-A.2- Basic transition operations

$$\text{or} \quad \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \cdot \\ \cdot \\ \lambda_n \end{bmatrix} = \begin{bmatrix} \epsilon_{11} & \epsilon_{12} \cdots & \epsilon_{1q} \\ \epsilon_{21} & \epsilon_{22} \cdots & \epsilon_{2q} \\ & \cdot & \\ & \cdot & \\ \epsilon_{n1} & \epsilon_{n2} \cdots & \epsilon_{nq} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \cdot \\ \cdot \\ a_q \end{bmatrix} .$$

A-tokens are represented by the following convention to facilitate their written representation:

example: $\lambda_1 = \langle \epsilon_{11}, \epsilon_{12}, \cdots \epsilon_{1q} \rangle$

or $\lambda_n = \langle \epsilon_{n1}, \epsilon_{n2}, \cdots \epsilon_{nq} \rangle$.

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

$$\lambda_0 = A(\lambda_1, \lambda_2 \cdots \lambda_n)$$
$$= \beta_1 a_1 + \beta_2 a_2 + \cdots \beta_q a_q \qquad \text{(A.2)}$$

where 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:

$$\lambda_0 = \beta_1 a_1 + \beta_2 a_2 + \cdots \beta_q a_q$$
$$= \langle \beta_1, \beta_2, \cdots \beta_q \rangle .$$

-A.3- Basic transition operations

A generalized attribute:

$$\beta_g = \sum_{h=1}^{n} \epsilon_{hg}$$

$$\beta_1 = \epsilon_{11} + \epsilon_{21} + \ldots \epsilon_{n1} ,$$

and this is the vector addition as is normally defined.

$$\lambda_0 = \lambda_1 + \lambda_2 \ldots + \lambda_n .$$

A.3 Transfer transition

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

$$\lambda_0 = \beta_1 a_1 + \beta_2 a_2 \ldots \beta_q a_q$$

$$= \langle \beta_1, \beta_2 \ldots\ldots \beta_q \rangle .$$

A generalized attribute:

$$\beta_g = \prod_{h=1}^{n} \epsilon_{hg}$$

$$\beta_1 = \epsilon_{11} \epsilon_{21} \ldots \epsilon_{n1} .$$

This is not a normal vector operation. To explain how this operation is performed, take the transfer transition of Fig. A.2 and let:

$$A = \{a_1, a_2, a_3\} .$$

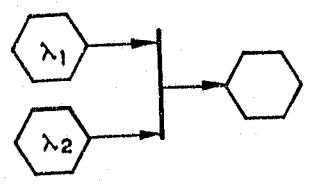


Fig.A.2 Transfer transition

-A.4- Basic transition operations

Define the transfer transition such that :

$$\lambda_0 = \lambda_1 \circ \lambda_2$$

$$\lambda_1 = \begin{bmatrix} \epsilon_{11}\epsilon_{12}\epsilon_{13} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$= \epsilon_1 A$$

$$\lambda_2 = \begin{bmatrix} \epsilon_{21}\epsilon_{22}\epsilon_{23} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$= \epsilon_2 A$$

$$\lambda_0 = \lambda_1 \circ \lambda_2$$

$$= \epsilon_{11}\epsilon_{21}a_1 + \epsilon_{12}\epsilon_{22}a_2 + \epsilon_{13}\epsilon_{23}a_3 .$$

To obtain the "$\circ$" -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:

$$\lambda_0 = \lambda_1 \circ \lambda_2$$

$$= \epsilon_1 \cdot D \cdot A$$

$$= \begin{bmatrix} \epsilon_{11}\epsilon_{12}\epsilon_{13} \end{bmatrix} \begin{bmatrix} \epsilon_{21} & 0 & 0 \\ 0 & \epsilon_{22} & 0 \\ 0 & 0 & \epsilon_{23} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$= \begin{bmatrix} \epsilon_{11}\epsilon_{21} & \epsilon_{12}\epsilon_{22} & \epsilon_{13}\epsilon_{23} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$= \epsilon_{11}\epsilon_{21}a_1 + \epsilon_{12}\epsilon_{22}a_2 + \epsilon_{13}\epsilon_{23}a_3 .$$

To obtain this matrix D the following procedure is followed:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} \epsilon_{21}\epsilon_{22}\epsilon_{23} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$+ \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} \epsilon_{21}\epsilon_{22}\epsilon_{23} \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$+ \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \epsilon_{21}\epsilon_{22}\epsilon_{23} \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \epsilon_{21} & 0 & 0 \\ 0 & \epsilon_{22} & 0 \\ 0 & 0 & \epsilon_{23} \end{bmatrix} .$$

A.4 Seperate subnet

The seperate subnet of Fig. 4.9(b) is discussed in detail in this section. Transition t_1 transfers the input token λ_1 to p_1. The seperation 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:

$$\lambda_{p2} = \nu a_1 + \omega a_2 + \omega a_3 .$$

The output produced after t_3 has fired is:

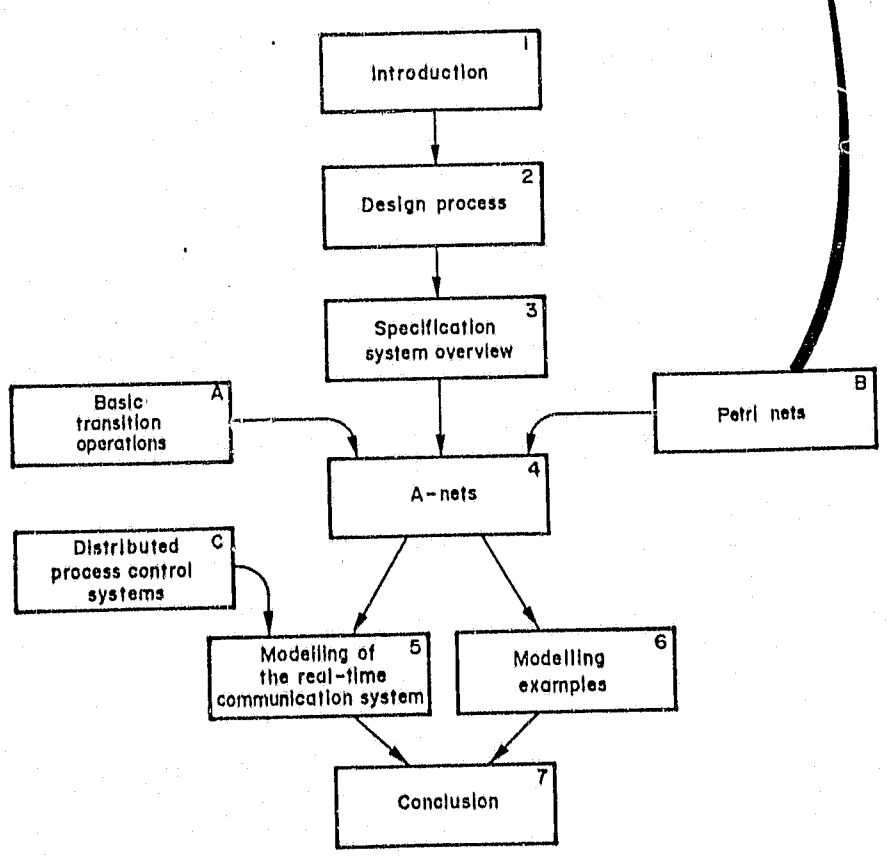
$$\lambda_{01} = \lambda_1 \circ \lambda_{p2}$$

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

$$= \epsilon_{11}a_1 + \omega a_2 + \omega a_3 .$$

The attribute value of a_1 is therefore isolated.

B PETRI NETS



A P P E N D I X B

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

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 devided 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

example.

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 a_1 is an input arc to transition t_1 and arc a_2 is one of the output arcs from transition t_1.

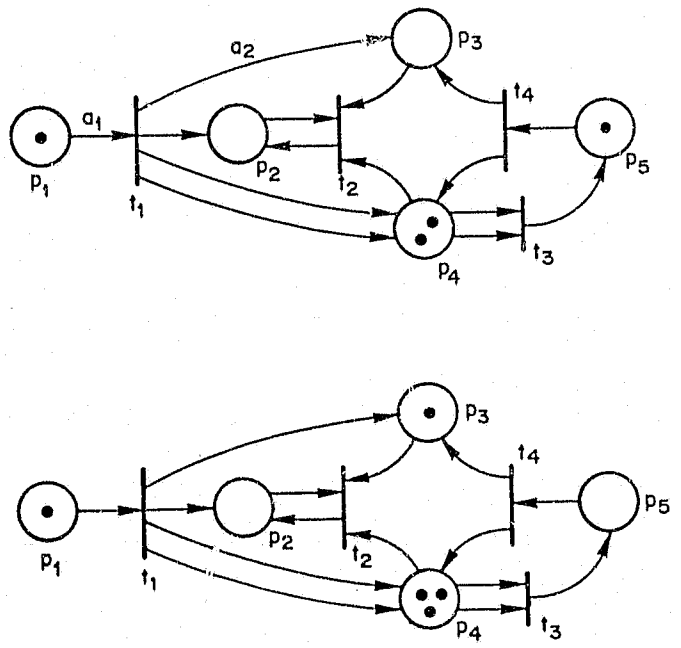


Fig. B.1 Marked Petri net

(Peterson, 1981)

Definition B.1. A Petri net structure, C, is a four-tuple, C = (P, T, I, O).

- $P = \{p_1, p_2 \ldots\ldots p_n\}$ is a finite set of places, $n \geq 0$
- $T = \{t_1, t_2 \ldots\ldots t_m\}$ is a finite set of transitions, $m \geq 0$
- $I : T \rightarrow P^{\infty}$ 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.5- Petri nets

$$P = \{p_1, p_2, p_3, p_4, p_5\} \qquad \text{(B.1)}$$

$$T = \{t_1, t_2, t_3, t_4\}$$

$$I(t_1) = \{p_1\}$$

$$I(t_2) = \{p_2, p_3, p_4\}$$

$$I(t_3) = \{p_4, p_4\}$$

$$I(t_4) = \{p_5\}$$

$$O(t_1) = \{p_2, p_3, p_4, p_4\}$$

$$O(t_2) = \{p_2\}$$

$$O(t_3) = \{p_5\}$$

$$O(t_4) = \{p_3, p_4\}$$

B.2.2 Petri net dynamics

The marking μ, 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 μ of a Petri net $C = (P, T, I, O)$ is a function from the set of places P to the non-negative integers N.

$$\mu : P \rightarrow N .$$

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

$\mu(p_1) = 1$

$\mu(p_2) = 0$

$\mu(p_3) = 0$

$\mu(p_4) = 2$

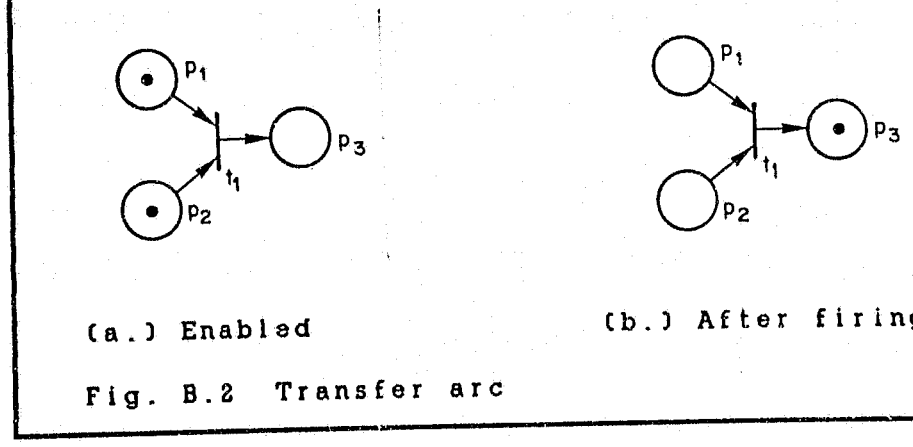
$\mu(p_5) = 1$ _____(B.2)

It is sometimes more convenient to represent the marking of a Petri net as a vector:

$\mu = (\mu(p_1), \mu(p_2), \ldots \mu(p_n))$.

For example function (B.2) would then be represented as a vector:

$\mu = (1, 0, 0, 2, 1)$ _____(B.3)



(a.) Enabled (b.) After firing

Fig. B.2 Transfer arc

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

p_3.

Definition B.3. A transition $t_j \in T$ in a marked Petri net $C = (P, T, I, O)$ with marking μ is enabled to fire if for all $p_i \in P$, $\mu(p_i) \geq \#(p_i, I(t_j))$.

Definition B.4. A transition t_j in a marked Petri net with marking μ may fire whenever it is enabled. Firing an enabled transition t_j results in a new marking μ_1 defined by:
$\mu_1(p_i) = \mu(p_i - \#(p_i, I(t_j)) + \#(p_1, O(t_j))$.

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.

B.3 Extensions to Petri nets

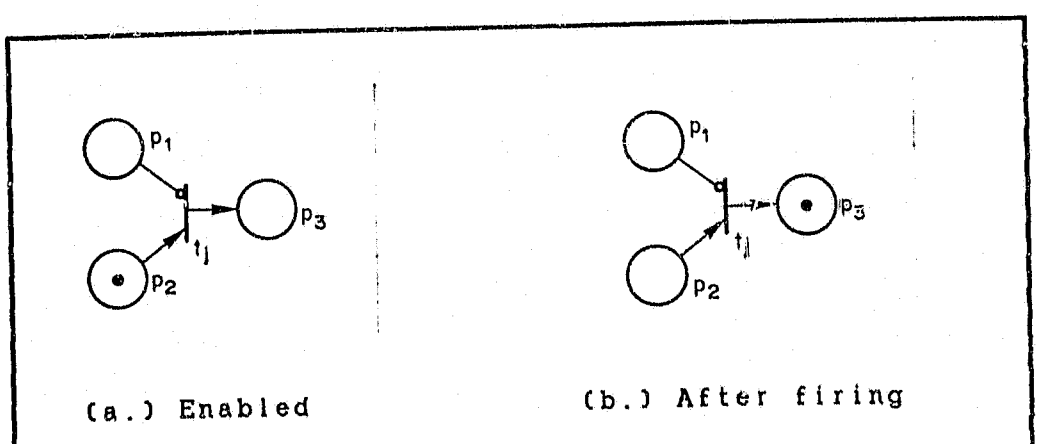
Various extensions were made to the original Petri nets. These extensions were mainly made to ease the graphical representation of the Petri net for a particular application.

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.



(a.) Enabled (b.) After firing

Fig. B.3 Inhibitor arc

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

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, τ_{1j} and τ_{2j}, are associated with a transition t_j. A transition t_j can only fire if it was enabled for more than τ_{1j} time units and it must fire before it was enabled for τ_{2j} time units. This extension is not useful to us.

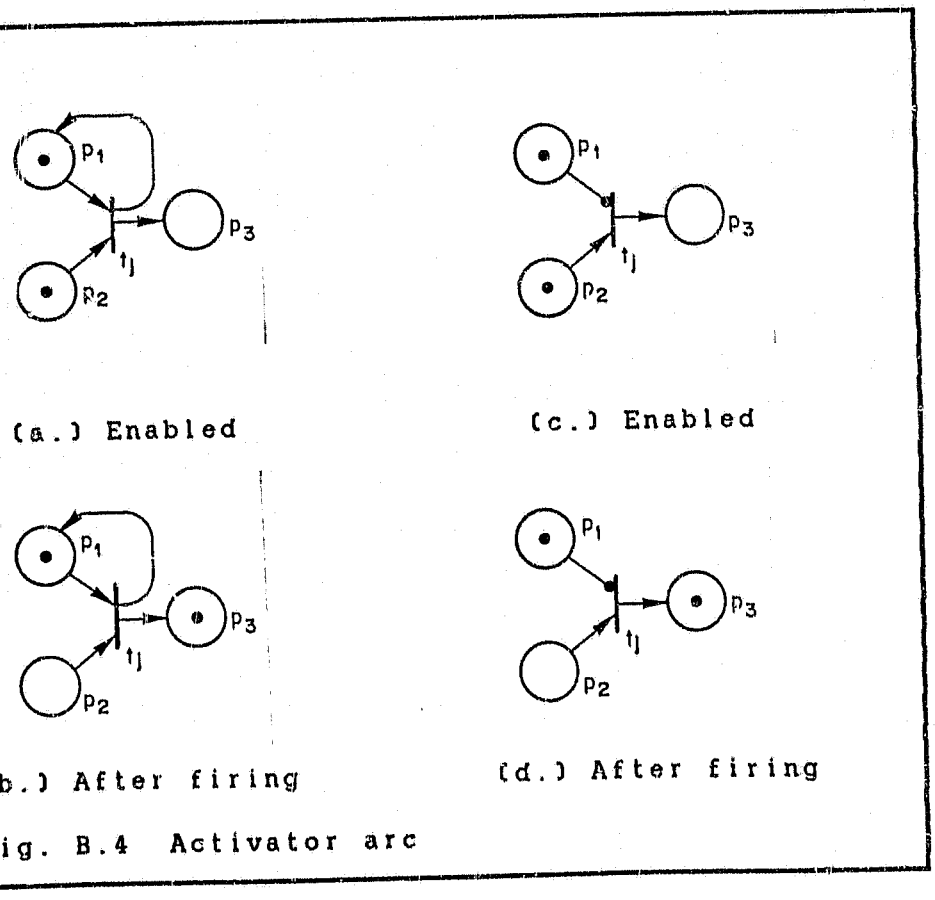


Fig. B.4 Activator arc

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:

- $P = \{p_1, p_2, \ldots\ldots p_n\}$ is a finite set of places, $n \geq 0$
- $T = \{t_1, t_2 \ldots\ldots t_n\}$ is a finite set of transitions, $m \geq 0$
- $I : T \rightarrow P^{\infty}$ 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.
- $O : T \rightarrow P^{\infty}$ is the output arc function, mapping

transitions to a multi-set of places.

- D : T → N is the duration time for each transition in time units.
- X : T → 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 μ is enabled to fire if for all $p_i \in P$:

$$\mu(p_i) \geq \#(p_i, I_n(t_j)) + \#(p_i, I_A(t_j))$$
$$\text{and } \#(p_i, I_I(t_j)) = 0 .$$

Definition B.4 changes to:

Definition B.7 A transition t_j in a marked Petri net with marking μ may fire whenever it is enabled and has the highest priority X_j. Firing an enabled transition t_j results in a new marking μ_1 after d_j time units defined by:

$$\mu_1(p_i) = \mu(p_i) - \#(p_i, I_N(t_j)) + \#(p_i, O(t_j)) .$$

B.4 Comparison with finite-state machines

Many specifications systems, like the one described by Vitins (1981), are based on a finite-state machine

model of the system.

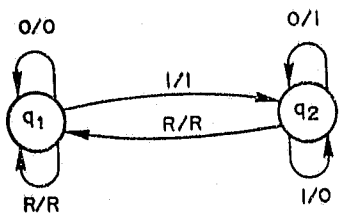


Fig. B.5 State diagram

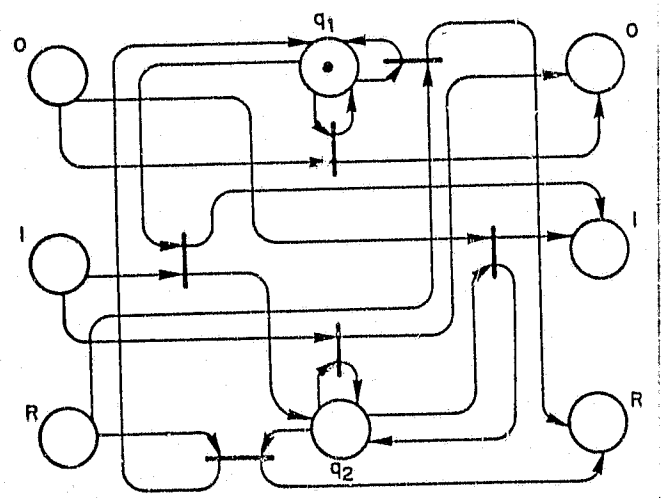


Fig. B.6 Petri net of state diagram

(Peterson, 1981)

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

-B.14- Petri nets

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:

$C = (P, T, I, O, X)$

$P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$

$T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$

$I = (I_N, I_I, I_A)$

$I_N(t_1) = \{p_2, p_5\}$

$I_N(t_2) = \{p_1\}$

$I_N(t_5) = \{p_3\}$

$I_N(t_6) = \{p_4\}$

$I_I(t_3) = \{p_2, p_3\}$

$I_I(t_4) = \{p_1, p_4\}$

$I_A(t_5) = \{p_2\}$

$I_A(t_6) = \{p_1\}$

$O(t_1) = \{p_1\}$

$O(t_2) = \{p_2, p_6\}$

$O(t_3) = \{p_3\}$

$O(t_4) = \{p_4\}$

Priority: $X = (0, 0, 1, 1, 2, 2)$

The initial marking in vector form is:

μ_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:

μ_1 = (1, 5, 0, 1, 9, 0).

Transitions t_1, t_2 and t_6 are all enabled to fire. Since the priority of t_6 is the highest, it will fire leaving the marking:

μ_2 = (1, 5, 0, 0, 9, 0).

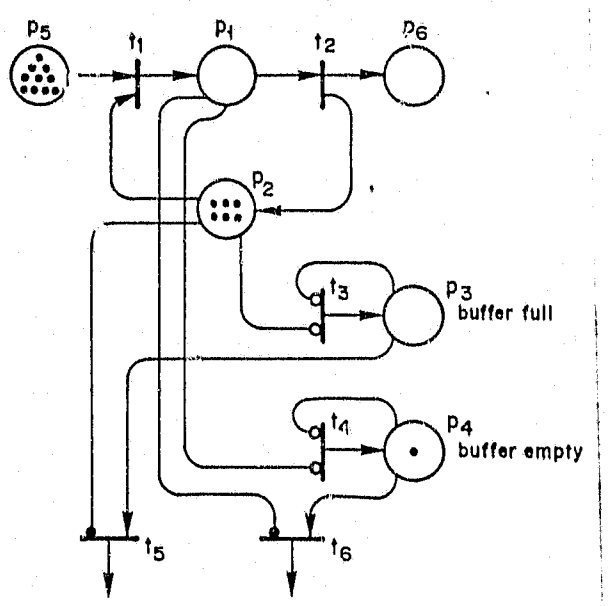


Fig. B.7 Buffer model

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:

$\mu_7 = (6, 0, 0, 0, 3, 0)$.

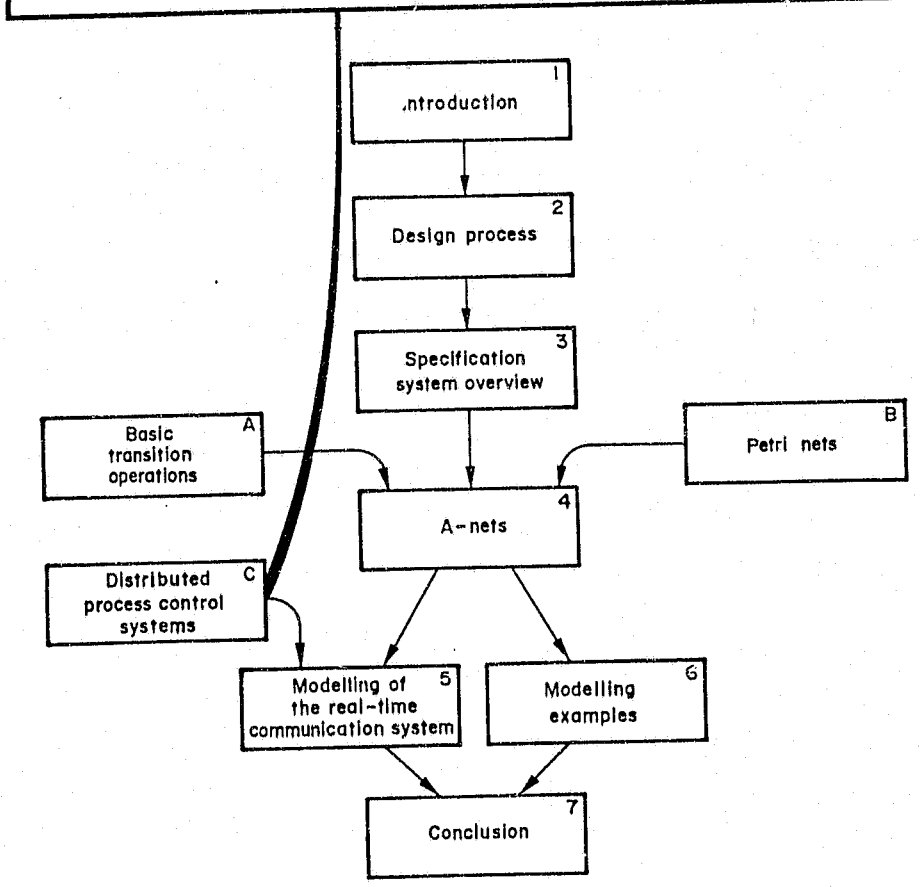
At this stage t_2 and t_3 will be enabled and since the priority of t_3 is higher, it will fire, leaving:

$\mu_8 = (6, 0, 1, 0, 3, 0)$.

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_1 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.

C DISTRIBUTED PROCESS CONTROL SYSTEMS



A P P E N D I X C

DISTRIBUTED PROCESS CONTROL SYSTEMS

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 benificially.

C.1.1 Technological drive

The availability of cheap, powerful microprocessor chips makes the use of multiple CPU systems a possiblility. 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 seperate 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

emerge 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 fulfil 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

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.

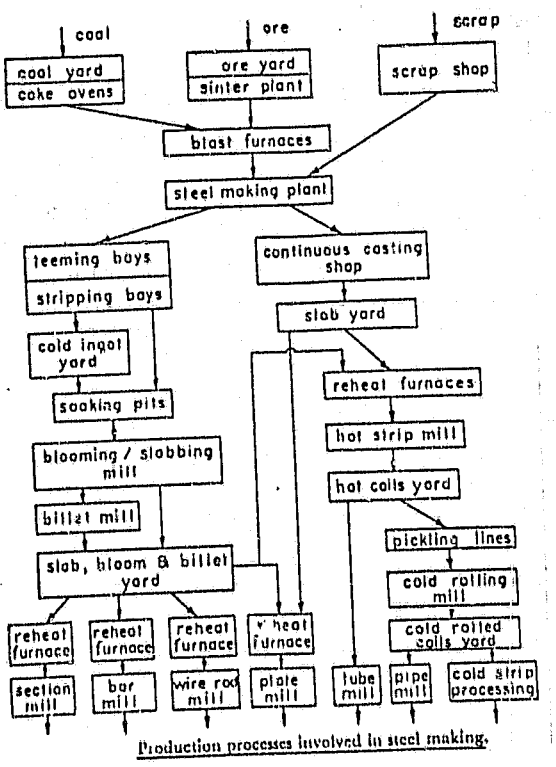


Fig. C.1 Steel-making processes

(Lefkowitz, 1977)

-C.4- Distributed control systems

The D-level can again be decomposed into a hierarchy according to Isermann (1982):

- D.1; direct control
- D.2; monitoring
- D.3; optimization
- D.4; coordination.

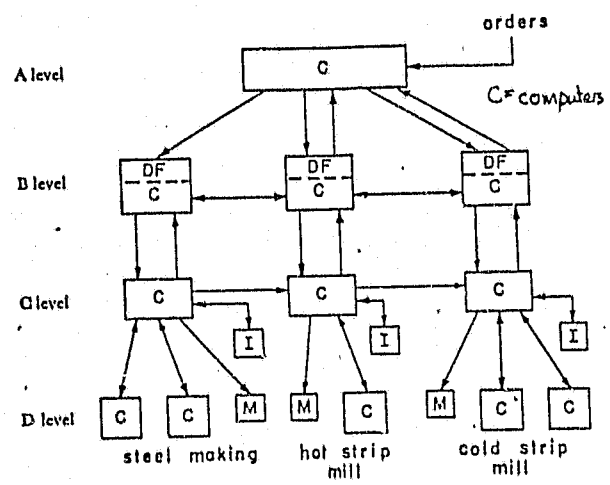


Fig. C.2 Steel works integrated control system (Lefkowitz, 1977)

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 seperated 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):

- 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

-C.6- Distributed control systems

- interconnection structure
- interdependance of components
- synchronization between components.

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 exists:

- 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.

-C.7- Distributed control systems

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 a 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 succesful 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 eventualy cost more than the initial system (De Roze, 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.

-C.10- Distributed control systems

To keep the system simple only the following fundamental concepts were adopted:

- Divide and conquer.

  The architecture supports the partioning of large problems into smaller, fairly independent subproblems. The hardware and software structure should follow the structure of the problem. The solution should for instance 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 occurence 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:

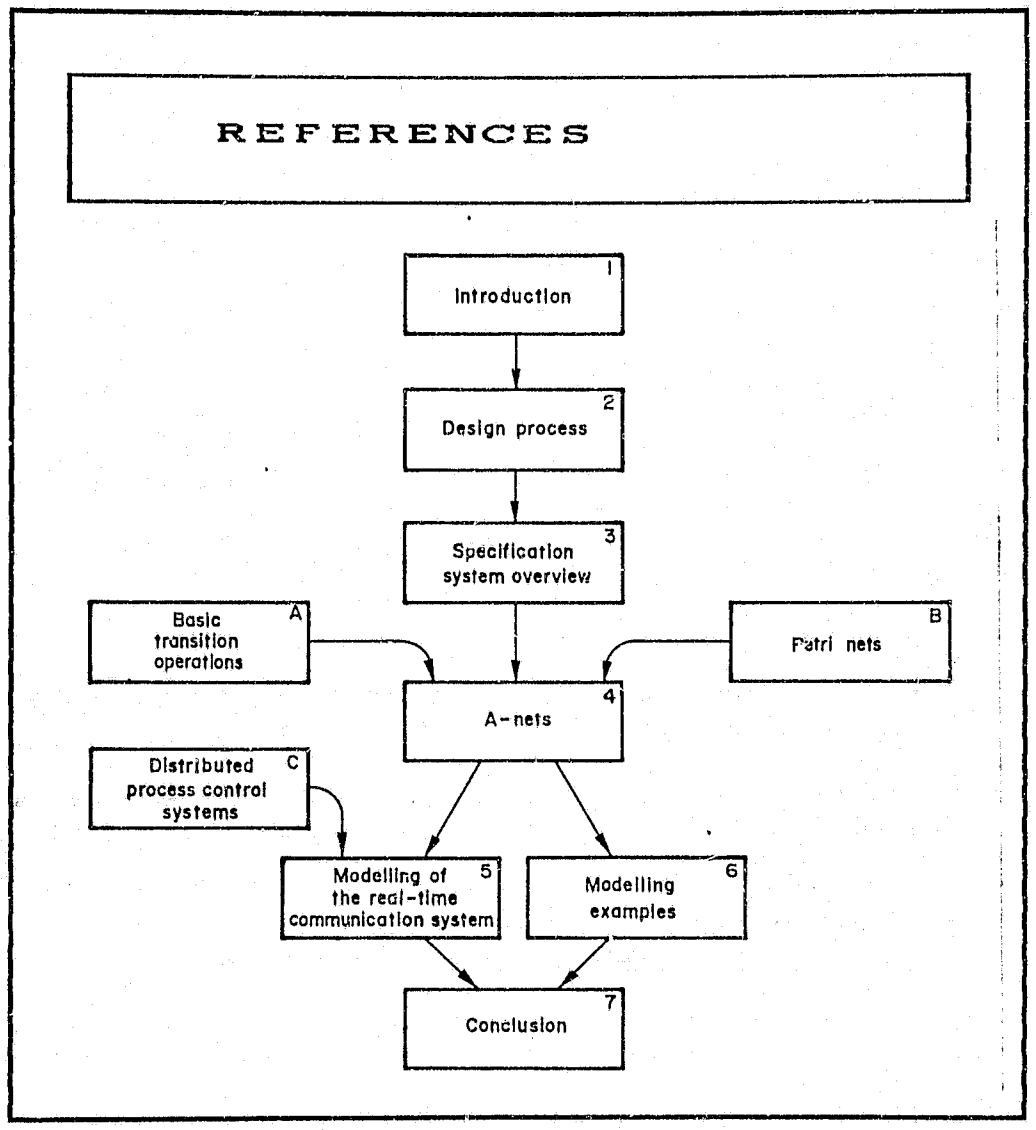
```
INPUT msg => normal_sequence
AFTER time => timeout_sequence END; _______ (C.3)
```

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.


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

R E F E R E N C E S

Alexander, C., (1963) The Determination of Components for an Indian Village. In: **Conference on Design Methods**. (Edited by Jones, J., and Thornley, D.G.) Oxford: Pergamon Press, New York.

Alexander, C., (1967) **Notes on the Synthesis of form.** Harvard University Press, Cambridge, Massachusetts.

Alford, M.W., (1977) A requirements engineering methodology for real-time processing requirements. **IEEE Trans. on Software Engineering.** Vol. SE-3, No. 1, Jan 77, pp. 60-69.

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

Archer, L.B., (1965) **Systematic Method for Designers.** Council of Industrial Design, London.

Asimow, M., (1962) **Introduction to Design** Prentice-Hall, New York.

Balzer, R. and Goldman, N. (1979) Principles of good software specification and their implementations for specification language. In: **Proceedings, specification of reliable software,** IEEE catalog No. 79 CH1401-9C, 1979, pp. 58-67.

Bazjanac, V., (1974) Architectural Design Theory: Models of the Design Process In: **Basic Questions of Design Theory** (Edited by W R Spillers) North-Holland Publishing Company, New York.

Becker, J.M., (1973) **A structural design process philosophy and methodology.** Ph.D. dissertation, University of California, Berkeley, Jun 1973.

Bell, T.E., and Bixler, D.C., (1976) **A flow-oriented requirements statement language.** TRW Defense and Space system Group. Report TRW-SS-76-02, Apr 76.

Bergland, G.D. and Gordon, R.D. (1981) Software project management. In Tutorial: **Software design strategies** (2nd ed) (Edited by Bergland, G.D., and Gordon, R.D.) IEEE Computer Society, Piscataway, N.J.

 # References

Bernus, P., and Hatvany, J. (1979) Computer aids to the design of integrated manufacturing systems. **Computers in Industry**, Vol. 1, No. 1, Jul 79, pp. 11-19.

Berthelot, G., and Terrat, R. (1982), Petri nets theory for the correctness of protocols, **IEEE Trans. on Communications**, Vol. COM-30, No. 12, Dec 1982, pp. 2497-2505.

Biewald, J., Goehner, P., Lauber, R., Schelling, H., (1979), EPOS- A specification and design technique for computer controlled real-time automation systems. **4th International Conference on Software Engineering**, Sep 17-19, 1979, Munich, pp. 245-273.

Bochmann, G.V., (1979), **Architecture of Distributed Computer Systems**. Springer-Verlag, Berlin, 1979.

Boehm, B.W., (1976) Software Engineering. **IEEE Trans. on Computers**. Vol. C-25, No. 12, Dec 1976, pp. 1226-1241.

Boehm, B.W., (1984) Verifying and Validating Software Requirements and Design Specifications. **IEEE Software**, Vol. 1, No. 1, Jan 84, pp. 75-88.

Booker, P.J., (1964) Written Contribution appended to Conference on the Teaching of Engineering Design (Edited by Booker, P.J.) Institution of Engineering Designers, London.

Borgida, A., Greenspan, S. and Mylopoulos, J. (1985), Knowledge representation as the basis for requirements specifications. **Computer**. April 1985, pp 82-91.

Brauer, W. (1979), **Net theory and applications**. Springer-Verlag, Berlin.

Brooks, F.P. (1979), **The mythical man-month**. Addison-Wesley Publishing Company. Reading Massachusetts

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.

Chestnut, H., (1970) Information Requirements for Systems Understanding. **IEEE Trans on Systems, Science and Cybernetics**, Vol. SSC-6, No. 1, Jan 70, pp. 3-12.

Chu, Y., (1982) **Software blueprint and examples**. Lexington books, D. C. Heath and Company, Lexington, Massachusetts.

-REF.3- References

Davis, A.M. and Rauscher, T.G. (1979), Formal techniques and automatic processing to ensure correctness in requirements specifications. In: **Proceedings, Specification of re`iable software**, IEEE catalog No. 79 CH1401-9C, 1979, pp. 15-35.

Davis, A.M., (1982) The design of a family of application-oriented requirements languages. **Computer**, May 82, pp. 21-28.

Davis, C.G., and Vick, C.R., (1977) The Software development system. **IEEE Trans. on Software Engineering.** Vol. SE-3, No. 1, Jan 77, pp. 69-84.

De Roze, B.C., and Nyman, T.H., (1979) The Software Life cycle - a management and technological challenge within the Department of Defence, **IEEE Trans. on Software Engineering**, Vol. SE-4, Jul 1979, pp. 309-318.

Dewey, J. (1910) **How we Think**, D.C. Heath & Co, Publishers, Boston, 1910.

Farr, M., (1966) **Design Management.** Hutchinson London.

Fielden, G.B.R., (1963) (The Fielden Report) Engineering Design. Her Majesty's Stationery Office, London.

Gane, C., and Sarson, T. (1979), **Structured Systems analysis: tools and techniques.** Prentice-Hall Inc., Englewood Cliffs, New Jersey.

Girault, C. and Reisig, W. (1981), **Application and theory of Petri nets.** Springer-Verlag, Berlin.

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

Guilford, J.P. (1967) **The Nature of Human Intelligence**, Mc Graw Hill Book Co. New York, 1967.

Hack, M. (1975), **Decidability Questions for Petri Nets.** Ph.D. dissertation, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts, Dec 1975.

Harrison, T.J., (1983) IEEE Project 802: Local and Metropolitan Area Network Standards., In: **Distributed Computer Control Systems 1983** (Edited by Rodd, M.G.) Proceedings of the Fifth IFAC Workshop, Sabi-Sabi, Transvaal, South Africa, 18-20 May, 1983.

Heninger, K.L. (1979), Specifying software requirements for complex systems: New techniques and their application. In: **Proceedings, Specification of reliable software**, IEEE catolog No. 79 CHi401-9C 1979 pp. 1-14.

Hesse, W., (1981), Methoden und Werkzeuge zur Software-Entwicklung: Einordnung und Überblick. In: **Werkzeuge der Programmiertechnik**, GI Arbeitstagung, Karlsruhe 16-17 Marz 81. pp. 113-153.

Himmelblau, D.M., (1974) Design in Chemical Engineering In: **Basic Questions of Design Theory** (Edited by W R Spillers) North-Holland Publishing Company, New York.

Holt, A., and Commoner, F. (1970), Events and Conditions. Record of the Project **MAC Conference on Concurrent Systems and Parallel Computation.** New York, ACM, Jun 1970, pp. 3-52.

Hommel,. G., (1980), **Vergleich verschiedener Spezifikationsverfahren am Beispiel einer Paketverteilanlage.** Report KfK-PDV-186, Kernforschungszentrum, Karlsruhe, August 1980.

Humphrey, J.W., (1979), The Isabelle control system - design concepts., In: **Distributed Computer Control Systems** (Edited by Harrison T.J.) Proceedings of the IFAC Workshop, Tampa, Florida, USA, 2-4 Oct 1979.

Jackson, K., and Simpson, H.R. (1975) **MASCOT- A modular approach to software contstruction operation and test.** RRE Technical note No. 778, Oct 75.

Jackson, K., and Harte, H.F., (1976) The achievement of well-structured software in real-time applications. **IFAC/IFIP Workshop on real-time programming**, 1976.

Jackson, M.A., (1976a) Constructive methods of program design. In Tutorial: **Software design strategies** (2nd ed) (Edited by Bergland, G.D., and Gordon, R.D.) IEEE Computer Society, Piscataway, N.J.

Jensen, R.W., and Tonies C.C., (1979) **Software Engineering.** Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Johnson, D.M., (1955) **The Psychology of Thought and Judgement.** Harper & Brothers, New York, 1955.

Jones, J.C., (1966) Design Methods Reviewed. In **The Design Method** (Edited by Gregory, S.), Butterworths, London.

Jones, J.C., (1969) **Design Methods**, Wiley-Interscience London 1969.

References

Klir, G.J., (1969) An Approach to General Systems Theory. Van Nostrand Reinhold Company, New York.

Kopetz, H., Lohnert, F., Merker, W., Pauthner G., (1982) An architecture for a Maintainable Real-Time System (MARS). Institut fur Technische Informatik der Technischen Universitat Berlin.

Kopetz, H., (1976), Software Reliability, The Macmillan Press, London.

Kruger, B.R. (1979), Considerations in the choice of the programmable logic controllers for the Richards Bay harbour complex. Paper presented at: SACAC Affiliates Forum, Pretoria, Feb 14, 1979

Kruger, B.R., Rodd, M.G. and MacLeod I.M. (1985) An approach to DCCS design based on the use of augmented Petri nets. Paper presented at DCCS-85, Monterey, California, May 20 - 22 1985.

Kruger, J.J., (1977) A unified approach to controller synthesis for technological plants., Ph.D. dissertation in Electrical Engineering, RAU, South Africa, Nov 1977.

Lauber, R.J., (1982) Development support systems., Computer, May 1982, pp. 36-46.

Leidigkeit, A. (undated) ALS: eine graphische Sprache zur Programmierung von Ablaufsteuerungen. Teil 1. Siemens AG, E81, Erlangen, Germany.

Lefkowitz, I., (1977), Integrated control of industrial systems., Trans. Royal Society, London, Vol. 287, 1977, pp. 443-465.

Ludewig, J., and Streng, W. (1978) Überblick und Vergleich verschiedener Mittel für die Spezifikation und den Entwurf von Software.
Report KfK2506
Kernforschungszentrum, Karlsruhe, Marz 1978.

Ludewig, J. (1980) PCSL- A process control software specification language. Report KfK 2874. Kernforschungszentrum. Karlsruhe, Apr 1980.

Ludewig, J. (1981a) Specification of a specification language. IFAC/IFIP Workshop on real-time programming, Kyoto, Japan.

Ludewig, J., and Sandmayr, H., (1981b) On the specification of distributed computer control systems. 3rd DCCS Workshop, Beijing, China, 1981.

Ludewig, J., and Eckert, K., (1981c) ESPRESO-W, ein Werkzeug fur die Spezifikation von Prozessrechner-Software. In **Werkzeuge der Programmiertechnik**, GI-Arbeitstagung, Karlsruhe, März 1981, pp. 101-112.

Ludewig, J., (1982a), ESPRESO- A system for process control software specification. **7th Conference on Operating Systems**. Visegrad, Hungary, Jan 82.

Ludewig, J., (1982b), Computer-aided specification of process control systems., **Computer**, May 1982, pp. 12-20.

MacLeod, I. (1983) **A study of issues relating to real-time in distributed computer control systems.** Ph.D. thesis, Department of Electrical Engineering, University of the Witwatersrand, Johannesburg, Dec. 1983.

Matchett, E., (1968) Control of Thought in Creative Work. **The Chartered Mechanical Engineer**, pp. 14, 4.

Merlin, P. (1976), A Methodology for the Design and Implementation of Communication Protocols. **IEEE Trans. on Communications**, Vol. COM-24, No. 6, Jun 76, pp. 614-621.

Mesarovic, M.D., Macko, D., and Takahara, Y. (1970) **Theory of Hierarchical, Multilevel, Systems.** Academic Press, New York.

Meyer, B., (1985) On Formalism in Specifications. **IEEE Software**, Vol. 2, No. 1, Jan 85, pp. 6-26.

Miller, G.A., (1956) The Magical Number Seven, Plus or Minus two: Some Limits on our Capacity for processing information. **The Psychological Review**, Vol. 63, No. 2, March 55. pp. 81-97.

Motus, L., and Karamees, K., (1982) A model based design of distrubuted control system software. **DCCS-82**, Tallinn, USSR.

Myers, G.J., (1976) **Software Reliability**, John Wiley & Sons, New York.

Newell, A., Shaw, J.G., and Simon, H.A. (1958) Elements of a Theory of Human Problem solving. **Psychological Review**. Vol. 65, No. 3, 1958, pp. 151-166.

Noe, J.D. and Nutt, G.J. (1973), Macro E-nets for representation of parallel systems. **IEEE Trans. on Computers**, Vol. C-22, No. 8, Aug 1973, pp. 718-727.

Page, J.K., (1966) Contribution to Building for **People**, 1965 Conference Report. Ministry of Public Building and Works, London.

-REF.7- References

Parnas, D.L. (1972) On criteria to be used in decomposing systems into modules. **Communications of the ACM**, Vol. 15 No. 12 Dec 1972 pp. 1053-1058

Petri, C., (1962) **Kommunikation mit Automaten**. Ph.D. dissertation, University of Bonn, Bonn, West Germany (1962).

Peterson, J.L. (1981), **Petri net theory and the modeling of systems**. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.

Quirk, W.J., and Gilbert, R., (1977) **The formal specification of the requirements of real-time systems**. Harwell, AERE-R8602, Jun 77.

Ramamoorthy, C.V., and So, H.H., (1978) Software Requirements and Specifications: Status and Perspectives. In Tutorial: **Software Methodology**. (Edited by Ramamoorthy, C.V., and Yeh, R.T.) IEEE Computer Society, Piscataway, N.J.

Reswick, J.B., (1965) **Prospectus for Engineering Design Centre**. Case Institute of Technology, Cleveland, Ohio.

Roberts, P.D., (1978) Hierarchial control and decomposition of a chemical plant. **Int. J. Systems Sci.**, Vol. 10, No. 2, Jan 78, pp. 207-223.

Rodd, M.G. (1982) **Proposal for the formation of a research program into Distributed Computer Control Systems**. Department of Electrical Engineering, University of the Witwatersrand, Johannesburg, 1982.

Ross, D.T., and Schoman, K.E. Jnr, (1977a), Structured analysis for requirements definition. **IEEE Trans. on Software Engineering**. Vol. SE-3, No. 1, Jan 77, pp. 6-15.

Ross, D.T., (1977b), Structured Analysis (SA): A language for communicating ideas. **IEEE Trans. on Software Engineering**. Vol. SE-3, No. 1, Jan 77, pp. 16-34.

Ross, D.T., (1985), Applications and extensions of SADT. **Computer**. April 1985, pp 25-34

Rubinstein, M.F., (1982) Problem solving on both sides of the brain. **IEEE Engineering Management Review.**, Vol. 10, No. 1, March 82, pp. 82-85.

Scheffer, P.A., Stone, A.H. (III) and Rzepka, W.E. (1985) A case study of SREM. **Computer**, April 1985, pp 47-54.

-REF.8- References

Siemens (undated) Simatic S32 brochure. Bestell Nr. E329/1012.

Simon, H.A., (1962) **The architecture of complexity.** Proceedings of the American Philosophical Society, Vol. 106, No. 6, Dec 1962, pp. 467-482.

So, H.H., (1979) **An approach to the requirements analysis and specification of large-scale software systems.** Ph.D. dissertation in Engineering at the University of California, Berkeley.

Stay, J.F., (1976) HIPO and integrated program design. **IBM Systems Journal.** Vol. 15, No. 2, 1976, pp. 143-154.

Stevens, W.P., Meyers, G.J., and Constantine, L.L., (1974) Structured design. **IBM Systems Journal.** Vol. 13, No. 2, 1974, pp. 115-139.

Teichroew, D., and Hershey, E.A., (1977) PSL/PSA: A computer-aided technique for structured documentation and analysis of information processing systems. **IEEE Trans. on Software Engineering,** Vol. SE-3, No. 1, Jan 77, pp. 41-48.

Tonies, C.C., (1979), Project Management fundamentals In: **Software Engineering** (Edited by Jensen, R.W., and Tonies C.C.) Prentice-Hall Inc., Englewood Cliffs, New Jersey.

Torn, A.A. (1981), Simulation graphs: a general tool for modelling simulation designs. **Simulation,** Dec 1981, pp. 187-194.

Van Leer, P. (1976) Top-down development using a program design language. **IBM Systems Journal,** Vol. 15, No. 2, 1976, pp. 155-177.

Vitins, M. (1981), **Requirements specifications for industrial real-time automation systems.** Brown Boveri, Forschungsbericht KLR-81-59C, Apr 81.

Yau, S.S., and Caglayan, M.V. (1983), Distributed software system design representation using modified Petri nets. **IEEE Trans. on Software Engineering,** Vol. SE-9, No. 6, Nov 83, pp. 733-745.

Zave, P., (1982) An Operational Approach to Requirements Specification for Embedded Systems. **IEEE Trans. on Software Engineering.,** Vol. SE-8, No. 3, May 1982, pp. 250-269.