### A.2 - Basic transition operations

The A-token produced when the generalized transition fires is the following:

\[
\lambda_0 = \mathbf{A}(\lambda_1, \lambda_2, \ldots, \lambda_n)
\]

\[
= \alpha_1 a_1 + \alpha_2 a_2 + \ldots + \alpha_q a_q \quad \text{(A.2)}
\]

where \(\mathbf{A}\) is a predefined function for a transition type.

The transfer and merge transitions are basic operations and they will both be discussed in the following paragraphs.

#### A.2 Merge transition

The merge transition, on firing, produces an A-token:

\[
\lambda_0 = \beta_1 a_1 + \beta_2 a_2 + \ldots + \beta_q a_q
\]

\[
= \langle \beta_1, \beta_2, \ldots, \beta_q \rangle
\]
A generalized attribute:
\[ \beta_g = \frac{\sum_{h=1}^{n} c_{hg}}{n} \]
\[ \beta_i = c_{i1} + c_{i2} + \cdots + c_{in} \]
and this is the vector addition as is normally defined.
\[ \lambda_0 = \lambda_1 + \lambda_2 \cdots + \lambda_n \]

A.3 Transfer transition

The transfer transition, on firing \( \nu \) produces an \( A \)-token:
\[ \lambda_0 = \beta_i a_1 + \beta_2 a_2 \cdots + \beta_q a_q \]
\[ = \langle \beta_1, \beta_2, \ldots, \beta_q \rangle \]

A generalized attribute:
\[ \beta_g = \frac{\sum_{h=1}^{n} c_{hg}}{n} \]
\[ \beta_i = c_{i1} \epsilon_{i1} \epsilon_{i2} \cdots \epsilon_{in} \]
This is not a normal vector operation. To explain how this operation is performed, take the transfer transition of Fig. A.2 and let:
\[ A = \{ a_1, a_2, a_3 \} \]

\[ \text{Fig. A.2 Transfer transition} \]
Define the transfer transition such that:

\[
\lambda_0 = \lambda_1 \circ \lambda_2 \\
\lambda_1 = \begin{bmatrix}
\epsilon_{11} & \epsilon_{12} & \epsilon_{13} \\
\epsilon_{21} & \epsilon_{22} & \epsilon_{23}
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
a_3
\end{bmatrix} = \epsilon_1 A
\]

\[
\lambda_2 = \begin{bmatrix}
\epsilon_{21} & \epsilon_{22} & \epsilon_{23}
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
a_3
\end{bmatrix} = \epsilon_2 A
\]

\[
\lambda_0 = \lambda_1 \circ \lambda_2 = \epsilon_1 \cdot D \cdot A
\]

To obtain the "o" function, the following procedure is followed:

Define a matrix:

\[
D = \begin{bmatrix}
\epsilon_{21} & 0 & 0 \\
0 & \epsilon_{22} & 0 \\
0 & 0 & \epsilon_{23}
\end{bmatrix}
\]

This gives:

\[
\lambda_0 = \lambda_1 \circ \lambda_2 = \epsilon_1 \cdot D \cdot A
\]

\[
= \begin{bmatrix}
\epsilon_{11} & \epsilon_{12} & \epsilon_{13}
\end{bmatrix}
\begin{bmatrix}
\epsilon_{21} & 0 & 0 \\
0 & \epsilon_{22} & 0 \\
0 & 0 & \epsilon_{23}
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
a_3
\end{bmatrix}
\]
-A.5- Basic transition operations

\[
\begin{bmatrix}
\epsilon_{11} & \epsilon_{12} & \epsilon_{13} \\
\epsilon_{21} & \epsilon_{22} & \epsilon_{23} \\
\epsilon_{31} & \epsilon_{32} & \epsilon_{33}
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
a_3
\end{bmatrix}
= \epsilon_{11}a_1 + \epsilon_{12}a_2 + \epsilon_{13}a_3.
\]

To obtain this matrix D the following procedure is followed:

\[
\begin{bmatrix}
1 & \epsilon_{21} & \epsilon_{22} & \epsilon_{23} \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
= \begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix}
+ \begin{bmatrix}
0 & \epsilon_{21} & \epsilon_{22} & \epsilon_{23} \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
+ \begin{bmatrix}
0 & \epsilon_{21} & \epsilon_{22} & \epsilon_{23} \\
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1
\end{bmatrix}
= \begin{bmatrix}
\epsilon_{21} & 0 & 0 \\
0 & \epsilon_{22} & 0 \\
0 & 0 & \epsilon_{23}
\end{bmatrix}
\]

A.4 Separate subnet

The separate subnet of Fig. 4.9(b) is discussed in detail in this section. Transition \( t_1 \) transfers the input token \( x_1 \) to \( p_1 \). The separation into the attributes takes place by means of transfer
transitions \( t_3, t_4 \) and \( t_5 \). Only \( t_3 \) is considered for this explanation since the other follow the same pattern.

The input token in \( p_1 \) is:

\[
\lambda_1 = \epsilon_{11} a_1 + \epsilon_{22} a_2 + \epsilon_{33} a_3
\]

The transfer mask provided in \( p_2 \) is:

\[
\lambda_p = \omega a_1 + \omega a_2 + \omega a_3
\]

The output produced after \( t_3 \) has fired is:

\[
\lambda_{01} = \lambda_1 \circ \lambda_p
\]

\[
= (\epsilon_{11} \cdot \omega)a_1 + (\epsilon_{22} \cdot \omega)a_2 + (\epsilon_{33} \cdot \omega)a_3
\]

\[
= \epsilon_{11} a_1 + \omega a_2 + \omega a_3
\]

The attribute value of \( a_1 \) is therefore isolated.
B PETRI NETS

B.1 Introduction
B.2 Petri net theory
B.2.1 Petri net structure
B.2.2 Petri net dynamics
B.3 Extensions to Petri nets
B.3.1 Inhibitor arcs
B.3.2 Activator arcs
B.3.3 Timed transitions
B.3.4 Priorities
B.3.5 The influence of these extensions on the Petri net theory
B.4 Comparison with finite-state machines
B.5 Modelling with Petri nets

Diagram:

1. Introduction
2. Design process
3. Specification
   - system overview
4. A-nets
   - Basic transition operations
   - Distributed process control systems
5. Modelling of the real-time communication system
6. Modelling examples
7. Conclusion

Additional text:
- B.0 - A design methodology based on augmented Petri nets
B.1 Introduction

Petri nets were developed by Petri (1962). The original work was used for the description and analysis of parallel systems in the late sixties in the USA (Holt, 1970), which led to research in this field and Petri nets as well as Petri net based systems are used extensively to model and analyse diverse systems such as computer hardware (Noe, 1973).

The following features of Petri nets make them well-suited for the modelling of complex systems:

- hierarchical representation
- graphical representation
- mathematical representation
- representation of parallel processes.

The hierarchical decomposition of complex problems was discussed in chapters 2 and 3 as a means of dealing with complexity.

Graphical representation allows a visual
-B.2- **Petri nets**

representation of the structure of the problem.

Mathematical representation allows analysis of the dynamics of the problem. Many problems to be modelled have parallel parts. Petri nets are well suited to deal with this type of problem. The original work by Petri was done in order to study communication between asynchronous components in a computer system.

The study of Petri nets can be divided into two fields namely:
- Applied Petri net theory and
- Pure Petri net theory.

Applied Petri net theory concerns itself with the application of Petri nets to the modelling of systems.

Pure Petri net theory develops the basic tools, techniques and concepts for the application of Petri nets.

The development of A-nets in chapter 4 is the application of Petri nets to the design of distributed process control systems.

The remainder of this appendix presents the basics of Petri net theory, extensions made in the past, comparisons with other representation methods and an example of the use of Petri nets to model a simple
Petri nets

B.2 Petri net theory

Petri net theory concerns two main fields, namely:
- the structure, and
- the dynamics
of the network.

Petri net theory is only briefly presented here for
clarity. For a more detailed introduction, the reader
is referred to Peterson (1981). The Petri net theory
is based on multiset (or bag) theory. (Cerf, 1972)

B.2.1 Petri net structure

A Petri net consists of four parts:
- the set of places, P, represented by circles;
- the set of transitions, T, represented by bars;
- the input function, I, represented by directed arcs
to the transitions;
- the output function, O, represented by directed
arcs from the transitions.

Fig. B.1(a) is a graphical representation of a Petri
net structure. It consists of five places and four
transitions with the connecting arcs. For example arc
a₁ is an input arc to transition t₁ and arc a₂ is one
of the output arcs from transition t₁.
**B.4- Petri nets**

**Fig. B.1 Marked Petri net**

(Peterson, 1981)

Definition B.1. A Petri net structure, C, is a four-tuple, \( C = (P, T, I, O) \).
- \( P = \{p_1, p_2, \ldots, p_n\} \) is a finite set of places, \( n \geq 0 \).
- \( T = \{t_1, t_2, \ldots, t_m\} \) is a finite set of transitions, \( m \geq 0 \).
- \( I : T + P^\rightarrow \) is the input function, mapping transitions to a multi-set of places.

For example the structure of the Petri net of Fig. B.1 is as follows:

\[
C = (P, T, I, O)
\]
B.2.2 Petri net dynamics

The marking \( u \) of a Petri net consists of tokens residing in places. The number and position of tokens determine the state of the net at any time. The state of the net changes with execution of the net. The execution of the net is accomplished by the firing of transitions. The firing of a specific transition is determined by the state of the net. Tokens are normally represented by dots in the places.

**Definition B.2.** A marking \( u \) of a Petri net \( C = (P, T, I, O) \) is a function from the set of places \( P \) to the non-negative integers \( \mathbb{N} \).

\[ u : P \rightarrow \mathbb{N} \]

The marking of the Petri net of Fig. B.1 is as follows
**Petri nets**

\[ u(p_1) = 1 \]
\[ u(p_2) = 0 \]
\[ u(p_3) = 0 \]
\[ u(p_4) = 2 \]
\[ u(p_5) = 1 \]  \hspace{1cm} (B.2)

It is sometimes more convenient to represent the marking of a Petri net as a vector:  
\[ u = (u(p_1), u(p_2), \ldots, u(p_n)) \]  

For example function (B.2) would then be represented as a vector: 
\[ u = (1, 0, 0, 2, 1) \]  \hspace{1cm} (B.3)

A Petri net executes by firing transitions. A transition fires when it is enabled. When it fires, tokens are removed from the input places and created in the output places connected to this transition.  

Fig. B.2(a) shows transition \( t_1 \) enabled to fire. Places \( p_1 \) and \( p_2 \) have a token. Fig. B.2(b) shows transition \( t_1 \) after firing. The tokens in places \( p_1 \) and \( p_2 \) are consumed and a token is created in place...
Definition B.3. A transition \( t_j \in T \) in a marked Petri net \( C = (P, T, I, O) \) with marking \( \mu \) is enabled to fire if for all \( p_i \in P \),
\[
\mu(p_i) \geq \#(p_i, I(t_j)) - \#(p_i, O(t_j)).
\]

Definition B.4. A transition \( t_j \) in a marked Petri net with marking \( \mu \) may fire whenever it is enabled. Firing an enabled transition \( t_j \) results in a new marking \( \mu_1 \) defined by:
\[
\mu_1(p_i) = \mu(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j)).
\]

Transitions \( t_1 \), \( t_3 \) and \( t_4 \) are enabled in the marked Petri net of Fig. B.1(b). Any one of these transitions may fire. Firing transition \( t_4 \) consumes the token in \( p_5 \) and produces tokens in \( p_3 \) and \( p_4 \), leaving a new marking:
\[
\mu_1 = (1, 0, 1, 3, 0).
\]

A sequence of markings \( (\mu_0, \mu_1, \mu_2, \ldots) \) and a sequence of transitions which were fired \( (t_{j0}, t_{j1}, t_{j2}, \ldots) \) result if the net executes. This shows the dynamic performance of the system and may be examined for deadlock or other undesirable situations.
Various extensions were made to the original Petri nets. These extensions were mainly made to ease the graphical representation of the Petri net for a particular application.

The following extensions are important for the modelling of distributed process computer systems:
- inhibitor arcs
- activator arcs
- timed transitions
- priority on transitions.

B.3.1 Inhibitor arcs

The lack of ability to perform zero-testing is one of the major problems with regard to Petri nets. Introducing inhibitor arcs is the easiest way to introduce this capability.
Inhibitor arcs are shown graphically by an arc from a place to a transition ending in an opened circle rather than an arrow. Fig. B.3 shows transition \( t_j \) with input places \( p_1 \) and \( p_2 \) and output place \( p_3 \). The arc from \( p_1 \) to \( t_j \) is an inhibitor arc. Transition \( t_j \) is enabled to fire in Fig. B.3(a). A token is available in place \( p_2 \) and no token is present in place \( p_1 \) (zero testing.) Fig. B.3(b) shows the situation after \( t_j \) has fired. The token in \( p_2 \) is consumed and a new token is created in \( p_3 \).

B.3.2 Activator arcs

Activator arcs simplify the graphical representation of certain Petri net models. Fig. B.4(a) shows an enabled transition \( t_j \). After firing the tokens are as is shown in Fig. B.4(b). Because of the output arc from \( t_j \) to \( p_1 \), a token is still present in \( p_1 \) after the firing of \( t_j \).

The Petri net representation is simplified by using an activator arc, as is shown in Fig. B.4(c) and (d). The activator arc is represented by an arc from a place to a transition ending in a closed circle like the arc from \( p_1 \) to \( t_j \).

B.3.3 Timed transitions

The firing of a transition in a normal Petri net takes
B.10- **Petri nets**

zero time. For the simulation of real processes, time elapses from the initiating of a process until it is completed. Torn (1981) explained an extension where a transition takes $d$ time units to create the output tokens.

Time Petri nets (Merlin, 1974) are not to be confused with the above-mentioned timed transitions nets. In this instance two times, $\tau_{1j}$ and $\tau_{2j}$, are associated with a transition $t_j$. A transition $t_j$ can only fire if it was enabled for more than $\tau_{1j}$ time units and it must fire before it was enabled for $\tau_{2j}$ time units. This extension is not useful to us.

Fig. B.4 Activator arc
B.3.4 Priorities

If more than one transition is enabled to fire, there is no way to determine which one will fire in normal Petri nets. To solve this problem, priorities may be attached to each transition (Hack, 1975). The transition with the highest priority will fire in such a situation. This allows the modelling of control systems where priority is important.

B.3.5 The influence of these extensions on the Petri net theory

The introduction of activator and inhibitor arcs as well as priorities and timed transitions changes definition B.1 to:

Definition B.5. A Petri net structure, $C$, is a six-tuple, $C = (P, T, I, O, D, X)$ where:

- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places, $n \geq 0$
- $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions, $m \geq 0$
- $I : T \rightarrow P^m$ is the input arc function, mapping transitions to a multi-set of places. It consists of $I_N$, the normal arcs, $I_I$, the inhibitor arcs and $I_A$, the activator arcs.
- $O : T \rightarrow P^m$ is the output arc function, mapping
transitions to a multi-set of places.

- $D: \mathbb{T} \rightarrow \mathbb{N}$ is the duration time for each transition in time units.
- $X: \mathbb{T} \rightarrow \mathbb{N}$ is the priority for each transition.

Definition B.2 remains as it is.

Definition B.3 changes to:

Definition B.6. A transition $t_j \in T$ in a marked Petri net with marking $\mu$ is enabled to fire if for all $p_i \in P$:

$$\mu(p_i) \geq \# (p_i, l_n(t_j)) + \# (p_i, l_A(t_j))$$
and
$$\# (p_i, l_i(t_j)) = 0.$$

Definition B.4 changes to:

Definition B.7. A transition $t_j$ in a marked Petri net with marking $\mu$ may fire whenever it is enabled and has the highest priority $X_j$. Firing an enabled transition $t_j$ results in a new marking $\mu_1$ after $d_j$ time units defined by:

$$\mu_1(p_i) = \mu(p_i) - \# (p_i, l_n(t_j)) + \# (p_i, l_O(t_j)).$$

B.4 Comparison with finite-state machines

Many specifications systems, like the one described by Vitins (1981), are based on a finite-state machine
A state machine is a five-tuple:
- the finite set of states
- a finite input alphabet
- a finite output alphabet
- a next state function
- an output function.

State-machines are represented by a state diagram as in Fig. B.5. This diagram converts into the Petri net of Fig. B.6. Modelling a state diagram with a
Petri net produces a special Petri net where only one token may be present.

B.5 Modelling with Petri nets

The following simple example of a buffer is used to demonstrate the modelling capability of Petri nets. Fig. B.7 presents the graphical representation of the model and the mathematical representation is as follows:

\[ C = (P, T, I, O, X) \]

\[ P = \{P_1, P_2, P_3, P_4, P_5, P_6\} \]
\[ T = \{t_1, t_2, t_3, t_4, t_5, t_6\} \]
\[ I = \{I_N, I_I, I_A\} \]
\[ I_N(t_1) = \{p_2, p_5\} \]
\[ I_I(t_2) = \{p_1\} \]
\[ I_N(t_3) = \{p_3\} \]
\[ I_I(t_4) = \{p_4\} \]
\[ I_N(t_5) = \{p_1\} \]
\[ I_A(t_1) = \{p_1\} \]
\[ O(t_2) = \{p_2, p_6\} \]
\[ O(t_3) = \{p_3\} \]
\[ O(t_4) = \{p_4\} \]

Priority: \[ X = (0, 0, 1, 1, 2, 2) \]
The initial marking in vector form is:
\[ V_0 = (0, 6, 0, 1, 10, 0) \).

Initially only transition \( t_1 \) is enabled to fire.
After the firing of \( t_1 \) the marking will be:
\[ \mu_1 = (1, 5, 0, 1, 9, 0) \).

Transitions \( t_1, t_2 \) and \( t_3 \) are all enabled to fire.
Since the priority of \( t_6 \) is the highest, it will fire
leaving the marking:
\[ \mu_2 = (1, 5, 0, 0, 9, 0) \).

![Buffer model diagram](image)

At this stage \( t_1 \) and \( t_2 \) are enabled. Since their
priorities are the same, anyone of the two can fire.
If the modeller wants to test a specific situation
like filling \( p_1 \) first before starting to empty it, he
could set a higher priority on \( t_1 \). Let us assume that
this is the case. Transition \( t_1 \) will keep on firing until:

\[ \mu_7 = (6, 0, 0, 0, 3, 0). \]

At this stage \( t_2 \) and \( t_3 \) will be enabled and since the priority of \( t_3 \) is higher, it will fire, leaving:

\[ \mu_8 = (6, 0, 1, 0, 3, 0). \]

The buffer can now empty by firing \( t_2 \).

This model can be extended to include the environment where a request for buffering may be made at \( t_4 \) and a request for emptying at \( t_2 \). Testing of \( p_3 \) (buffer full) and \( p_4 \) (buffer empty) by the producer and user is now possible.
DISTRIBUTED PROCESS CONTROL SYSTEMS

C.1 Introduction
C.1.1 Technological drive
C.1.2 Requirements of the plant and control system

C.2 Distributed systems
C.2.1 Classification of distributed systems
- Degree of coupling
- Interconnection structure
- Direct interconnection
- Indirect interconnection
- Interdependence of components
- Synchronization between components

C.3 A maintainable DCCS
C.3.1 Message transfer
C.3.2 Message primitives
  - The OUTPUT statement
  - The INPUT statement

A -nets

Modelling of the real-time communication system

Conclusion

- A design methodology based on augmented Petri nets
C.1 Introduction

Distributed process control systems are developed because the technology to do it became available and the requirements of the control system could use it beneficially.

C.1.1 Technological drive

The availability of cheap, powerful microprocessor chips makes the use of multiple CPU systems a possibility. Multiple CPU's, tightly coupled to each other on a bus, have been used for some time in larger as well as process computer systems. For instance, the use of a separate I/O processor in process computers was introduced more than 10 years ago.

The recent development in local area networks (LAN) (Harrison, 1983) however made the looser coupling of multiple computers a possibility. These LAN's are developed for office automation systems. It is still to be seen whether a viable process control LAN will
- C.2- Distributed control systems

determine from this. The two basic contenders in the LAN field are:

- CSMA/CD media access method
- Token-passing access method.

While both systems will most probably emerge for office automation systems, a controversy still exists regarding which of these two is the most suitable for a process control environment.

C.1.2 Requirements of the plant and control system

Various properties of the plant as well as the control system favour distributed systems.

Historically the need for distribution arose because a single computer could not fulfill the functions. The limitations have been either memory capacity or CPU time. This required some rudimentary data link systems. At this point in time, the computer power was still provided centrally.

The requirements of the plant and the control system could favour distribution on a geographical (Humphrey, 1972) or functional basis (Roberts, 1978).

Lefkowitz (1977) describes integrated control of
C.3  Distributed control systems

Industrial systems with a steel works as an example. Fig. C.1 shows the material flow through this steel works. For overall control, the plant is decomposed into four hierarchial levels (Fig. C.2), namely:

- A-level: production planning, order processing, etc.
- B-level: production scheduling, data gathering etc.
- C-level: production control and reporting.
- D-level: process control.

Fig. C.1 Steel-making processes

(Lefkowitz, 1977)
The D-level can again be decomposed into a hierarchy according to Isermann (1983):
- D.1: direct control
- D.2: monitoring
- D.3: optimization
- D.4: coordination.

Re-examination of Fig. C.1 shows that the steel works consists of various plants like blast furnaces, hot strip mills, etc. These plants are geographically separated and a geographical as well as a functional distribution of the computer control system can be imagined.

The direct control on level D.1 can be partitioned, again depending on the type of process, into the following three time phases (Kruger, 1977):
C.5 Distributed control systems

- D.1.1; start-up
- D.1.2; run
- D.1.3; shut-down.

This type of control is important in batch-continuous processes like the oxygen blow steelmaking process and batch annealing of cold rolled steel.

A process may also have requirements regarding the reliability of the control system. To achieve high reliability various methods were used in the past, such as:
- a two out three system
- hot stand by system.

Utilizing the unique capabilities of a distributed computer system to achieve higher reliability is discussed by Kopetz (1982). This system forms the background of the work at the Witwatersrand University and is discussed in more detail in the next paragraph.

C.2 Distributed systems

C.2.1 Classification of distributed systems

Different aspects of distributed systems may be classified. Each may yield a different classification for the classified systems. The following aspects may be considered for classification:—(Bochmann, 1979)

- degree of coupling
Degree of coupling

The degree of coupling is determined by the amount of data exchanged per unit of local processing performed. This leads to a classification based on the bandwidth availability of the communications channel. The following three groups exist:
- weak coupling
- strong coupling
- very strong coupling.

For efficient operation of a system this would imply that very strongly coupled systems will have to operate in very close proximity to each other, while weakly coupled systems may tolerate a larger distance.

Interconnection structure

This classification involves the logical structure of the network. Two main interconnection structures may be distinguished:
- direct interconnection
- indirect interconnection.
Direct interconnection

- Dedicated facility for each pair of communicating components leading to a complete interconnection structure or a loop structure.
- Communication facility shared between all components leading to a bus or ring structure. In this type of system contention must be resolved. The two major methods are CSMA/CD and token passing. Shared memory is also an example of this method.

Indirect interconnection

- Centralized routing: this leads to a star-like network.
- Non-centralized routing: this may result in two types. A tree is an example of the first type where only one path is possible between two components and an n-cube is an example where more than one path is possible between two components.

Interdependence of components

The structure of the communicating components is used for classification in this instance.

If a component relies on the successful operation of
other components, they are strongly dependant. If components can still operate when another component has failed, they are weakly dependant. The latter type may have a better availability and may degrade gracefully in case of partial failures.

Synchronization between components

In asynchronous systems, each node operates at its own pace and waits for information from other nodes.

Synchronous systems require a common clock, usually provided through the communication medium.

C.3 A maintainable DCCS

The maintenance of real-time systems consists of the following:
- Repair work:
  Hardware failures and software design errors require repair work.
- Functional enhancement:
  For a system to remain relevant, it must change with the changing environment.

These functions may eventually cost more than the initial system (De Rose, 1978).

The reliability of hardware is treated by the normal
methods. The reliability of software, however, differs in the sense that it fails because of design errors rather than wear out.

The whole issue of software reliability is treated extensively by Kopetz (1976) and Myers (1976). Basically two complementing methodologies for producing reliable systems exist:

- producing error-free software
- detecting errors at run time and reconfigure.

The system described in the following paragraph uses the second method. It does, however, make use of the first method to produce software that is as reliable as possible.

The MARS (Maintainable Real Time System) system described by Kopetz (1982) and MacLeod (1983) is designed with the specific goal to produce a system with a high availability.

The architecture of MARS is such that it provides facilities for active and stand-by redundancy to tolerate faults during the operation of the system, and to support the evolutionary development of the real-time system.
To keep the system simple only the following fundamental concepts were adopted:

- **Divide and conquer.**
  The architecture supports the partitioning of large problems into smaller, fairly independent subproblems. The hardware and software structure should follow the structure of the problem. The solution should not be forced into any structure, for instance hierarchical, if that is not the structure of the problem.

- **Replaceable units.**
  The architecture enforces replaceable hardware and software units for maintenance.

- **Redundancy.**
  The introduction of active and stand-by redundancy is transparent to the application software.

- **Forget about the past.**
  Information is invalidated by the passage of real-time and is automatically discarded by the system.

This system is described by Kopetz (1982).

Various issues relating real-time and the communication primitives required, are discussed by MacLeod (1983). This is important for realizing the notion of forgetting about the past. Messages are time-stamped with a validity time. After this
validity time has expired, the message is automatically removed by the communications channel.

C.3.1 Message transfer

A message sent consists of a header field and a user-defined field. The header field also contains the following:
- the message name
- source and destination modules
- the validity time of the message.

State and event information

In real-time control systems, event and state information are distinguished. Event information concerns the occurrence of events, i.e., a happening in time. State information represents attribute values of objects that will remain valid for a known time interval. The communication system must handle state and event messages. The distinction between a state or event message is made at the receiver. A message sent may be treated as an event message by one receiver and as a state message by another. Event messages are consumed when read. State messages, however, may be read several times and stay the same until changed or discarded.
C.3.2 Message primitives

The OUTPUT statement

The OUTPUT statement takes the following form:

```
OUTPUT msg TO receiver VALID time; (C.1)
```

- "msg" is the name of a previously declared message
- "receiver" specifies the destination module
- "time" is a time expression giving the validity time.

The INPUT statement

The INPUT statement is used to read a message from the communication system into a task-local message variable. Various forms exist.

The simplest form is:

```
INPUT msg END;         (C.2)
```

If a message with the name "msg" is already available at the communication system it will, for an event message, be consumed. For a state message, it will only be assigned to the task-local message variable specified in the associated declaration. If this message is not available, the receiver will wait until the message arrives.
Another form of `INPUT` statement is the following:

```
INPUT msg => normal_sequence
AFTER time => timeout_sequence END; (C.3)
```

provides a timeout if the message does not arrive.

The following `INPUT` statement provides a selection of messages on a `filter` expression.

```
INPUT msg FILTER filter_exp END; (C.4)
```

If the evaluation of "filter_exp" delivers true, the message will be read, otherwise it remains in the communication system.
-Ref.0- A design methodology based on augmented Petri nets

REFERENCES

1. Introduction
2. Design process
3. Specification system overview
4. A-nets
5. Modelling of the real-time communication system
6. Modelling examples
7. Conclusion

Basic transition operations
Distributed process control systems

Petri nets
REFERENCES


Alford, M.W., (1985) SREM at the age of eight; the distributed computing design system. Computer, April 1985, pp 36-46.


-REF.2- References


Cerf, V.G. (1972), Multiprocessors, semaphores, and a graph model of computation. Ph.D. dissertation, Computer Science Department, University of California, Los Angles, California, April 1972.


Gregory, S., (1966) In a contribution to The Design Method (Edited by Gregory, S.), Butterworths, London.


REFERENCES


Author  Kruger B R
Name of thesis  A design Methodology for Distributed real-time control systems based on augmented petri nets.  1985

PUBLISHER:
University of the Witwatersrand, Johannesburg
©2013

LEGAL NOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed, or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page) for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.
Author: Kruger B R
Name of thesis: A design Methodology for Distributed real-time control systems based on augmented petri nets. 1985

PUBLISHER:
University of the Witwatersrand, Johannesburg
©2013

LEGAL NOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed, or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page) for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.