeterministic automata.
87/11	P.Lemmens	Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.
87/12	K.M. van Hee and A.Lapinski	OR and AI approaches to decision support systems.
87/13	J.C.S.P. van der Woude	Playing with patterns, searching for strings.
87/14	J. Hooman	A compositional proof system for an occam-like real-time language

87/15	C. Huizing R. Gerth W.P. de Roever	A compositional semantics for statecharts
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas
87/17	K.M. van Hee G.-J.Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples
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 J. Paredaens	The R^2 -Algebra: An extension of an algebra for nested relations
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples
88/04	G.J. Houben J.Paredaens D.Tahon	The Nested Relational Algebra: A Tool to handle Structured Information
88/05	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specifications for Information Systems
88/06	H.M.J.L. Schols	Notes on Delay-Insensitive Communication
88/07	C. Huizing R. Gerth W.P. de Roever	Modelling Statecharts behaviour in a fully abstract way
88/08	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	A Formal model for System Specification
88/09	A.T.M. Aerts K.M. van Hee	A Tutorial for Data Modelling

88/10	J.C. Ebergen	A Formal Approach to Designing Delay Insensitive Circuits
88/11	G.J. Houben J.Paredaens	A graphical interface formalism: specifying nested relational databases
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.