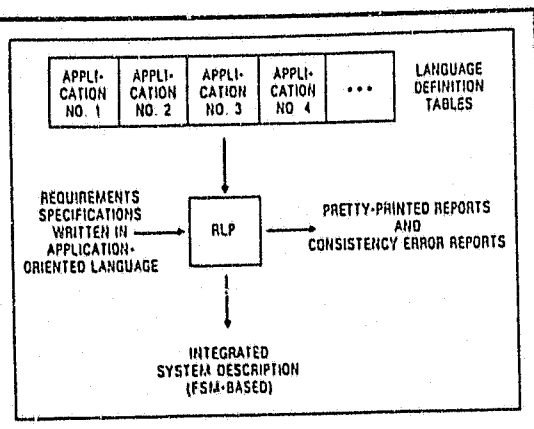
different applications, as is shown in Fig. 3.8(a). The readability of this system is shown in the example in Fig. 3.8(b) of a telephone exchange.



(a.)

The requirements language processor.

```
IDLE
WAIT FOR CALLING_PARTY TO GO_OFF_HOOK
IF SYSTEM IS SATURATED, THEN DISCONNECT
ELSE CALLING_PARTY RECEIVES DIAL_TONE
    WAIT FOR CALLING_PARTY TO GO_ON_HOOK, OR
          CALLING_PARTY DIALS LOCAL_NUMBER
    IF CALLING _PARTY IS ON_HOOK, SYSTEM IS IDLE
    ELSE IF CALLED_PARTY IS BUSY,
       THEN CALLING_PARTY RECEIVES BUSY_TONE.
          WAIT FOR CALLING_PARTY TO GO_ON_HOOK.
          SYSTEM IS IDLE
       ELSE CALLING_PARTY RECEIVES RINGBACK TONE.
          CALLED_PARTY RECEIVES RING_TONE
          WAIT FOR CALLING_PARTY TO GO_ON_HOOK, OR
                CALLED_PARTY TO GO_OFF_HOOK
          IF CALLING_PARTY IS ON_HOOK,
          THEN SYSTEM IS IDLE.
          ELSE CONNECT CALLED_PARTY AND CALLING_PARTY
             WHILE CALLED_PARTY IS OFF_HOOK
                WAIT FOR CALLED_PARTY TO GO_ON_HOOK, OR
                      CALLING_PARTY TO GO_ON_HOOK.
                IF CALLING_PARTY IS ON_HOOK,
                THEN DISCONNECT PARTIES.
                      WAIT FOR CALLED_PARTY TO GO_ON_HOOK
                      SYSTEM IS IDLE
                ELSE WAIT FOR CALLING_PARTY TO GO_ON_HOOK, OR
                      CALLED_PARTY TO GO_OFF_HOOK
```

(b.)

.. A local telephone call.

Fig. 3.8 RLP - example

(Davis, 1982)

3.2.10 Motus Quirk system

Of all the specification systems discussed up to now, very few cater for distributed computer control systems. The SREM system can deal with parallel

processes but the special problems of a distributed system is not provided for.

Motus (1982) uses a model developed by Quirk (1977) to specify distributed computer control systems. Quirk's model is based on the conception that a system is a family of interacting subsystems. These subsystems form a hierarchy. A system will function correctly if:

- each subsystem functions
- the subsystems interact cooperatively.

The system is analyzed until each subsystem is responsible for evaluating a single conceptual entity; a state of the system. This subsystem is called a process. It is assumed that this state is evaluated correctly in a specified time. A channel provides the interaction between the processes. This implies that the state of the supplier process is coupled to the requestor process through the channel.

Quirk identifies three types of channels, while Motus uses four types. It is, however, stated that this is only a subset of the types of available channels. The following channels are defined:

- null channel, which forces the requestor and supplier to have the same time set, with an empty transfer
- synchronous channel, which again forces the

requestor and supplier to have the same time set, and the transfer contains a predetermined number of elements

- semi-synchronous channel, which is similar to the synchronous channel except for the processes that execute cofrequently
- asynchronous channel, which transmits a predetermined number of elements but the requestor and supplier execute in different time sets.

Motus concentrates on the modelling of the channels. The structure used concists of parallel processes interacting cooperatively. The following processes are important in process control systems:

- event-driven processes
- time cyclic driven processes
- process-driven processes.

3.2.11 Grafcet

A system designed for use on Programmable Logic Controllers (PLC) is available as a recommended standard (Grafcet, IEC SC65A/WG6). This methodology is based on Petri nets, discussed in appendix B of this thesis.

Since this methodology is designed for use on PLC's, it is suitable for sequence control of a plant or machine. The example shown in Fig. 3.9 is for the

sequencing of a press. An overview is shown in Fig. 3.9(a) and the detail in Fig. 3.9(b). An older standard, (DIN 40719), is used for the detail design. Detail of this standard is shown in Fig. 3.9(c). Fig. 3.9(d) shows how this method caters for parallel execution and selection. For example, either the loop starting with T1 or T10 or T12 is executed, while the loops starting with S5, S11 and S12 are executed in parallel.

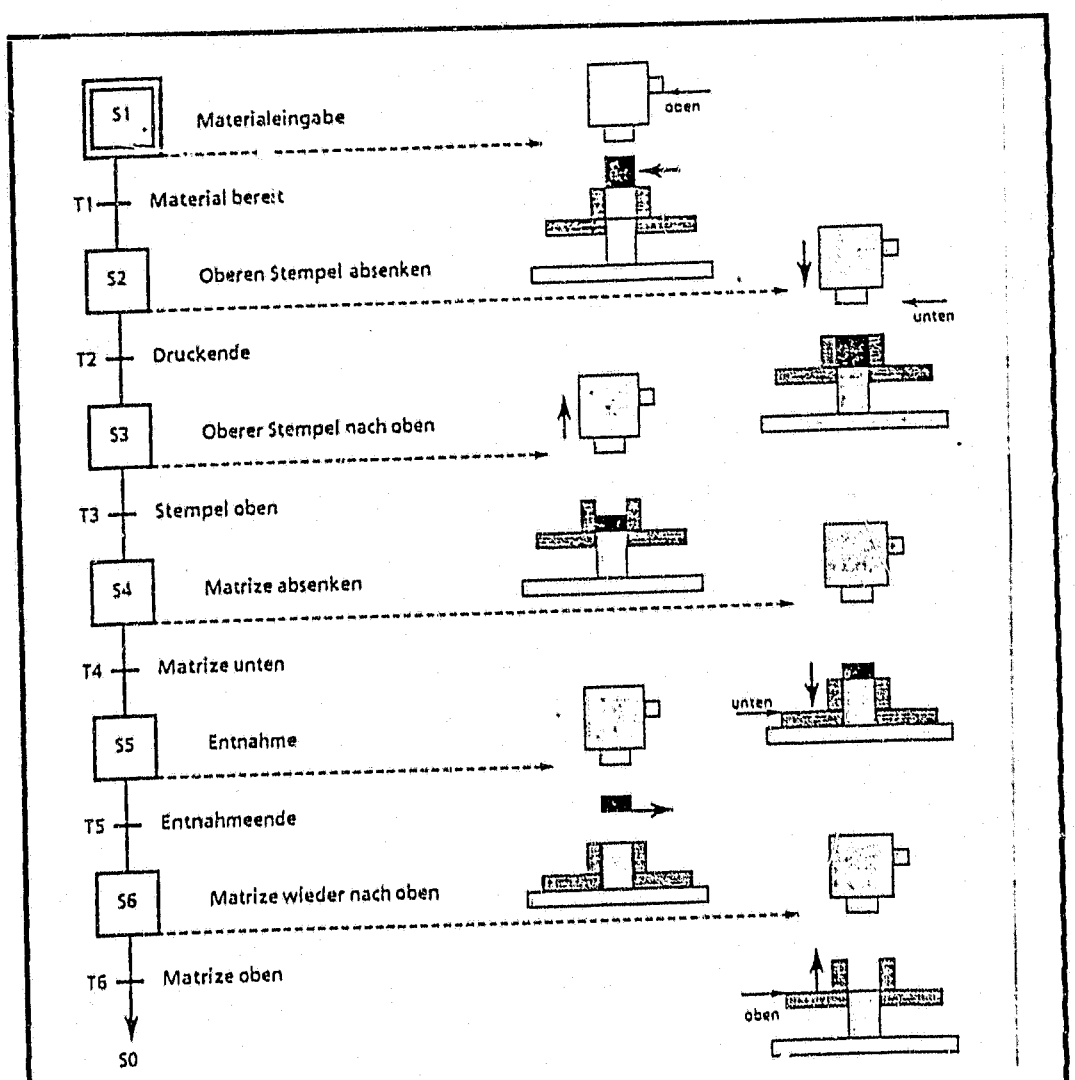


Fig. 3.9(a) Grafcet - example (overview)

(Leidigkeit, undated)

-3.24- Specification system overview

This system is based on Petri nets but does not use the tokens to evaluate the dynamics of the designed system. It only uses the structure of Petri nets to take care of concurrency.

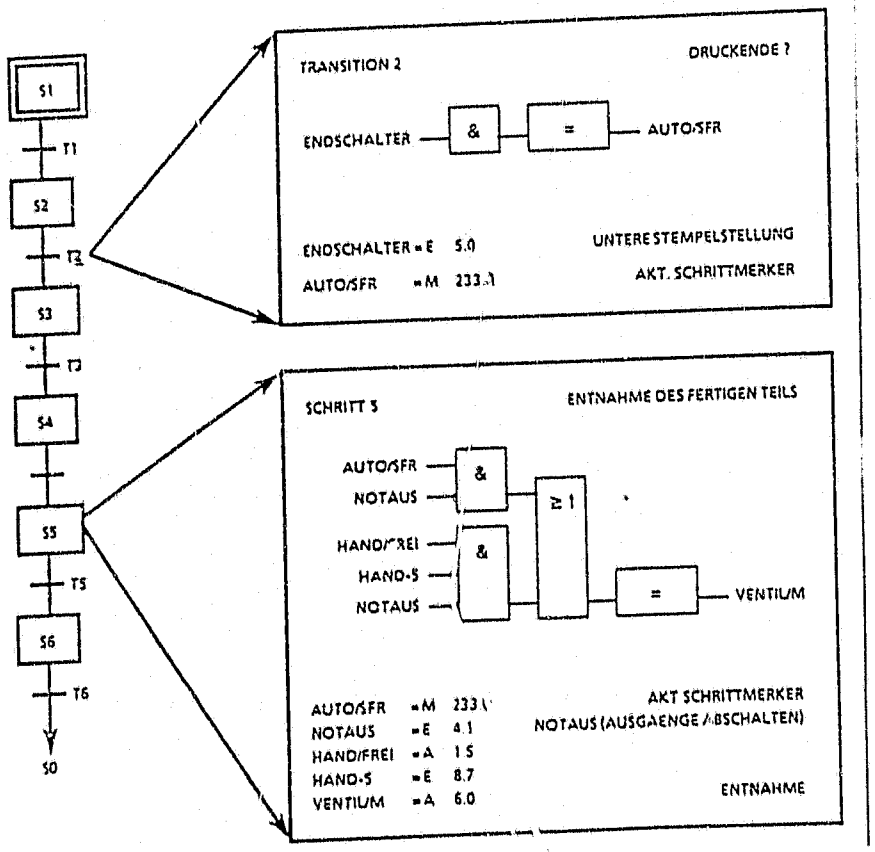


Fig. 3.9(b) Grafcet - example (detail)

(Leidigkeit, undated)

A practical system using this methodology is described by Leidigkeit (undated).

3.3 Conclusion

The following important factors emerged from all the systems that were considered:

- specifying the requirements of a system is difficult because it must communicate to the user and the designer and they do not talk the same language.
- modelling of the environment is most important as this is where the changes occur.
- quick prototyping is an advantage. It is much easier for the user to specify if a prototype is available.
- a top-down (hierarchical) approach to deal with complexity is beneficial in the design process.

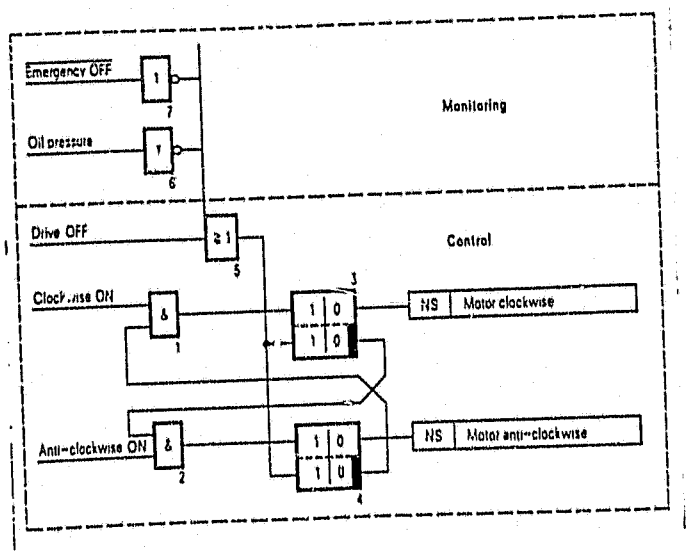


Fig. 3.9(c) Grafcet - example (sequence control)

(Siemens, undated)

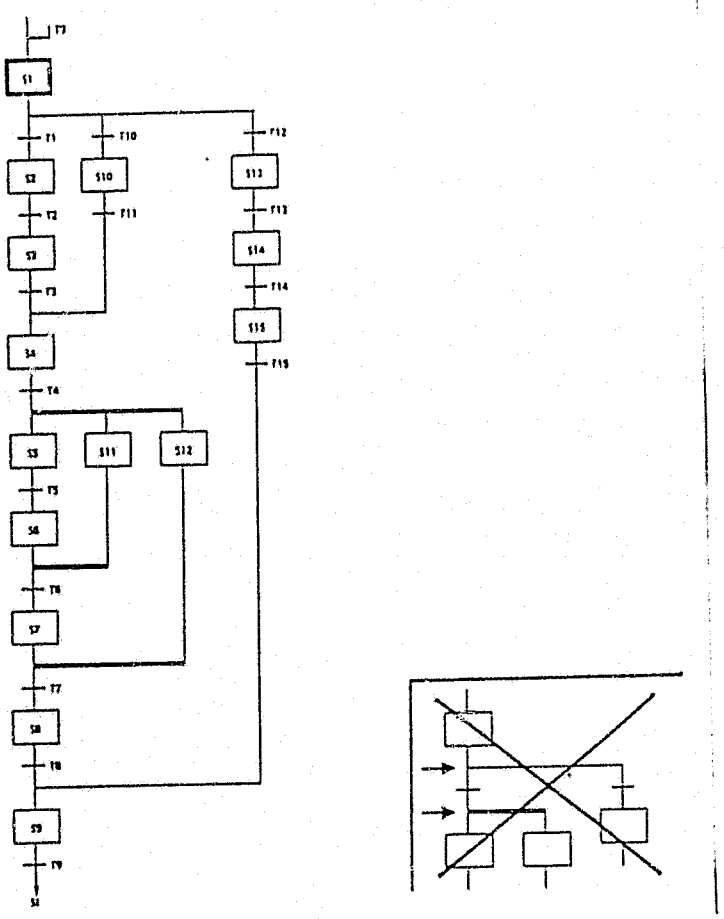


Fig. 3.9(d) Grafcet - example (concurrency)

(Leidigkeit, undated)

The systems discussed are all different. Each one has a different application in mind and caters for a different phase in the life cycle. One could compare the various methods to different criteria. The following could, for example, be compared:

- data dominance v. control dominance systems
- sequential v. parallel processes
- verification of specification
- mathematical base of methodology

- applicability to process control systems
- applicability to the different phases of the life cycle.

Different criteria may be used for each of these factors stated. It is, however, felt that an elaborate comparison will not produce any new results since the comparisons were done by Ramamoorthy (1978) and Ludewig (1978). The comparison in table 3.1 has only been done on the base that is used by the system.

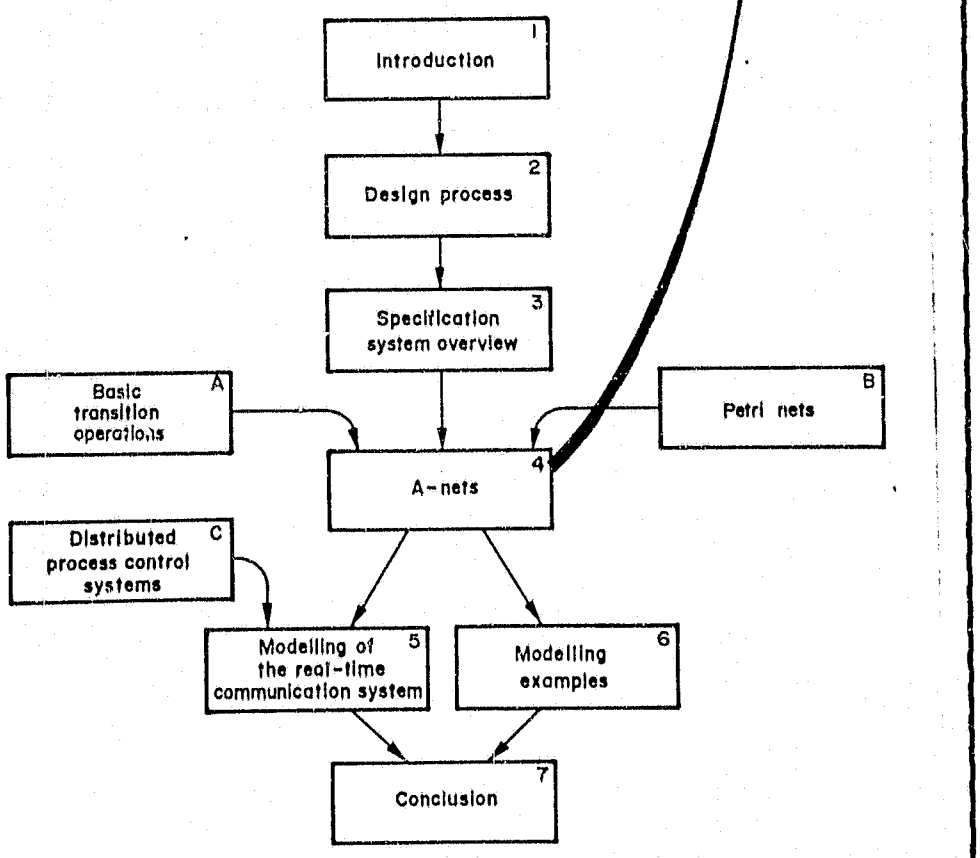
Paragraph	System	Base
3.2.1	SADT	data flow, control flow
3.2.2	PSL/PSA	----
3.2.3	SREM	stimulus response
3.2.4	MASCOT	cooperating parallel processes
3.2.5	PAISLey	cooperating parallel processes
3.2.6	FSM	state machines
3.2.7	ESPRESO	computer language
3.2.8	EPOS	----
3.2.9	RLP	computer language
3.2.10	Motus-Quirk	cooperating parallel processes
3.2.11	Grafcet	Petri nets

Table 3.1 Comparison of specification systems.

As can be seen from the table, a sound mathematical base is absent in most cases. The systems with a

mathematical base such as FSM and Grafcet, do not use it in the predictive sense. SREM has elaborate simulation capabilities with which it is possible to gain confidence for the design by exercising it, but which cannot be used to predict the outcome. The Grafcet methodology uses the static capabilities of Petri nets but not the dynamic or predictive capabilities. The proposed A-nets of chapter 4 provide a base for a design methodology that can overcome these deficiencies.

4 A-NETS



C H A P T E R 4

A-NETS

4.1 Introduction

In chapter 3 it was shown that one of the deficiencies of specication systems is the lack of a mathematical base for the design methodology used. Chapter 2 illustrated that the design process is complex and difficult to model. It was indicated that a design methodology should be based on a hierarchical approach to allow for the limitations of the attention span of the human designer. A graphical representation is one of the other important factors that helps the designer to cope with complexity as well as to create innovative designs.

The mathematical base of A-nets is developed in this chapter. It is proposed that these A-nets can be used as a base for design methodology. They are related to Petri nets and share the same mathematical foundation. A hierarchical approach is also possible by using subnets. A-nets and Petri nets can be represented graphically as well as mathematically, as has already been mentioned. It is therefore proposed that a design

-4.2- A-nets

methodology based on A-nets fulfils the needs of a design methodology.

4.1.1 Advantages of using a Petri net based system

A-nets are based on augmented Petri nets. Petri nets are discussed in detail by Peterson (1981). Appendix B provides a summary of the relevant Petri net theory. The following features of Petri nets make them well-suited for the modelling of complex systems:

- hierarchical representation
- graphical representation
- mathematical representation
- representation of concurrent processes

Graphical representation allows a visual representation of the structure of the problem. It was illustrated in chapter 2 that a graphical representation is desirable for complex problem representation. It results in the use of both sides of the brain.

Mathematical representation allows analysis of the dynamics of the problem. Many problems to be modelled have parallel parts and Petri nets are well-suited to deal with concurrency.

A large amount of theoretical work has been done to analyse Petri nets for conditions like deadlock, for

example (Hack, 1976), (Brauer, 1979) and (Girault, 1981).

4.1.2 Petri net deficiencies

Petri net deficiencies are the following:

- no zero testing ability
- transitions fire in zero time
- random firing of enabled transitions
- the token represents only binary values.

The first three deficiencies are rectified by extensions to Petri nets and are recorded in appendix B. They are the following:

- inhibitor arcs
- timed transitions
- priorities added to transitions.

Activator arcs are also added to be able to produce a more understandable and clear graphical representation.

Tokens represent only binary values because in Petri nets all tokens are exactly the same and they are undistinguishable. Berthelot (1982) introduced tokens with attributes. The net, however, stayed the same and the graphical representation became complex. These tokens with attributes fulfil the need to represent certain theoretical problems.

-4.4- A-nets

4.1.3 A-net proposal

A-nets are augmented Petri nets where tokens with attributes (A-tokens) are used. This chapter will introduce these A-tokens as well as the special transitions that are used to operate on them in an informal way. The formal definition is followed by an example of the use of A-nets on a FIFO buffer.

4.2 Informal description

The introduction of tokens with attributes (A-tokens) constitutes the major difference between A-nets and Petri nets. The attributes of an A-token are typed and have a value. An attribute may be as simple as a single integer or it may be a structured entity like a message. A Petri net token is indicated by a dot residing in a place as is shown in Fig. 4.1(a). As is indicated, it is only possible to model binary values. Fig. 4.1(b) shows an A-token with attributes, $a_1,..a_q$. Each attribute has a value.

(a.) Binary token (b.) A-token

Fig. 4.1 Symbols for tokens

-4.5- A-nets

The transfer transition used in Petri nets acts on a token as a complete entity. Special transitions are defined that will, on firing, have an influence on the attribute make-up of an A-token. Various new transitions will be described in the next paragraph, for example:

- the merging of attributes in a token
- the separation of an A-token into different A-tokens
- operations on the value of attributes.

The places are defined in such a way that they may only contain A-tokens of a certain type.

A-nets therefore allow the modelling of data flow as well as of the flow of control. Data flow modelling is difficult to achieve in normal Petri nets.

4.3 Formal description of A-nets

The following parts of A-nets are formally described:

- Definition of the A-net structure
- Definition of the marking
- Definition of A-tokens
- Definition of transitions
- Definition of transition types.

-4.6- A-nets

4.3.1 Definition of the A-net structure

The A-net structure is defined in the same way as the Petri net structure. Definitions B.1 and B.5 is used as they were defined for Petri nets.

4.3.2 Definition of the marking

The marking of an A-net is defined in the same way as for a Petri net. Definition B.2 is used as it was defined for Petri nets.

4.3.3 Definition of A-tokens

This is the area where A-nets and Petri nets differ. Tokens for Petri nets are all alike. A-tokens consist of attributes. The value of each attribute as well as the type of attribute may differ in different A-tokens.

Definition 4.1. The marking μ of an A-net, C, consists of A-tokens. An A-token consists of attributes. For a specific net, a set of attributes $A = \{a_1, a_2 \ldots\ldots\ldots a_q\}$ is defined.

It is convenient to consider an A-token as a vector as it will facilitate the formulation of transitions.

-4.7- A-nets

Definition 4.2. An A-token of an A-net, C, is a vector:

$\lambda = \epsilon_1 a_1 + \epsilon_2 a_2 + \ldots + \epsilon_q a_q$,

where ϵ_h (h = 1 to q) is the value of the attribute a_h.

This value ϵ_h of attribute a_h is of a specified type and is a function of its definer, α_h, and of its magnitude, ∇_h.

Definition 4.3. The value ϵ_h of an attribute a_h is a function of α_h and ∇_h.

$\epsilon_h = \alpha_h \nabla_h$,

where α_h is either ν when a_h is defined for this token, or ω when a_h is not defined, and ∇_h is the magnitude of the value of attribute a_h.

To be able to do calculations, the following three values are to be defined:

- unit value
- nil value
- don't care value.

Definition 4.4. A unit value ν is defined in such a way that:

$\nu * \nabla_h = \nabla_h$.

-4.8- A-nets

Definition 4.5. A nil value ω is defined in such a way that:

$\omega * \nabla_h = \omega$

$\omega + \nabla_h = \nabla_h$.

Definition 4.6. A don't care value σ is defined in such a way that:

$\sigma * \nabla_h = \nabla_h$

$\sigma + \nabla_h = \nabla_h$

and $\sigma < \nabla_h$

for all ∇_h.

The places in an A-net are defined to contain a certain type of A-token. The type of A-token is defined by the allowable attributes. This typing is similar to data-typing in a language like Pascal.

Definition 4.7. A set of A-token types, B, is defined for an A-net such that

$B = \{B_1, B_2 \ldots\ldots B_v\}$

and:

$$\begin{bmatrix} B_1 \\ B_2 \\ . \\ . \\ B_v \end{bmatrix} = \begin{bmatrix} \alpha_{11}\ \alpha_{12} \ldots\ldots \alpha_{1q} \\ \alpha_{21}\ \alpha_{22} \ldots\ldots \alpha_{2q} \\ \qquad . \\ \qquad . \\ \alpha_{v1}\ \alpha_{v2} \ldots\ldots \alpha_{vq} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ . \\ . \\ a_q \end{bmatrix},$$

where $\alpha_{hd} = v$ when the attribute is assigned to A-token type B_h,
and $\alpha_{hd} = \omega$ when the attribute is not assigned, for $h = 1$ to v and $d = 1$ to q.

Places form subsets that may only contain A-tokens of a certain type.

Definition 4.8. A place $p_i \in P$ may only contain A-tokens of a specified A-token type:

$$P = P_{B1} \cup P_{B2} \cup \ldots \cup P_{Bv}$$
$$= \{p_1, p_2 \ldots p_n\},$$

where P_{Bh}, $h = 1$ to v, is the subset of places that may contain A-tokens of type B_h. Fig. 4.2 shows the symbol to be used for places with A-tokens.

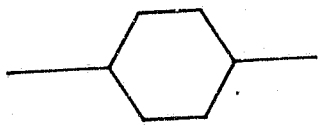


Fig. 4.2 Symbol for A-net place

Tokens for Petri nets may be considered as binary tokens in A-nets. A binary token is present or not present, and the attributes are all of the don't care type. Places where binary tokens reside will be indicated with circles.

Definition 4.9. A binary token is defined as a vector λ where:

$\epsilon = \sigma$ for all a_h.

4.3.4 Definition of transitions

The transitions of an A-net are more complex than those of a Petri net. A-net transitions must operate on the attributes of tokens. A transition may be in any one of the following states:

- disabled
- μ-enabled
- a-enabled
- firing.

These states are illustrated in the state diagram of Fig. 4.3.

Fig. 4.3.

The definition for a μ-enabled transition is similar to definitions B.3 and B.6. However, the definition to replace definition B.3 is given to clarify the differences.

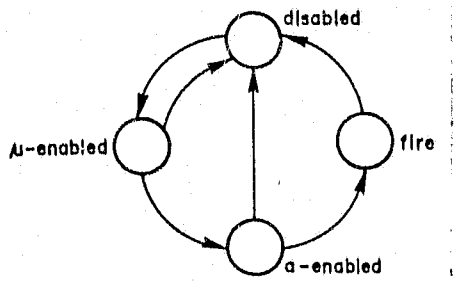


Fig. 4.3 Transition state diagram

-4.11- A-nets

Definition 4.10. A transition $t_j \in T$ in a marked A-net with marking μ is μ-enabled to fire if for all $p_i \in P$

$\mu(p_i) \geq \#(p_i, I_N(t_j))$

In order to define the tokens produced after firing, an a-enabling function as well as a token-produced function are defined. Each transition type will have a specified a-enabling and token-produced function.

Definition 4.11. The a-enabling function, $F_E(t_j)$, of a transition $t_j \in T$ in an A-net is:

$F_E(t_j) = F_E(\lambda_1, \lambda_2 \ldots \lambda_w)$,

where λ_b (b = 1 to w) is the A-token on input b of the transition t_j that caused it to be μ-enabled. The a-enabling function is a binary function with true or false values. $F_E(t_j)$ = true when $B_g(\epsilon_{1g}, \epsilon_{2g} \ldots \epsilon_{wg})$ is true, for attribute g(g = 1 to q) and B_g is a binary function defined for each transition type.

The new marking after a transition has fired is defined in the same way as for Petri nets. Definitions B.4 and B.7 therefore apply for A-nets.

The attribute make-up of the A-token produced when a transition fires depends on the type of transition. A transformation function is defined for each type of transition.

Definition 4.12. The A-token produced (λ_0) when a transition $t_j \in T$ in an A-net fires, is:

$\lambda_0 = T_T(\lambda_1, \lambda_2 \ldots . \lambda_w)$,

where λ_b (b = 1 to w) is the A-token on input b of the transition t_j and T_T is the transformation function of transition t_j. Each transition type will have a transformation function defined in such a way that the output A-token:

$\lambda_0 = \beta_1 a_1 + \beta_2 a_2 + \ldots . \beta_q a_q$ ——————————(4.1),

is a function of the input A-tokens:

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ . \\ \lambda_w \end{bmatrix} = \begin{bmatrix} \epsilon_{11}\ \epsilon_{12} \ldots \ldots \epsilon_{1q} \\ \epsilon_{21}\ \epsilon_{22} \ldots \ldots \epsilon_{2q} \\ . \\ \epsilon_{w1}\ \epsilon_{w2} \ldots \ldots \epsilon_{wq} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ . \\ a_q \end{bmatrix},$$

and $\beta_q = A(\epsilon_{1q}, \epsilon_{2q} \ldots \epsilon_{wq})$ and A is a specified function for a transition type.

4.3.5 Definition of transition types

The following useful transitions are defined:

- Transfer transition, T_N
- Merge transition, T_M
- Seperate transition, T_B
- Arithmetic operation transition, T_A
- Conditional firing transition, T_C
- Select transition, T_S
- Subnet transition.

For each transition type the a-enabling function as

well as the token produced will be defined. Transfer and merge transitions form the basic operations of the transitions and is explained in appendix A.

Transfer transition

Definition 4.13. A transfer transition, T_N, has an a-enabling function $F_E(T_N)$ always true, and

$$\beta_g = \prod_{h=1}^{w} \epsilon_{hg} \quad \text{for} \quad g = 1 \text{ to } q$$

in the token-produced equation 4.1.

Take as an example the transfer transition t_j in Fig. 4.4. Three input tokens λ_1, λ_2 and λ_3 are present. If the input tokens have four attributes:

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} \epsilon_{11} & \epsilon_{12} & \epsilon_{13} & \epsilon_{14} \\ \epsilon_{21} & \epsilon_{22} & \epsilon_{23} & \epsilon_{24} \\ \epsilon_{31} & \epsilon_{32} & \epsilon_{33} & \epsilon_{34} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix},$$

then the token produced will be:

$$\lambda_0 = \begin{bmatrix} zz_A & zz_B & zz_C & zz_D \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

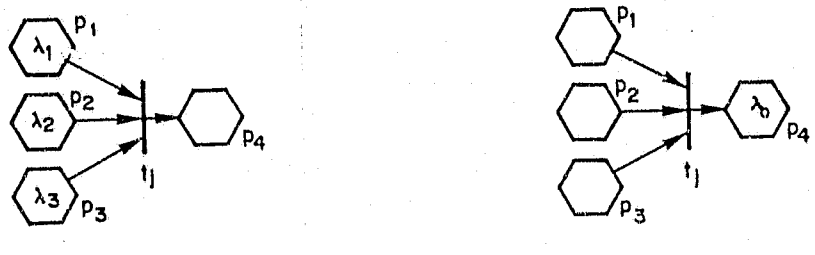
Where:

$$zz_A = \epsilon_{11}\epsilon_{21}\epsilon_{31}$$

$$zz_B = \epsilon_{12}\epsilon_{22}\epsilon_{32}$$

$$zz_C = \epsilon_{13}\epsilon_{23}\epsilon_{33}$$

$$zz_D = \epsilon_{14}\epsilon_{24}\epsilon_{34}.$$



(a.) Enabled (b.) After firing

Fig. 4.4 Transfer transition

When λ_1, λ_2 and λ_3 are all binary tokens then all ϵ would be σ (don't cares). This would imply that the token produced:

$$\lambda_0 = \begin{bmatrix} \sigma & \sigma & \sigma & \sigma \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}.$$

This is a binary token. This is the case as it would have been for Petri nets.

-4.15- A-nets

If λ_1 and λ_2 are binary tokens, then ϵ_{11}, ϵ_{12}, ϵ_{13}, ϵ_{14}, ϵ_{21}, ϵ_{22}, ϵ_{23}, ϵ_{24} is equal to σ.
The output token produced would then be:

$$\lambda_0 = \begin{bmatrix} \epsilon_{31} & \epsilon_{32} & \epsilon_{33} & \epsilon_{34} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

$$= \lambda_3 .$$

This implies that the A-token λ_3 has been transferred with the binary tokens λ_1 and λ_2 present.

Merge transition

Definition 4.14. A merge transition, T_M, has an a-enabling function $F_E(T_M)$ always true, and

$$\beta_g = \sum_{h=1}^{w} \epsilon_{hg} \quad \text{for} \quad g = 1 \text{ to } q$$

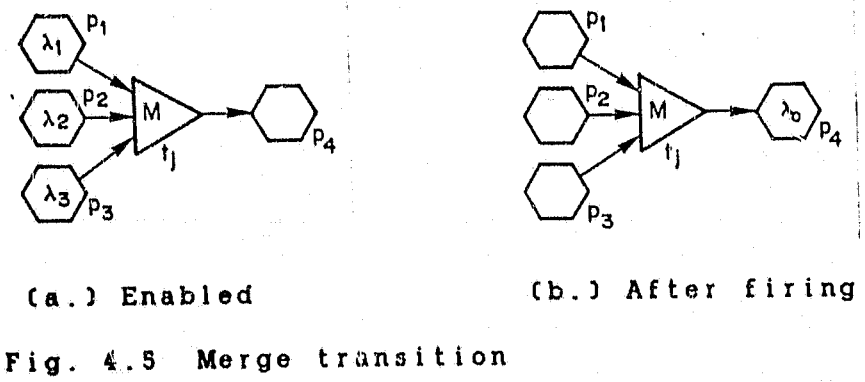
in the token-produced equation 4.1.

Take as an example the merge transition t_j in Fig. 4.5. Three input tokens, namely λ_1, λ_2 and λ_3 are present. If the input tokens have three attributes,

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} \epsilon_{11} & \epsilon_{12} & \epsilon_{13} \\ \epsilon_{21} & \epsilon_{22} & \epsilon_{23} \\ \epsilon_{31} & \epsilon_{32} & \epsilon_{33} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix},$$

then the token produced will be:

$$\lambda_0 = \begin{bmatrix} \epsilon_{11}+\epsilon_{21}+\epsilon_{31} & \epsilon_{12}+\epsilon_{22}+\epsilon_{32} & \epsilon_{13}+\epsilon_{23}+\epsilon_{33} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$



(a.) Enabled (b.) After firing

Fig. 4.5 Merge transition

If each input token only had one attribute

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} \epsilon_{11} & \omega & \omega \\ \omega & \epsilon_{22} & \omega \\ \omega & \omega & \epsilon_{33} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix},$$

then:

$$\lambda_0 = \begin{bmatrix} \epsilon_{11} & \epsilon_{22} & \epsilon_{33} \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}.$$

This is the merging of attributes into one token that was required.

If the input tokens were:

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} \epsilon_{11} & \omega \\ \omega & \epsilon_{22} \\ \sigma & \sigma \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix},$$

then the output token will be:

$$\lambda_0 = \begin{bmatrix} \epsilon_{11} & \epsilon_{22} \end{bmatrix} \begin{bmatrix} a_1 & a_2 \end{bmatrix}.$$

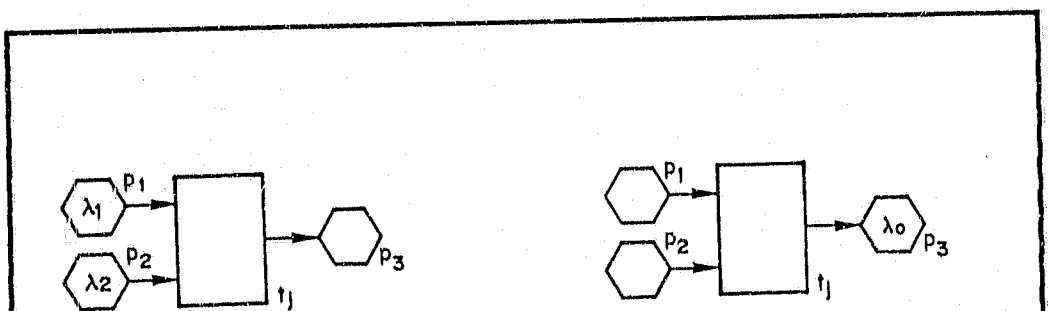
This shows that two tokens λ_1 and λ_2 are merged when a Boolean token, λ_3, is present.

Arithmetic transition

Definition 4.15. An arithmetic transition, T_A, has an a-enabling function $F_E(T_A)$ always true, and

$$\beta_g = A(\epsilon_{hg}) \quad \text{for } g = 1 \text{ to } q$$

in the token-produced equation 4.1. A is an arithmetic function.



(a.) Enabled (b.) After firing

Fig. 4.6 Arithmetic transition

Fig. 4.6 presents an arithmetic transition t_j with input places p_1 and p_2 as an example.

Let:

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} \epsilon_1 \\ \epsilon_{21} \end{bmatrix} \begin{bmatrix} a_1 \end{bmatrix}, \quad \text{then}$$

$$\lambda_0 = A(\epsilon_{11}, \epsilon_{21})a_1 .$$

If A(x,y) was defined to be x+y,

then:

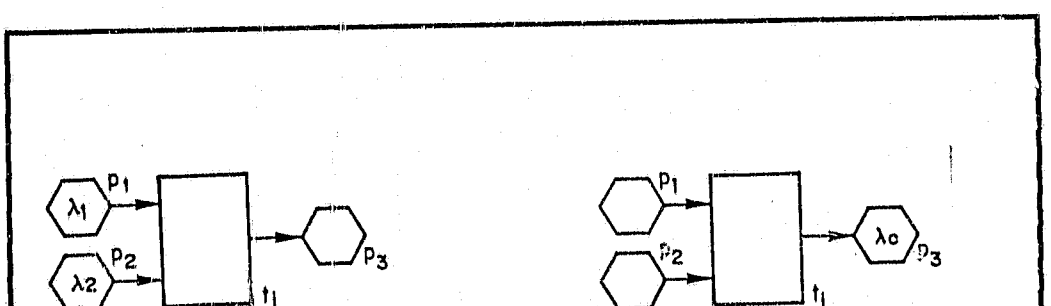
$$\lambda_0 = (\epsilon_{11}+\epsilon_{21})a_1 .$$

This gives the capability of arithmetic functions.

Definition 4.16. A conditional firing transition, T_C, has an a-enabling function, $F_E(T_C)$ = true when $\beta_g(\epsilon_{hg})$ = true (for g = 1 to q), where β_g is a binary function defined for each attribute a_g.

$\beta_g = \sigma$ if $F_E(T_c)$ = true (for g = 1 to q)

in the token-produced equation 4.1.



(a.) Enabled (b.) After firing

Fig. 4.7 Conditional firing transition

For example, in Fig. 4.7, if the input tokens are:

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 1 & \omega & \omega \\ 2 & \omega & \omega \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix},$$

and B was defined in such a way that:

$B(\epsilon_{11}, \epsilon_{21})$ = true when $\epsilon_{21} > \epsilon_{11}$,

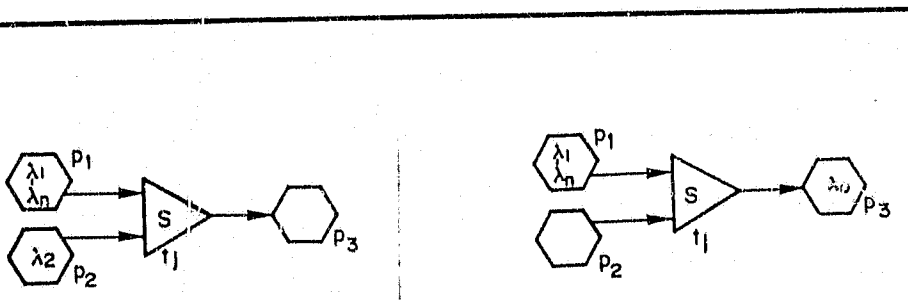
then $F_E(t_j)$ = true and

$$\lambda_0 = (\sigma\ \sigma\ \sigma)\ (a_1\ a_2\ a_3) .$$

If $F_E(t_j)$ = false,
then no output token would have been produced.

Select transition

Definition 4.17. A select transition, T_S, is a two input transition with an a-enabling function $F_E(T_S)$ = true when
$B(\epsilon_{1g}) = \epsilon_{2g}$ for g = 1 to q and
$\beta_g = \epsilon_{1g}$ when $\epsilon_{1g} < \epsilon_{2g}$
and $\beta_g = \epsilon_{2g}$ when $\epsilon_{2g} \geq \epsilon_{1g}$.



(a.) Enabled (b.) After firing

Fig. 4.8 Select transition

For example if:

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_n \end{bmatrix} = \begin{bmatrix} \gamma & \epsilon_{12} & \epsilon_{13} \\ \gamma & \sigma & \sigma \\ \epsilon_{n1} & \epsilon_{n2} & \epsilon_{n3} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

in Fig. 4.8, then:

$$\lambda_0 = \begin{bmatrix} \gamma & \epsilon_{12} & \epsilon_{13} \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}.$$

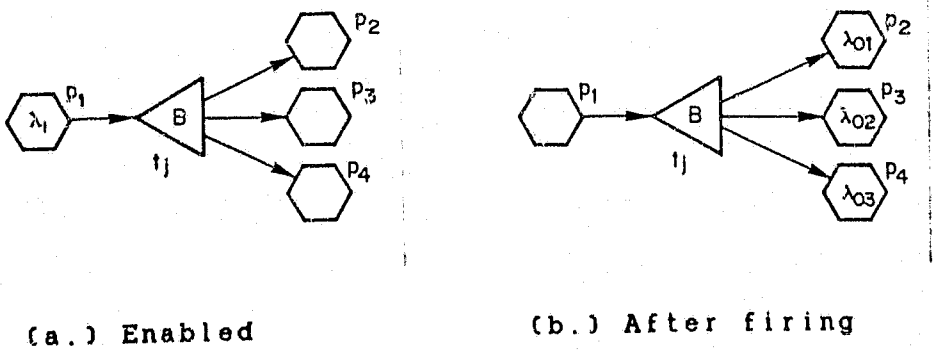
A mask λ_2 was provided in p_2 to select one of the many tokens in $p_1(\lambda_1, \lambda_n)$. The mask in this example was set

up to select a value γ for attribute a_1. Attributes a_2 and a_3 were set to don't cares (σ). The token selected was λ_1 because its value for attribute a_1 matched that of the mask.

Subnet transition

To be able to deal with complex A-nets, a subnet capability is defined. The method of producing a hierarchical system to combat complexity was stressed in chapters 2 and 3.

Definition 4.18. A subnet transition is a subnet that will execute on the conditions at the input.



(a.) Enabled (b.) After firing

Fig. 4.9(a) Seperate subnet (overview)

One could define a seperate transition but it would be complex and it is easier dealt with as a subnet. Fig. 4.9 shows a subnet that will seperate an A-token λ_1, into its three parts λ_{01}, λ_{02} and λ_{03}. Fig. 4.9(a) shows the general symbol for a separating transition while Fig. 4.9(b) shows the detail of the

subnet that will execute.

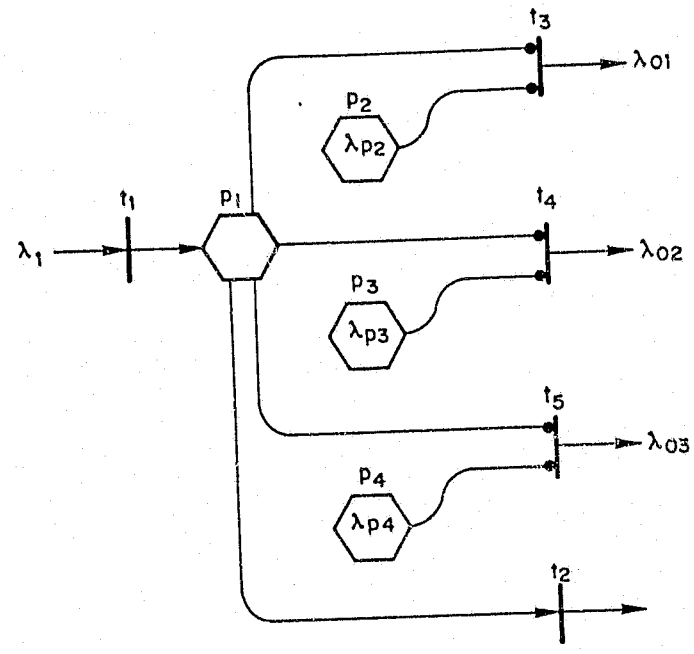


Fig. 4.9(b) Seperate subnet (detail)

The A-token on the input is:

$\lambda_1 = \epsilon_{11}a_1 + \epsilon_{22}a_2 + \epsilon_{33}a_3$ and it consists of three attributes. The seperating transition must produce three A-tokens each with only one attribute:

$$\begin{bmatrix} \lambda_{01} \\ \lambda_{02} \\ \lambda_{03} \end{bmatrix} = \begin{bmatrix} \epsilon_{11} & \omega & \omega \\ \omega & \epsilon_{22} & \omega \\ \omega & \omega & \epsilon_{33} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} .$$

The input A-token is placed in p_1. Places p_2, p_3 and p_4 contain a separating mask with a unit value for the attributes to be seperated and nil values for the

attributes that are not to be transferred by t_3, t_4 and t_5. Transition t_2 is required to remove the input token to the token bucket after the seperation process has been completed. A detailed discussion of this subnet is provided in appendix A.

4.4 Design methodology

A-nets are used in the following way as an aid in the design of real-time computer control systems:

- the plant is modelled
- the control system is modelled by using a state diagram approach. This state diagram is implemented by means of A-nets.
- analysis is done by using the mathematical methods developed for Petri nets.

The design methodology is explained by means of examples in chapters 5 and 6.

4.5 Example of an A-net

A simple example will be used to demonstrate some of the capabilities and the use of A-nets. The example of a buffer in appendix B is extended to be a FIFO buffer for this example.

The graphical representation of the A-net model is

given in Fig. 4.10. The data to be entered is submitted via transition t_1. It is saved in the buffer, p_2, with a sequence number supplied when t_2 fired. Requests for output is done via transition t_4. The correct one is selected with a sequence number when t_3 fires.

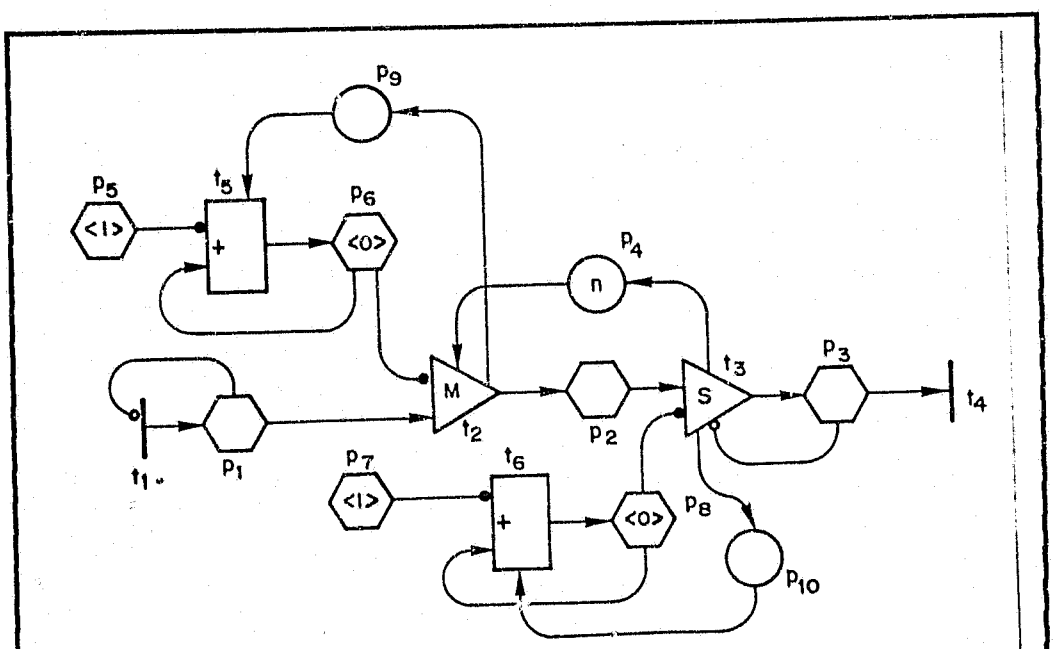


Fig. 4.10 FIFO buffer

A formal mathematical formulation follows:

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

$P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}\}$

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

$I_N(t_2) = \{p_1, p_4\}$

$I_N(t_3) = \{p_2\}$

$I_N(t_4) = \{p_3\}$

$I_N(t_5) = \{p_6, p_9\}$

$I_N(t_6) = \{p_8, p_{10}\}$

$I_A(t_2) = \{p_6\}$

$I_A(t_3) = \{p_8\}$

$I_A(t_5) = \{p_5\}$

$I_A(t_6) = \{p_7\}$

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

$I_I(t_3) = \{p_3\}$.

The attributes are:

$A = \{a_1, a_2\}$

where a_1 is a sequence number

and a_2 is the data to be stored.

The token types are defined as follows:

$$\begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix} = \begin{bmatrix} \nu & \nu \\ \nu & \omega \\ \omega & \nu \\ \omega & \omega \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} .$$

The places where the different token types may reside are defined as follows:

$P_{B1} = \{p_4, p_9, p_{10}\}$

$P_{B2} = \{p_5, p_6, p_7, p_8\}$

$P_{B3} = \{p_1\}$

$P_{B4} = \{p_2, p_3\}$.

The transitions:

$T_N = \{t_1, t_4\}$

$T_M = \{t_2\}$

-4.25- A-nets

$T_S = \{t_3\}$

$T_A = \{t_5, t_6\}$

Initial marking:

$\mu_0 = (0, 0, 0, n, \langle 1,0\rangle, \langle 0,0\rangle, \langle 1,\sigma\rangle, \langle 0,\sigma\rangle, 0, 0)$,

where n+2 = size of the buffer.

Two loops exist to generate sequence numbers, namely loop p_5, t_5, p_6, p_9 for the input sequence number and p_7, t_6, p_8, p_{10} for the exit select sequence number. Only the input sequence number generator will be explained. A request for a new sequence number will be placed in p_9, after which transition t_5 will be enabled to fire. On firing it will use the old sequence number in p_6 and the constant 1 in p_5 will be added to it. The new updated sequence number will be in p_6 and the request for a new sequence number in p_9 will be consumed.

New data will be presented to t_1 (not shown). If p_1 is empty it will be allowed into p_1. If the buffer (p_2) still has some space, p_4 will not be empty and t_2 will be enabled to fire. It will merge the data from p_1 with the sequence number from p_6 and place it in the buffer, p_2. It will consume a token from p_4 indicating that another space in the buffer has been taken. It will also produce a request for a new sequence number in p_9.

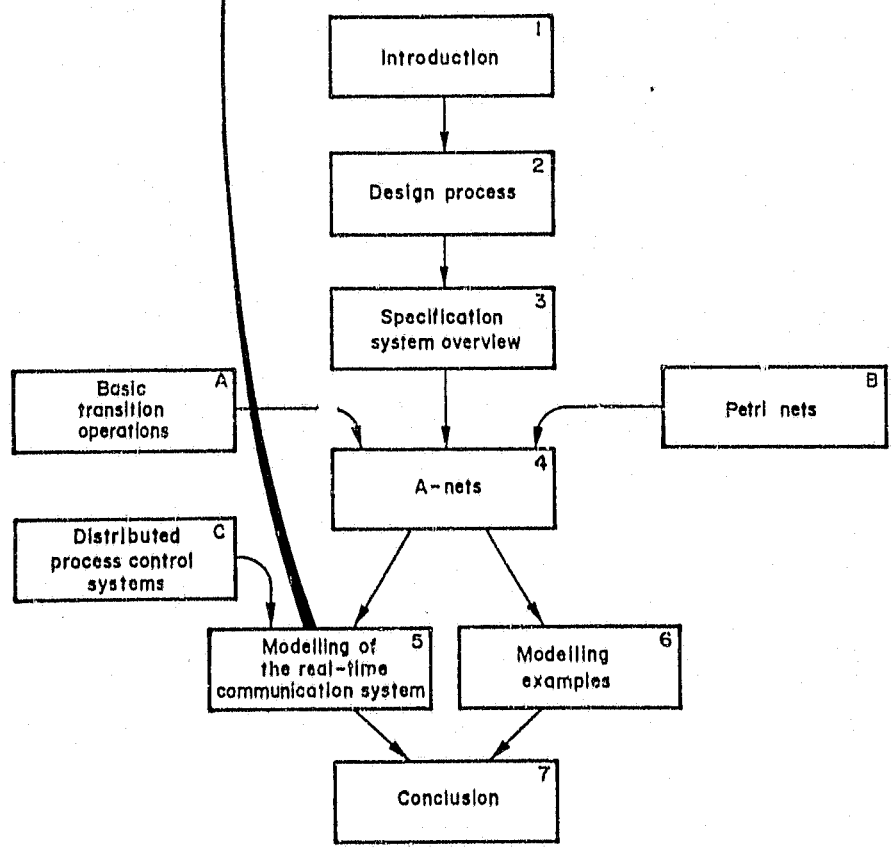
Requests for data from the buffer are done via t_4 (not shown). The data in p_3 will then be transferred. If p_3 is empty and data is present in the buffer, p_2, t_3 will be enabled to fire. The sequence number in p_8 is used as a mask to select the proper data piece out of the buffer p_2 and place it in p_3. A token is sent to p_4 to indicate that a place is available for a new input. A request for a new sequence number is generated in p_{10}.

4.6 Conclusion

A formal definition of A-nets has been presented and explained by means of a simple example. Where Petri nets can only handle tokens with binary values, A-nets can deal with tokens with multi-attributes and each attribute may be multi-valued. Where Petri nets were useful to model sequencing type problems, the range of modelling has been extended by the introduction of A-nets, so that more complex data type problems can be dealt with.

The use of A-nets is demonstrated in chapters 5 and 6. Chapter 5 consists of the solving of a software analytical problem of a communication system in a distributed computer control system. Chapter 6 presents the application of A-nets to the design of a software engineering product as the computer control of an industrial system.

5 MODELLING OF THE REAL-TIME COMMUNICATION SYSTEM



C H A P T E R 5

MODELLING OF THE REAL-TIME COMMUNICATION SYSTEM

5.1 Introduction

Appendix C contains information on a proposed distributed computer control system (DCCS). This system was developed by Kopetz (1982) and MacLeod (1983) and forms part of a DCCS research project (Rodd, 1982).

The problem of consistency, coordination and synchronization was discussed by MacLeod (1983). In a distributed computer system, each node is dependent on the state of the other nodes. Because the states of nodes may change without the other nodes knowing it, inconsistency may occur. This problem is solved by sending a state message with a validity period assigned to it. The sender node now commits itself not to change state until the end of the validity period. At the end of this validity period, the message is removed from the system. This mechanism implies that a global real-time is available to all the nodes.

The message transfer primitives of this system, as

discussed in C.3.2, form part of the real-time communications system. In this chapter A-nets are used to model these message primitives. The sender, channel and receiver are modelled seperately. A combined model for the communications channel is then provided as a subnet.

5.2 Sender

The sender uses the output statement (C.1) to send a message. This statement is modelled in Fig. 5.1. The sender supplies the message m and the validity period v_p. The validity period is added to the real-time (r_s) at the sender in t_1 to create the validity time v. The validity time and the message m is merged in t_2 to create the message to be delivered in place p_2.

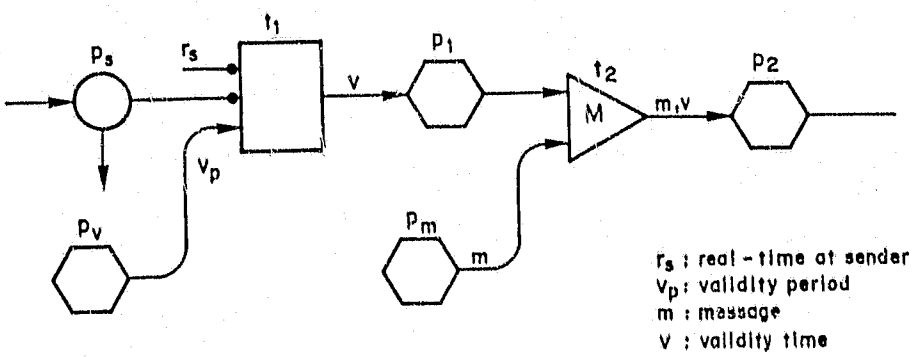


Fig. 5.1 Sender

5.3 Channel

The model of the channel is shown in Fig. 5.2. Two

independent paths can be traced through the model. The first, p_3, can be traced to t_{EM} for event messages and the second, p_{11}, to t_{SM} for state messages. The use of event and state information was discussed in appendix C.

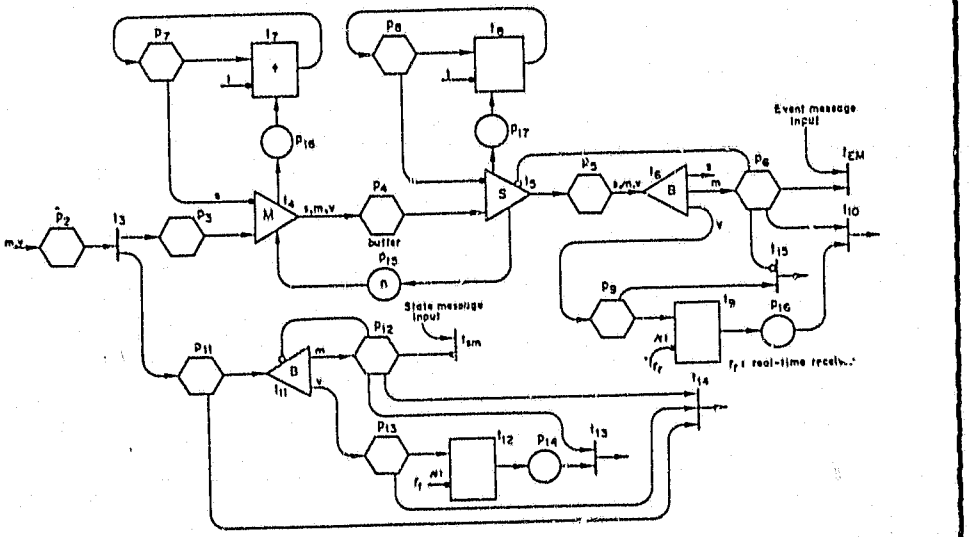


Fig. 5.2 Channel

5.3.1 Event messages

The event message in p_3 is merged with a sequence number when t_4 fires. Transition t_4 can only fire when the buffer is not full. This is done by having a place p_{15} with tokens to indicate the buffer length. The sequence numbers for merging with the message as well as for selecting, is done in the same way as was

discussed for the FIFO buffer in paragraph 4.4. Selection of the appropriate message out of the buffer takes place when t_5 fires. Transition t_5 may only fire when p_6 is empty. The message in p_6 is consumed by the receiver when t_{EM} fires. This message is also automatically removed by the system when the validity period has expired. This was one of the requirements stated in appendix C. The validity time is stripped from the message in t_6 and is then compared with the receiver real-time. As soon as the real-time exceeds the validity time, t_{10} fires to remove the message from p_6.

5.3.2 State messages

The state message in p_{11} is seperated into its message and validity time attributes in t_{11}. The message m is placed in p_{12} for further use. The validity time v is compared with the real-time in t_{12}. As soon as the real-time exceeds the validity time, the message in p_{12} is removed. The entering of a new message in p_{11} before the message in p_{12} has expired will generate the following sequence of events:

Transition t_{14} will fire to remove the old message and its associated validity times from p_{12} and p_{13}. With p_{12} vacant, t_{11} is enabled to fire to seperate the new message and its validity time.

5.4 Receiver

The receiver may employ various ways of using the message. This was explained in appendix C where the INPUT statements, as well as the distinction that was made between event and state messages, were discussed. The receiver decides to use a message as a state or event message. This will determine the hook to the channel model. State messages are hooked to t_{SM} and event messages to t_{EM} in Fig. 5.2. Transition t_{ch} is used as a general transition for both in the following discussion.

The following basic receiver constructs are modelled:

- simple input (statement C.2)
- input with timeout period (statement C.3)
- input with a filter expression (statement C.4).

5.4.1 Simple input

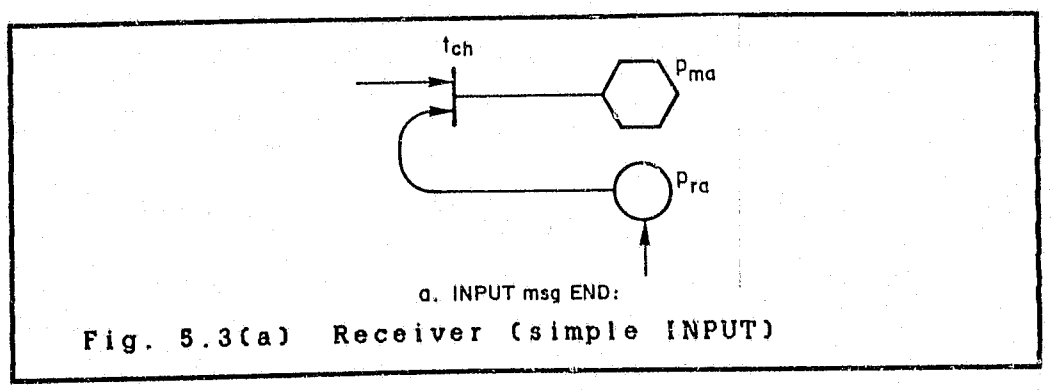


Fig. 5.3(a) Receiver (simple INPUT)

Fig. 5.3(a) models the simple input of statement C.2. The receiver waits until a message is available.

Transition t_{ch} fires and the message is placed in p_{ma} for the receiving module.

5.4.2 Input with timeout period

Fig. 5.3(b) models the input of statement C.3. The timeout period in p_{at} is added to the real-time in t_{b1} and compared with the real-time in t_{b2}. If a message is available or becomes available before this period has elapsed, t_{ch} fires and the message is available in p_{mb}. If the timeout period elapses, t_{b2} fires and a token is available in p_{nb}.

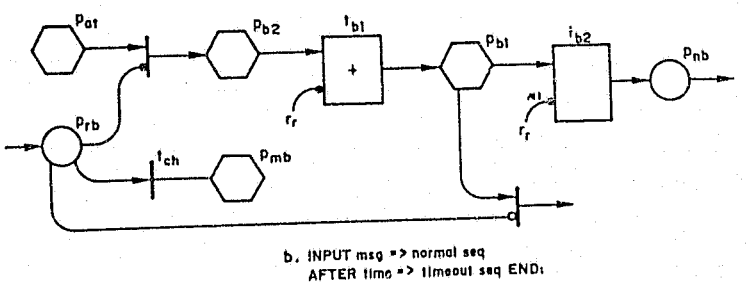


Fig. 5.3(b) Receiver (timeout period)

5.4.3 Input with a filter expression

Fig. 5.3(c) models the input statement C.4. For this discussion the following filter expression was taken as an example:

$$p_{fx} \rangle \text{ msg } \rangle p_{fy}$$

-5.7- The communication system

The model provides a hook p_{nc} if the message does not comply with the filter expression. This hook is not asked for in statement C.4.

The available message is placed in p_{c1} when t_{ch} fires. It is then compared with the user-supplied filter expression in t_{c1}. On compliance the message is placed in p_{mc}, otherwise a token is placed in p_{nc}.

Statement C.4 is the basic input statement. Statement C.5 is a combination of C.4 and timeout and statement C.6 a combination of C.4 and a filter expression. Different and more complex combinations are possible but that would only re-use the existing models in a combined way.

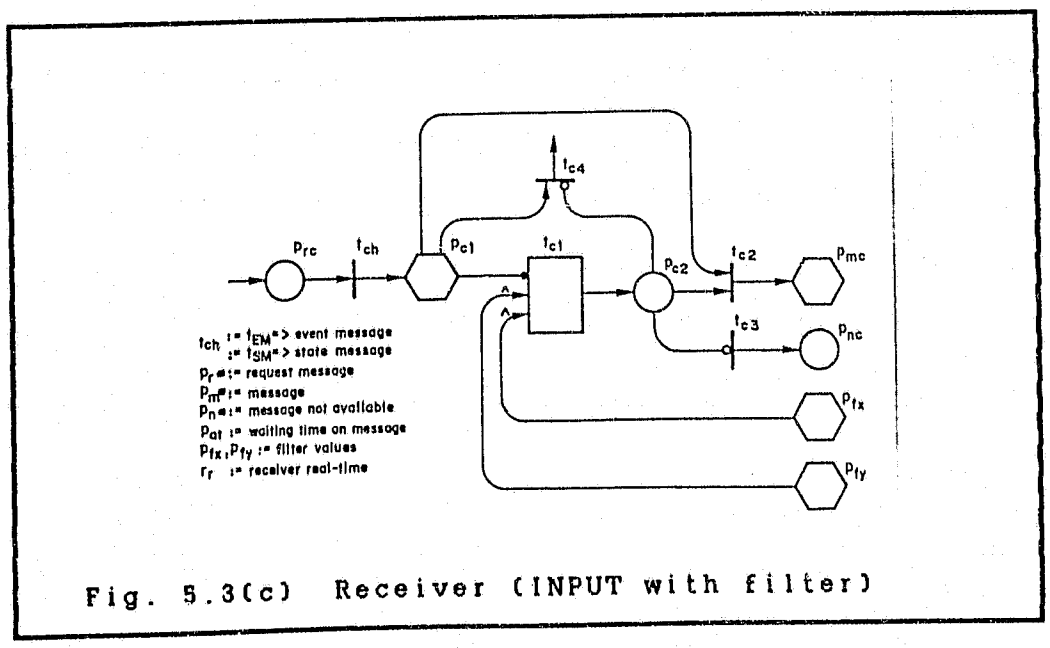


Fig. 5.3(c) Receiver (INPUT with filter)

5.5 Combined channel model

A combined channel model is developed in this section to provide a subnet that will be available to the user when modelling his system.

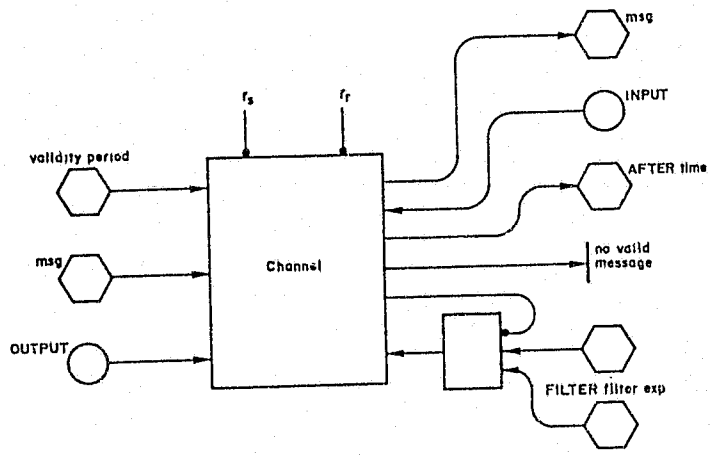


Fig. 5.4(a) Combined model (subnet)

Fig. 5.4 shows the subnet that is available to the user. Fig. 5.4(a) shows the interface and Fig. 5.4(b) the detail of the interface between the channel and the receiver. The detail for the sender and the channel is the same as was previously presented in Fig. 5.1 and Fig. 5.2.

The sender supplies the message and the validity period as part of the OUTPUT statement. The receiver may use any of the three basic methods as well as combinations thereof to retrieve a message from the

channel. The receiver must also decide whether it is a state or event message by placing a token in the appropriate place.

This universal interface supplied by the subnet to the receiver, and shown in Fig. 5.4(b), will now be discussed for the three cases of input statements.

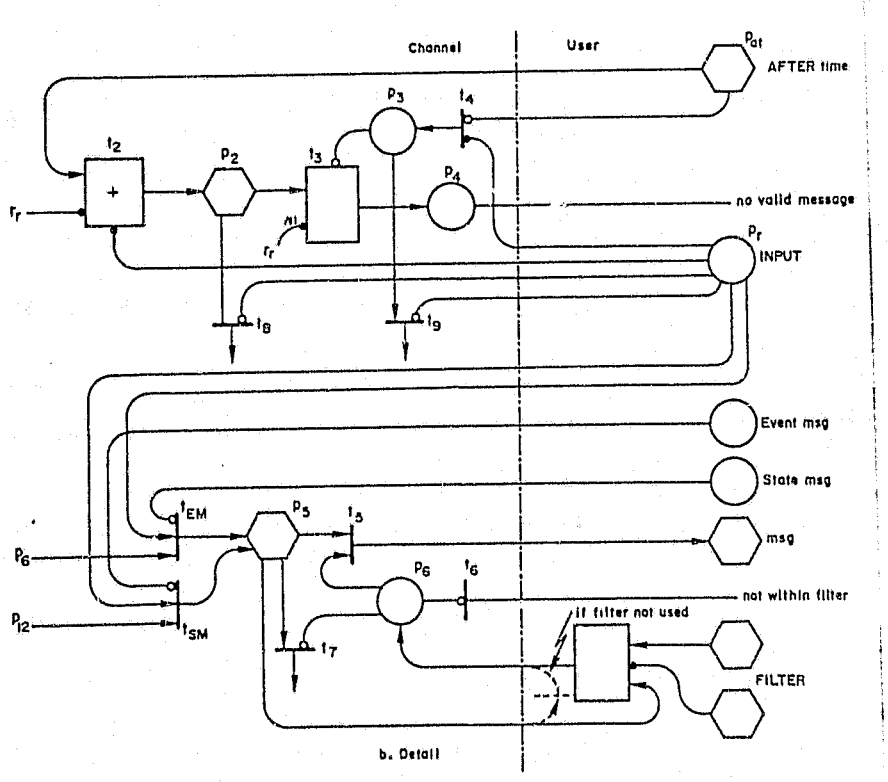


Fig. 5.4(b) Combined model (detail)

For INPUT statement C.2 the following applies:

- p_{at} empty
- no filter between p_5 and p_6
- a token arrives at p_r.

-5.10- The communication system

With p_{at} empty and a token in p_r, t_4 will fire, placing a token in p_3. This will prevent t_3 from firing. The message from either the event or state channel will be transferred to p_5 when t_{EM} or t_{SM} fires, depending on the selection of either a state or event message. With the token in p_5 and a filter not present, p_6 will also receive a token. Transition t_5 is now enabled and the message is transferred to the receiver.

For INPUT statement C.3 the following applies:

- p_{at} contains a value
- no filter between p_5 and p_6
- a token arrives at p_r.

The timeout period is added to the real-time when t_2 fires. This time is compared with the real-time in t_3. Transition t_3 will fire if a valid message is not available within the timeout period. If a valid message is available or becomes available, the transfer of the message takes place as was described for the previous case.

For INPUT statement C.4 the following applies:

- p_{at} empty
- a filter between p_5 and p_6
- a token arrives at p_r.

Since p_{at} is empty, the first part of case 1 is

applicable. The message is transferred to p_5 and compared with the filter expression supplied. If the message complies with the filter requirements it is transferred to the user. If the message is filtered out, p_6 will not receive a token and t_6 will fire to indicate to the user that the message is not within the allowable filter range.

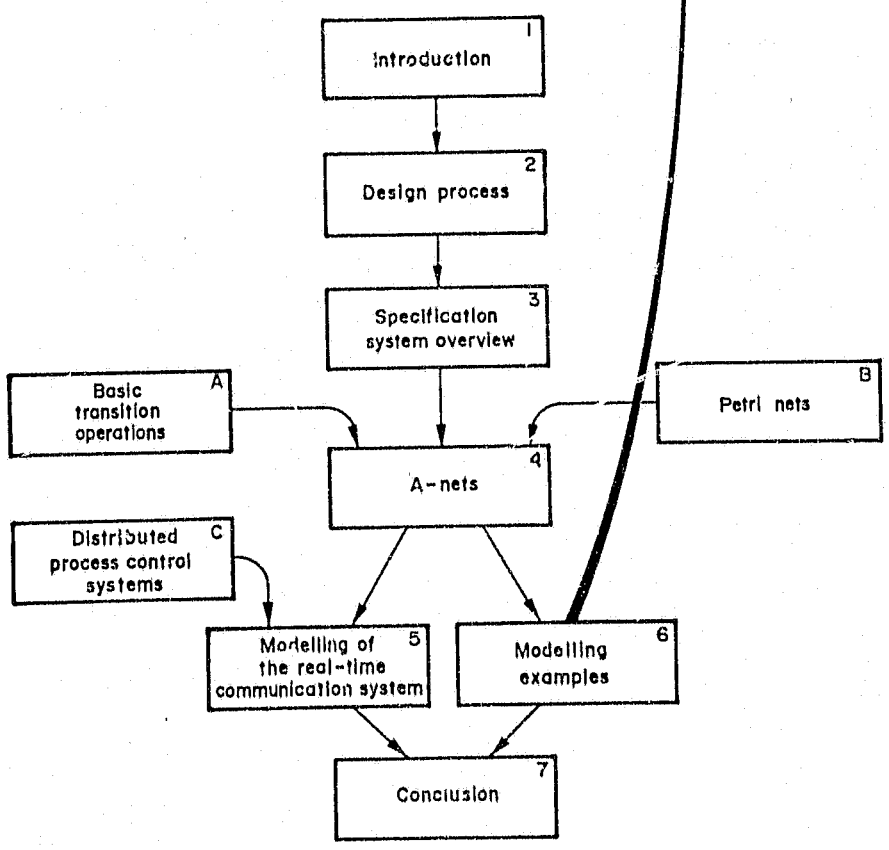
5.6 Conclusion

This chapter has shown the modelling of a real-time communication system with A-nets. The use of subnets helps with the hierarchical breakdown of the design process.

The use of A-nets as a modelling tool for this software analytical problem has the following advantages:

- a graphical representation is provided to give a better insight into the problem
- a mathematical representation of the problem can be derived from the graphical representation. This mathematical representation is available for analysis based on Petri net theory which, in turn, may for example be used to predict a deadlock situation.
- the model may be executed through which better insight into the system is gained and during which improvements may be tried before implementation.

6 MODELLING EXAMPLES



C H A P T E R 6

MODELLING EXAMPLES

6.1 Introduction

In the previous chapter the modelling power of A-nets was demonstrated in connection with a communication system for real-time applications. Yau (1983) demonstrated that Petri net based modelling is useful for the design of software for distributed systems. As was indicated in chapters 2 and 3, a model of the environment to be controlled is also important. This chapter will provide examples of the modelling of the environment and the control.

This chapter provides examples of the application of A-net modelling to industrial control engineering product design. It is shown how this method helps to deal with complexity by using a hierarchical approach. The advantages of a hierarchical approach have been discussed in chapter 2.

6.2 Pressure vessel

A pressure vessel at the end of a pipeline is used to

transfer material. Fig. 6.1 shows the vessel with its associated control valves and sensors. In this example A-nets with binary tokens are used.

6.2.1 Description of operation

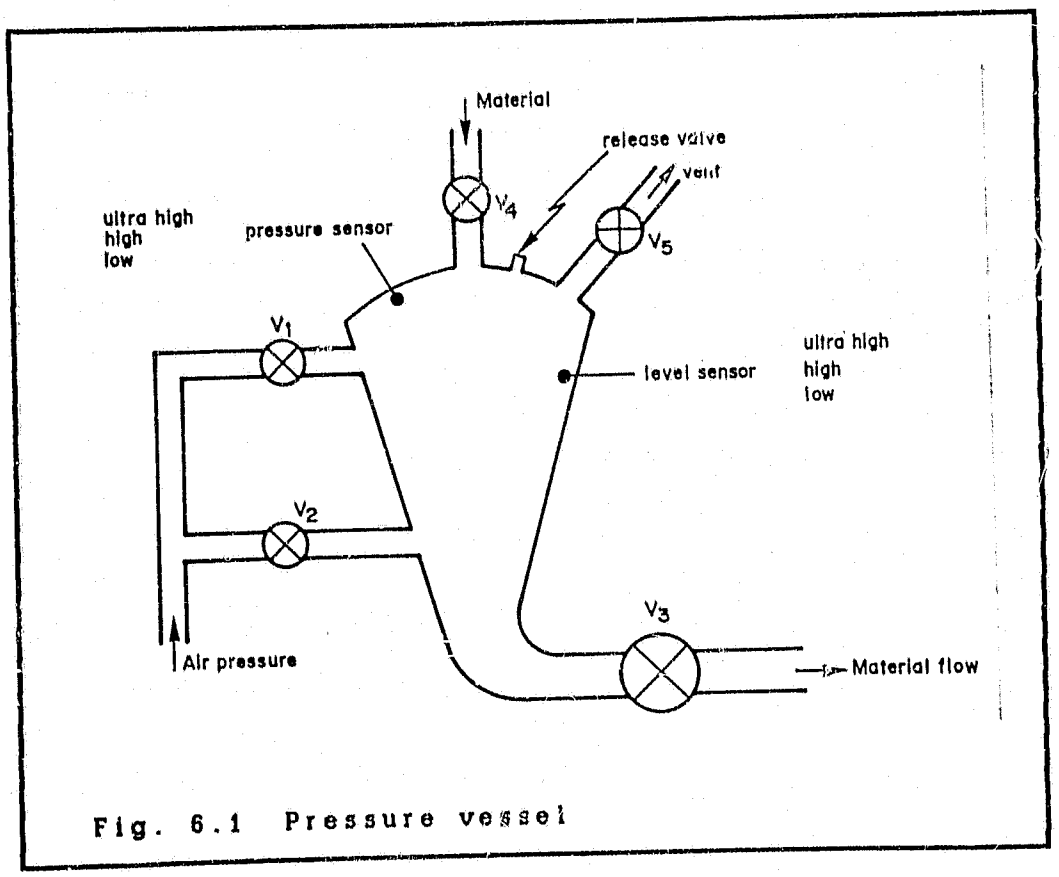


Fig. 6.1 Pressure vessel

With valves V_1, V_2 and V_3 closed and V_4 and V_5 open, material enters the vessel through V_4 until a high level signal is given. V_4 and V_5 close and V_1 and V_2 open to allow pressure build-up. As soon as the pressure is high, V_3 opens to allow material flow. As soon as the level sensor is on low, V_1 and V_2 close and the pressure is allowed to fall to low before V_3 closes. Then V_4 and V_5 open to repeat the whole cycle.

-6.3- Modelling examples

The following mishaps may occur:

- level ultra high may cause vessel blockage
- pressure ultra high may cause vessel rupture.

6.2.2 Model of the process

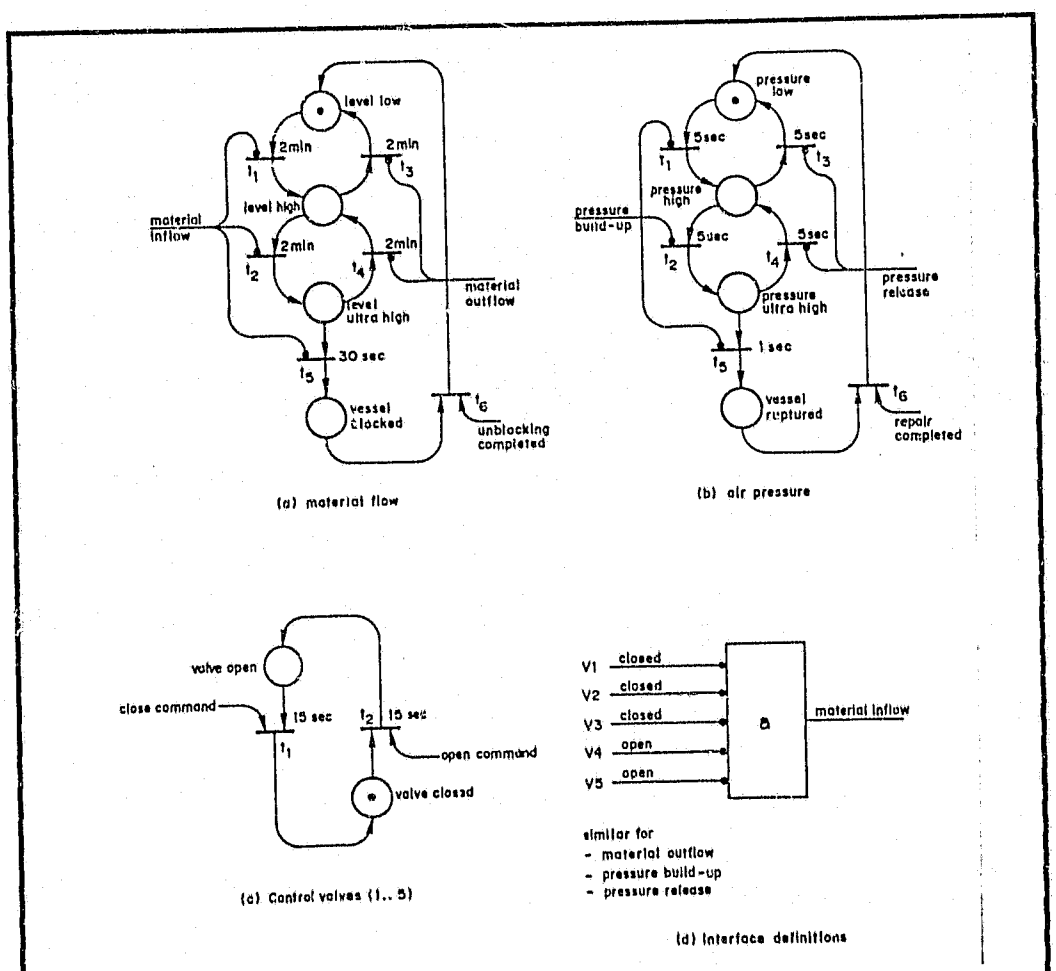


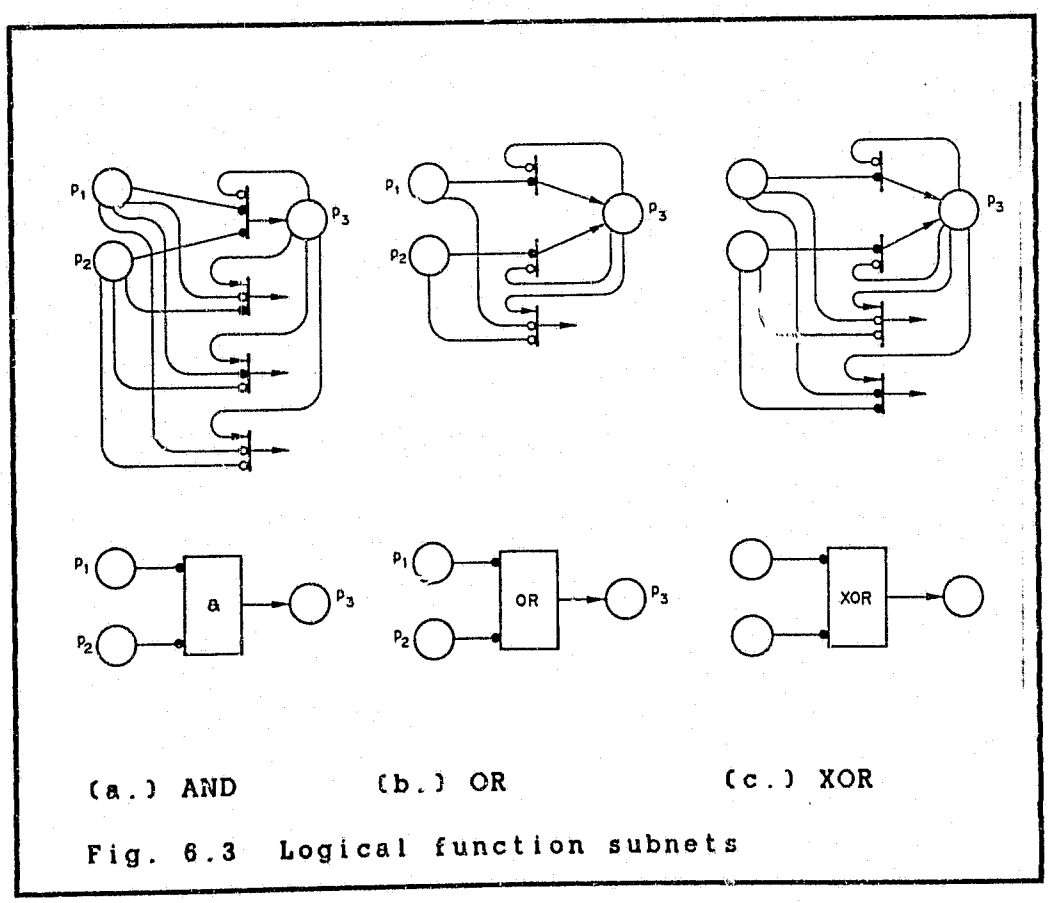
Fig. 6.2 Plant description

The following parts of the process are modelled seperately:

- material flow (Fig. 6.2(a))
- air pressure (Fig. 6.2(b))
- control valves (Fig. 6.2(c)).

-6.4- Modelling examples

These parts are then connected with diagrams like Fig. 6.2(d). A subnet known as "&" is used to make the diagram more readable. This subnet provides a logical AND-function. The realisation of a 5-input AND-function will give a complex A-net. Fig. 6.3(a) shows an AND-function for two inputs. Since logical functions are important and since they demonstrate the subnet principle, subnets for OR- and exclusive OR-functions are shown in Fig. 6.3(b) and (c).



(a.) AND (b.) OR (c.) XOR

Fig. 6.3 Logical function subnets

This model of the process is a first order approximation model only. The material inflow is for example taken to be a function of time only, while in practice it may for instance be a function of the type

of material and the humidity. This also demonstrates the limitations of using binary tokens only. A better approximation is possible with A-tokens. The humidity value could for example be used to select one of a couple of transitions with different times attached to it.

A description of the material flow follows. Assume that the level low contains a token. As soon as the conditions are fulfilled for material inflow, the transition t_1 will fire. After two minutes, the token will be produced in level high. Normally the material inflow should now be stopped for the discharge. If, however, the material inflow remains on for another two minutes, the level ultra high condition will come on. If it stays on for 30 seconds, the vessel will be blocked and a manual unblocking process will have to take place. A similar chain of events will take place on material outflow. The other parts of the model are self-explanatory.

6.2.3 Model of the control

The control has the following signals as inputs:

- level low
- level high
- level ultrahigh
- pressure low
- pressure high

-6.6- Modelling examples

- pressure ultrahigh
- start transfer command
- stop transfer command.

The control sends commands to the valves for opening and closing and may sound an alarm under certain conditions.

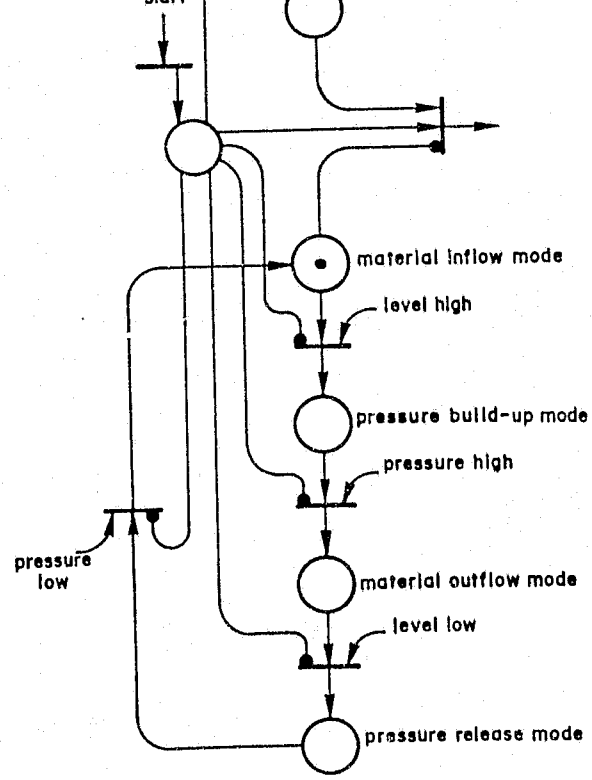


Fig. 6.4(a) Control description (Modes)

Fig. 6.4 shows the partial model of the control system. Fig. 6.4(a) shows the different modes the control system may assume. This is basically a state

diagram. Fig. 6.4(b) gives the alarm logic and Fig. 6.4(c) presents the materials inflow mode. This is an example of all the other mode diagrams.

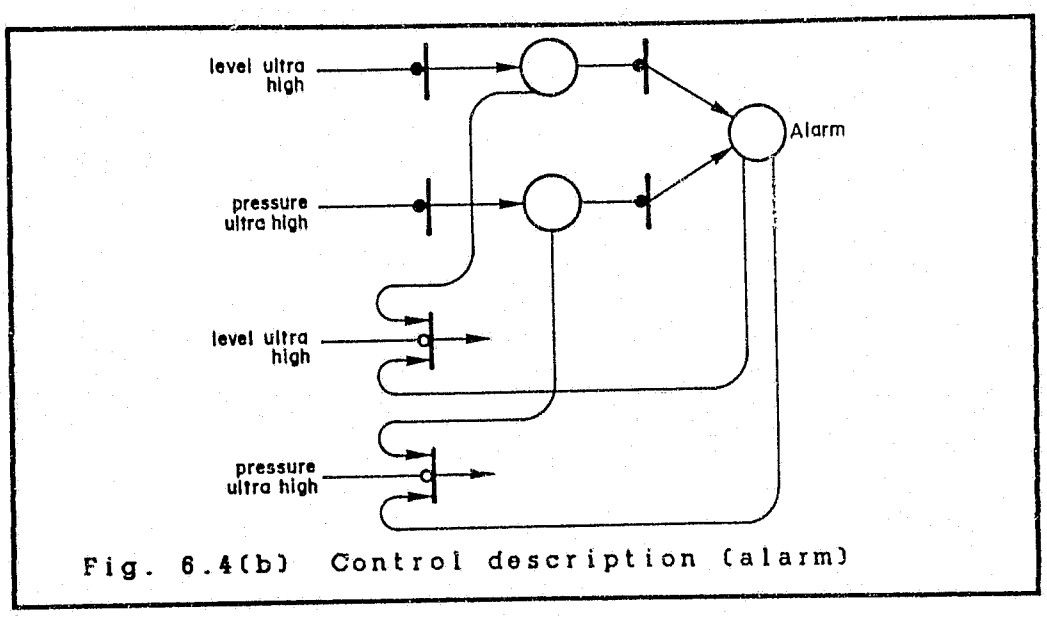


Fig. 6.4(b) Control description (alarm)

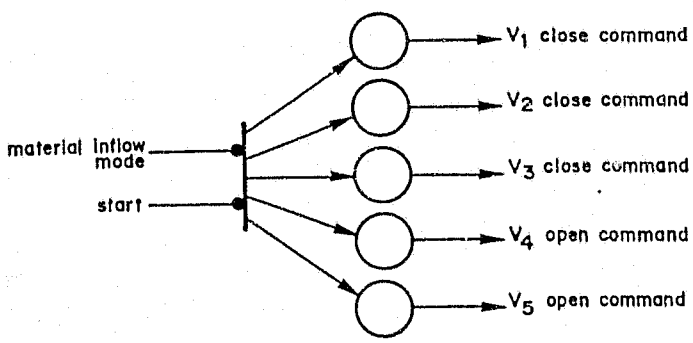


Fig. 6.4(c) Control description (material inflow)

6.3 Conveyor system

A conveyor control system is now used to demonstrate some of the capabilities of A-nets for the design of a

system. Fig. 6.5 shows a part of a large harbour with a conveyor system for materials transfer (Kruger, 1979) and (Kruger, 1985).

6.3.1 Description of operation

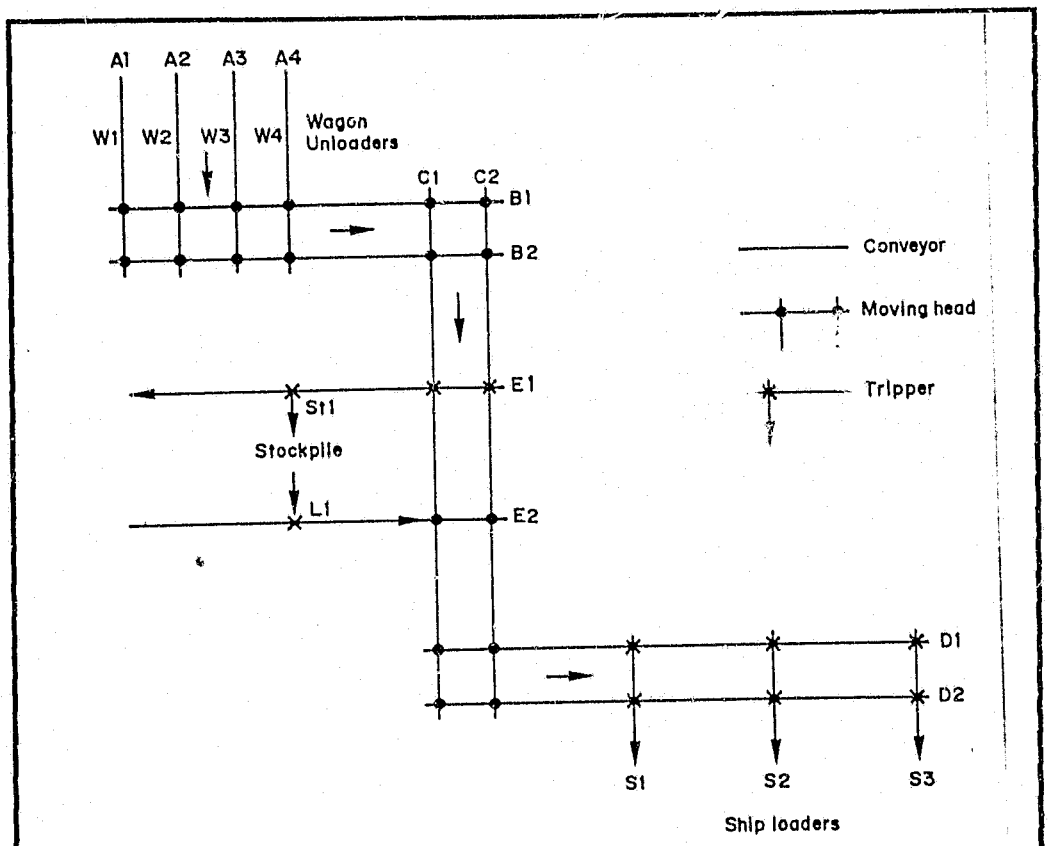


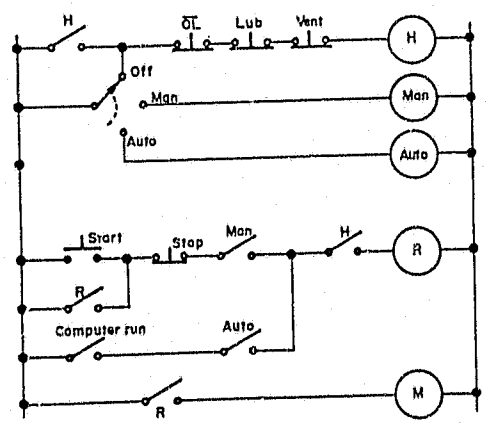
Fig. 6.5 Conveyor system

The material is unloaded from the wagons at W1 to W4. A conveyor route is set up where the feeder and receiver devices are given to the system by the operator. The route is then automatically selected according to certain criteria. The conveyors are started up beginning with the receiving device and terminating with the feeder. The shut-down of a route

is done in the reverse order.

6.3.2 Model of the process

The plant consists of many different devices like conveyors, moving heads, trippers etc. For this exercise a model of the electrical control (Fig. 6.6(a)) of a conveyor has been constructed (Fig. 6.6(b)). As can be seen, the conveyor may be selected for manual or automatic control. It may change from either of these modes to not healty if something fails, or to running when selected to run.



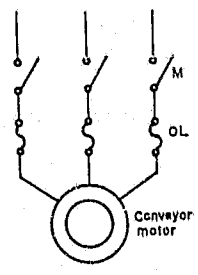


Fig. 6.6(a) Electrical control (circuit diagram)

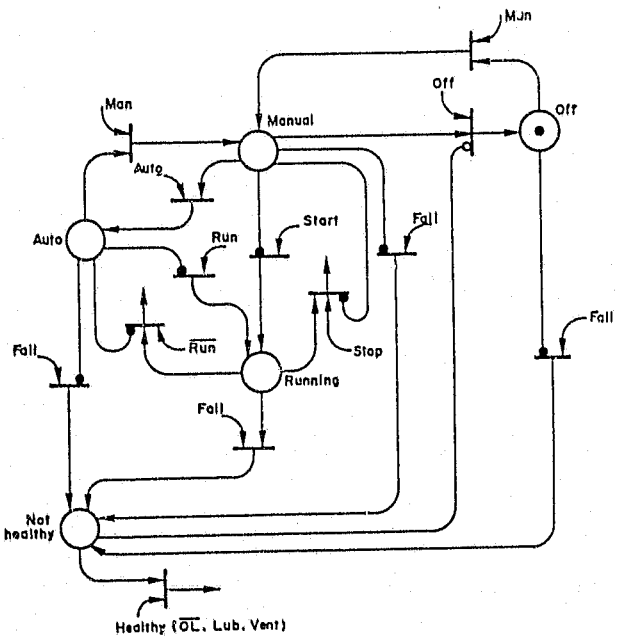


Fig. 6.6(b) Electrical control (A-net model)

6.3.3 Model of the control

The model of the design follows a hierarchical approach in this case. The overall design of the system is given with various subnets included. The detail design of one of these subnets is then presented in more detail.

Overall design

Fig. 6.7 gives an overall picture of the design. The following subnets are mapped to specific tasks:

- route setup (t_1)
- route selector (t_2)
- start-up (t_3)

-6.11- Modelling examples

- shut-down (t_4)
- accumulate runtime (t_5).

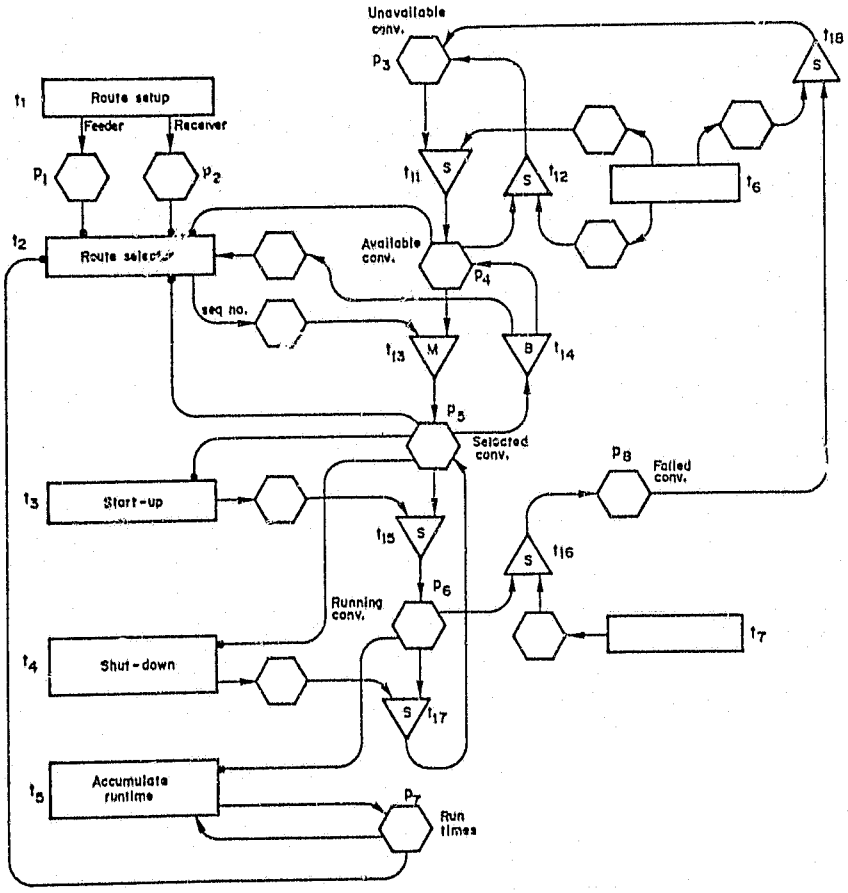


Fig. 6.7 Conveyor system - overall picture

The conveyors, as main components of a route, are mapped according to their status in places. For

example place p_5 contains the selected conveyors. The conveyors may be in any of the following states:

- unavailable (p_3)
- available (p_4)
- selected (p_5)
- running (p_6)
- failed (p_8).

The route setup subnet (t_1) is the interface to the conveyor operations personnel. A feeder (p_1) and a receiver (p_2) device are selected by the operator. These devices are used by the route selector (t_2) to select the route out of the available conveyors (p_4). The route selector uses the conveyor with the least run time (p_7) if more than one conveyor is available. The selected conveyors (p_5) are used by the start-up subnet (t_3) to get the route in the running mode. The run times of the running conveyors (p_6) are accumulated by the accumulate run time subnet (t_5). Shut-down (t_4) takes place on request or on an alarm condition.

Detailed design

The start up subnet (Fig. 6.8(a)) is developed further in order to demonstrate the detailed design.

The selected conveyors are started up in sequence.

-6.13- Modelling examples

The sequence numbers were added by the route selector routine. The loop p_1, p_2 and t_2 generates the sequence numbers for start-up. The selected conveyors are transferred to p_4 and p_5. The two adjacent conveyors are in p_4 and p_5. The moving head or tripper between these two adjacent conveyors is moved to the correct position by subnet t_{11}. The start conveyor routine (t_{12}) uses p_5 to start the conveyors in the right sequence.

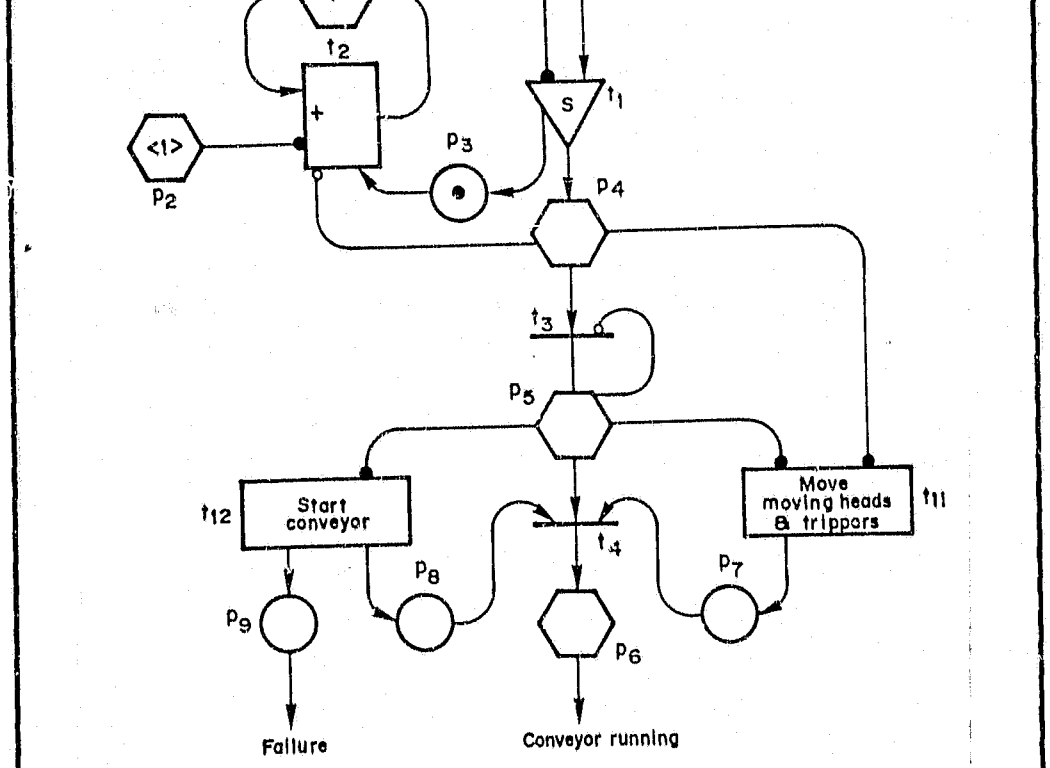


Fig. 6.8(a) Conveyor system - detail (start-up)

Fig. 6.8(b) shows the detail of the start conveyors

subnet (t_{12}). The conveyor in p_5 generates a start-up signal to the conveyor via t_{22}. A time delay is started. When the time delay expires, t_{21} will fire if the conveyor speed-up switch is on. If not, t_{23} will fire to generate a failure message in p_{23}. The electrical control model of Fig. 6.6(b) uses the start-up signal to start the appropriate conveyor.

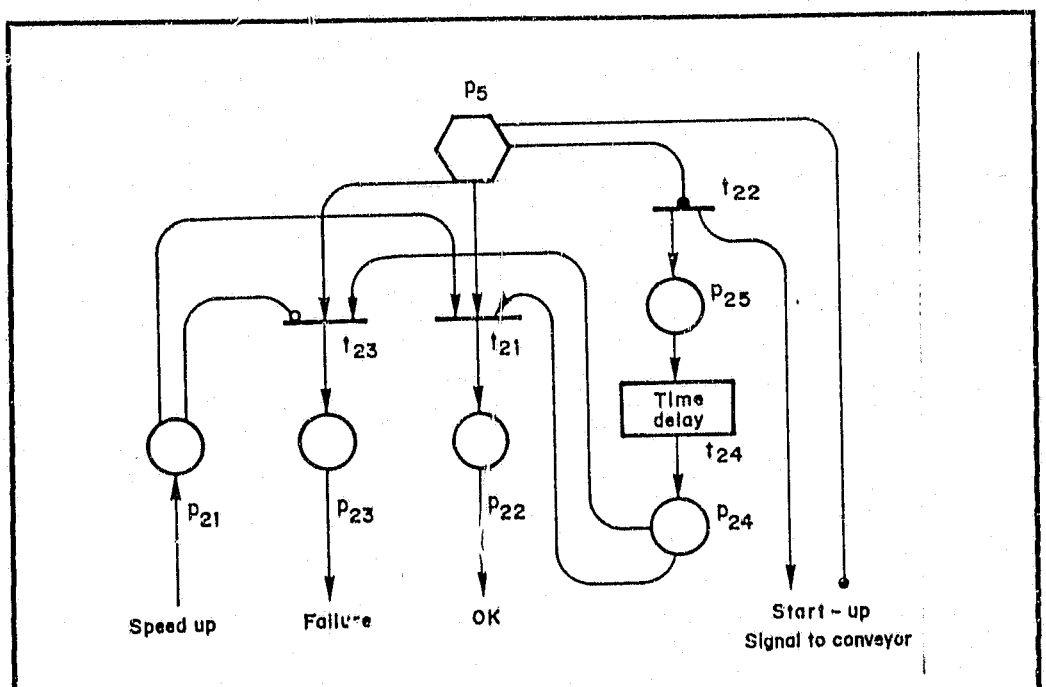


Fig. 6.8(b) Conveyor system - detail (start-up)

This illustrates the ability of A-nets to model a hierarchical (top-down) design approach. The eventual low level model can be linked to the model of the conveyor previously presented.

-6.15- Modelling examples

6.4 Access control to a building

6.4.1 Description of operation

Access control is applied to areas as is indicated in Fig. 6.9. Requests to enter an area are made and only if a person is authorised to enter, will the door be opened. Only persons that have entered will be allowed to leave.

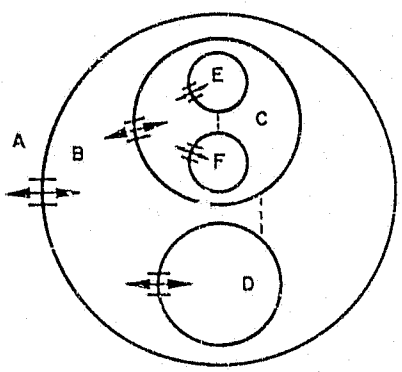


Fig. 6.9 Access control to an area

6.4.2 Model of the control

Fig. 6.10(a) shows a model for a single door. Subnets t_1 and t_2 appear on the entry and exit doors respectively. The door sends an enter request as well as a demand for the identification of the person about to enter. A signal is sent back to open the door or for the alarm to sound if the entrance is illegal. In this example a person is allowed to enter if his name

appears in the authorised person file and if he has already entered an adjacent area with access control.

The detail of the subnet is shown in Fig. 6.10(b). Transition t_{11} selects whether a person is authorised to enter and t_{12} selects whether he has already entered the adjacent area.

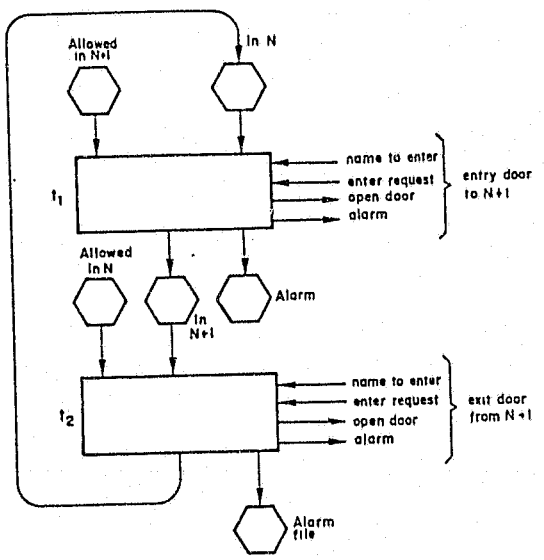


Fig. 6.10(a) Access control - overall diagram

6.5 Conclusion

This chapter has illustrated the ability of A-nets to model the process as well as the control of three different process control systems. The graphical representation helps to deal with complexity and

allows the designer to create innovative designs. These advantages were discussed in chapter 2. A-nets may be considered as a blueprint for the design of the software for complex industrial control problems. Chu (1982) has used this analogy of engineering blueprint extensively for his methodology used for data-processing systems.

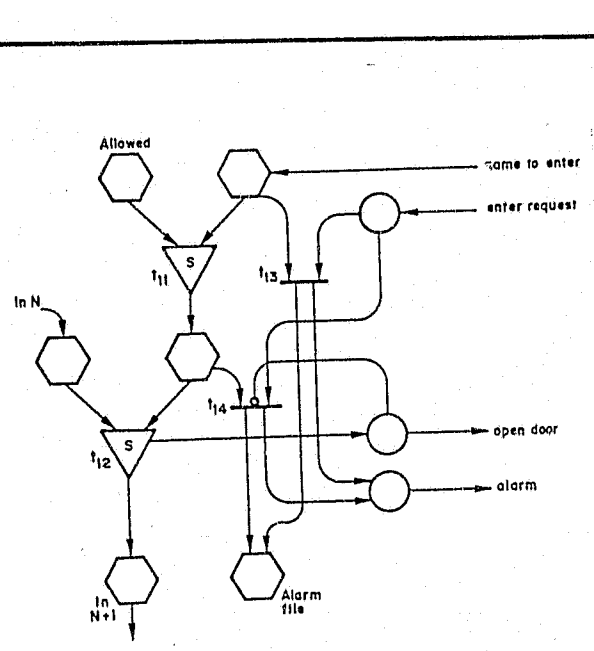


Fig. 6.10(b) Access control - door subnet

A mathematical representation of the model is possible. This mathematical representation may be analysed to predict the performance of the design - deadlock situations may for example be disclosed. The argument for predictive capabilities was presented in chapter 1.

The A-net models are executable. The designer can play

-6.18- Modelling examples

"what ... if" games on his design to improve certain characteristics. It is therefore a quick prototyping system. It was shown in chapter 2 that a prototype is an aid in producing a product to satisfy a need. The user can make a better judgement with regard to the fulfilment of his needs when he watches the performance of a prototype.

The perception that the system designer has of the environment of the control system is also modelled with A-nets. This has the following advantages:

- the user and designer can agree on the environment since both can understand the model and discuss it before the control is designed.
- misconceptions of the environment is clear to the designer and user once the operation phase is entered.
- changes in the environment during the operation phase can be modelled and the effect on the control system can be evaluated.

A-nets allow for a hierarchical approach and it is an effective method to deal with complexity, as was illustrated in paragraph 2.2.1. In this methodology, as was demonstrated in the examples of this chapter, the concepts of the state of the system as well as state changes are used.

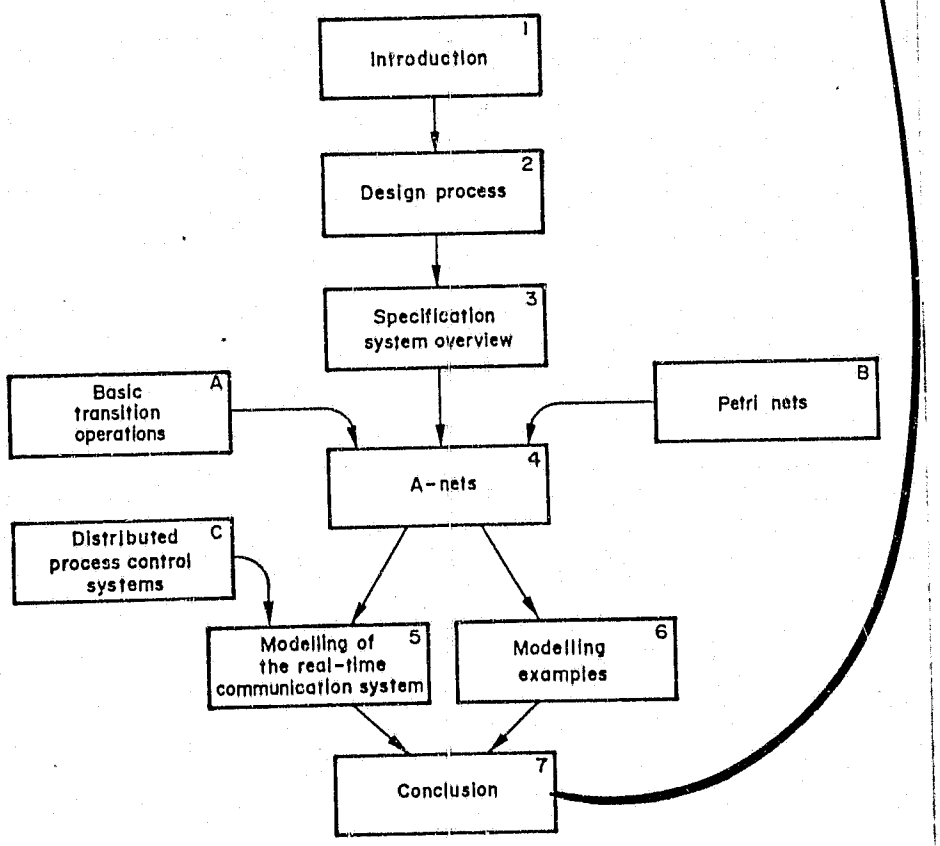
We can conclude from these examples and from the

-6.19- Modelling examples

discussion that the use of A-nets is beneficial during the following life cycle phases of the industrial control system:

- requirements (needs) analysis
- architecture design
- detail design
- evaluation and test
- operation and maintenance.

7 CONCLUSION



C H A P T E R 7

CONCLUSION

7.1 Introduction

This chapter firstly describes future work to be undertaken to create an automated specification system based on A-nets. Concluding remarks then follow.

7.2 Future work

Looking closely at the examples in chapters 5 and 6, it is evident that an automated system is required to assist the designer. Creating and updating the graphical representation in a manual way is time-consuming and error prone.

The use of static tests to predict certain factors such as deadlock is also required. A complete set of static tests must be implemented.

The execution of the A-net to establish certain factors and to debug the design is important. The execution also renders a prototype that is valuable in discussions with the user to establish his real needs.

-7.2- Conclusion

The system described in the following paragraphs is proposed to perform the above-mentioned functions. This system, however, is not implemented as part of the study for this thesis.

7.2.1 Structure of the system

The structure of the system is shown in Fig. 7.1. It consists of:

- an editor
- a filer
- a simulator.

Editor

The editor allows one to enter a new A-net or to modify an existing one. It also allows one to display the net in graphical form.

Filer

The filer is used to do maintenance on the saved A-nets. An A-net can be saved, retrieved, or filed.

Simulator

The simulator executes an A-net and displays results to the designer in an interactive way. It also includes

the static tests for analysis.

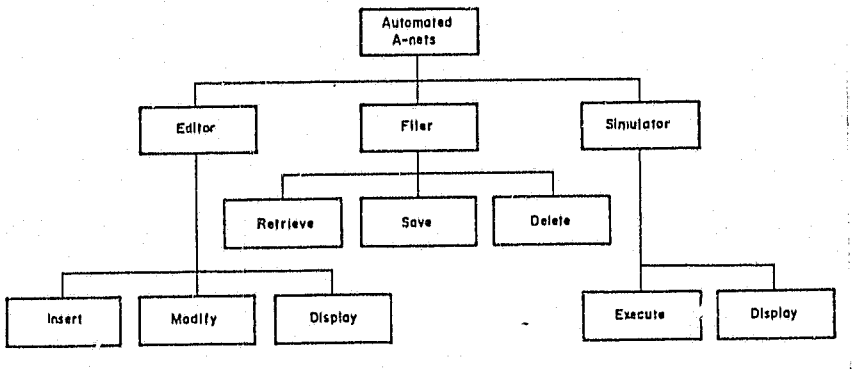


Fig. 7.1 Design system structure

7.2.2 Modelling of A-nets

The following information is required to build A-net models:

- attributes of A-tokens
- places
- transitions.

This is illustrated by the conveyor example in chapter 6.

Define attributes:

a1: conveyor_name;

a2: sequence_number;

Define places:

p1: feeder_device;

attributes: conveyor_name;

initially: empty;

p2: .....

Define transitions:

t2: route_selector;

type: subnet;

transfer inputs: p_{t14};

activator inputs: p_1, p_2, p_4, p_5, p_7;

inhibitor inputs: empty;

outputs: p_{t15};

priority: 1;

duration: 1;

t3: .....;

7.2.3 Summary of future work

Paragraphs 7.2.1 and 7.2.2 give an explanation of the short-term work required to create an automated tool for the use of this methodology for the design of industrial control systems.

The longer term research consists of the following:

- The examination of the use of artificial intelligence principles in the design process as was discussed by Borgida (1985).
- The examination of the application of A-nets in a wider application area. Petri nets are used to model systems such as a harbour complex (Torn, 1981). The examples in chapter 6 have illustrated that the A-nets can be used to model the

environment of various systems. It is therefore proposed that A-nets may be used to model many different types of systems. Of direct importance is the design of data-processing systems. Verification of this claim is to be confirmed in future research.

7.3 Concluding remarks

Distributed Computer Control Systems have recently become a very important means to implement industrial computer control. Driving factors are the advantages of these systems, such as fault tolerance as well as compatibility with the distributed nature of the plant to be controlled. A proper design methodology is, however, important for further advances and to take full advantage of the benefits of the cheaper hardware. The benefits of cheaper computer hardware can only be beneficial if more reliable software can be produced more cheaply. This study has proposed A-nets as a way of modelling distributed systems to make the advantages of distributed computer control systems more accessible to different applications.

A new design methodology has been introduced in this thesis. The key idea of this thesis is to provide a design methodology with a firm mathematical base. The mathematical formulation of Petri nets forms the necessary theoretical base of this methodology.

-7.6- Conclusion

Although it has not been done in this thesis, Petri net theory can be used for static analysis to predict certain features of the design. It has been illustrated that A-nets can be used to model various types of control systems.

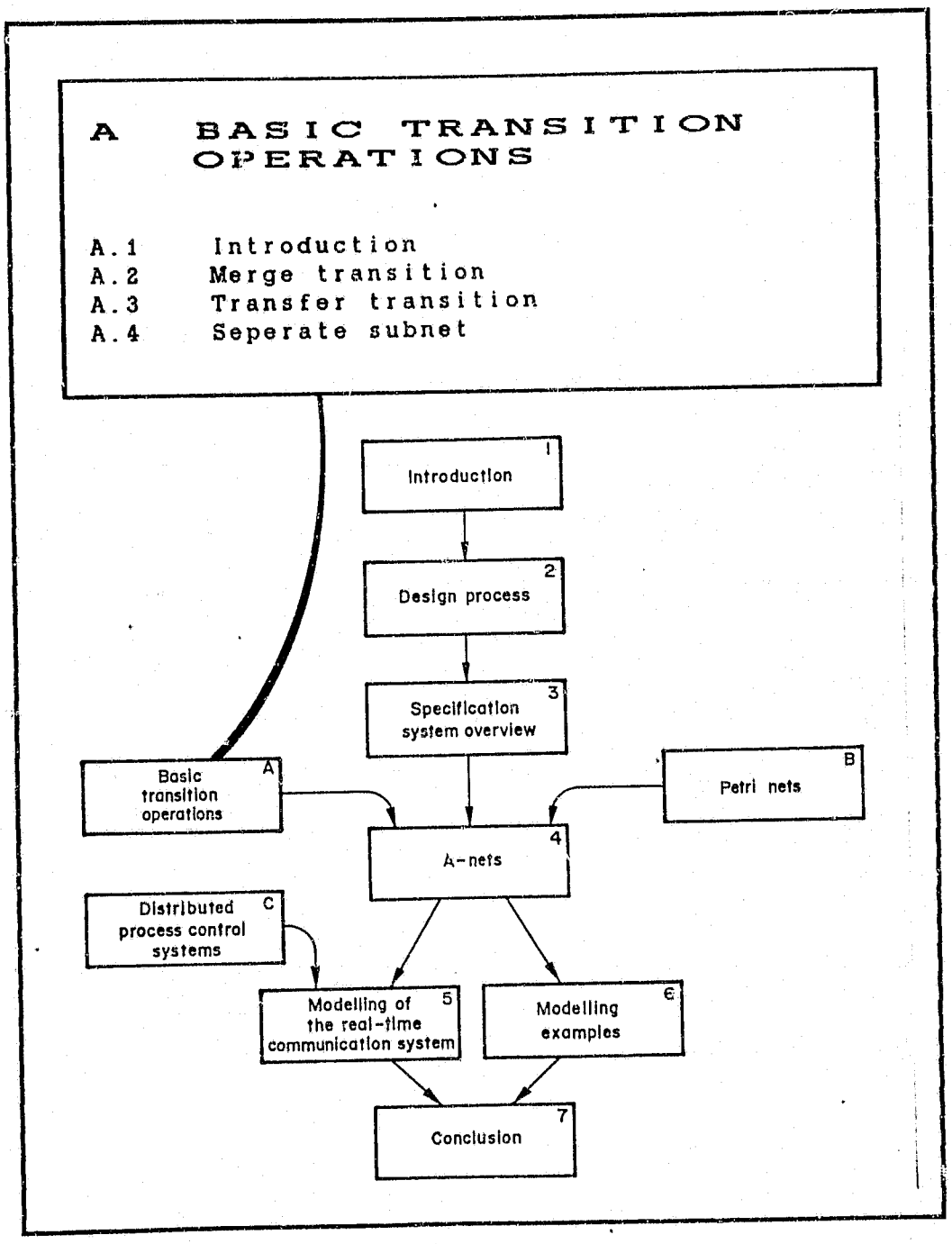
This thesis provides a theoretical introduction to A-nets. A-nets are based on Petri nets with a vast amount of theoretical work available for analysis. This allows one to predict the outcome of the design at an early stage of the design process. Apart from the fact that Petri nets are used to model various systems, they also provide a hierarchical method for modelling a complex system in levels of finer detail.

The execution of A-nets can be automated. This provides a way of executing the specifications of a system modelled with A-nets. Executing this model provides the opportunity to remove discrepancies. It also provides the user with a quick prototyping system where the designer and specifier may see the outcome of the design before it is implemented. This increases the information base of the control problem as well as of the proposed design of the control strategy.

APPENDICES

A Basic transition operation
B Petri nets
C Distributed process control systems

-A.0- A design methodology based on augmented Petri nets



A P P E N D I X A

BASIC TRANSITION OPERATIONS

A.1 Introduction

A-tokens are considered to be represented by vectors. For example take the generalized transition of Fig. A.1. If we assume that the attributes are defined as:

$$A = \{a_1, a_2, \ldots a_q\} \qquad \text{(A.1)}$$

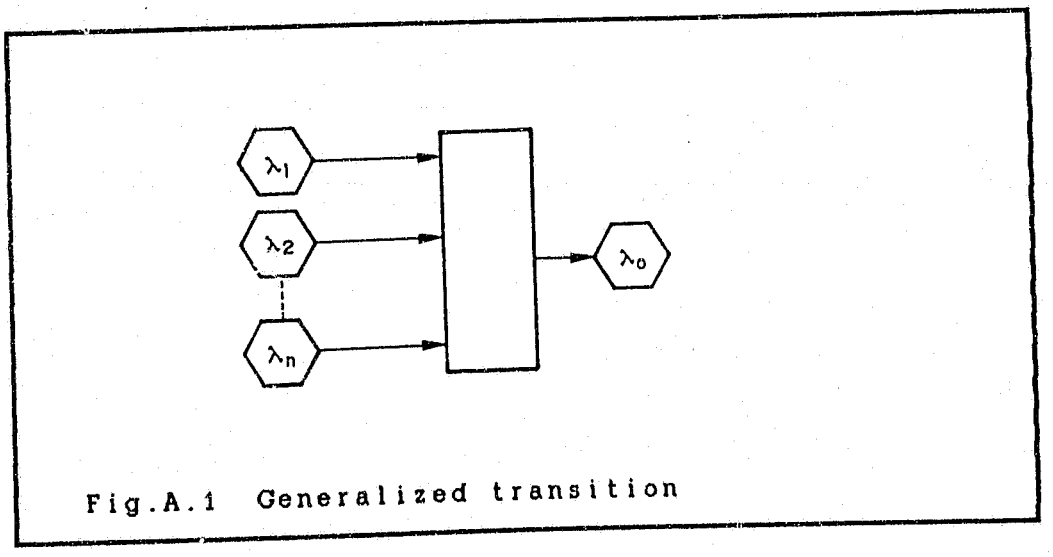


Fig.A.1 Generalized transition

Let token $\lambda_1 = \epsilon_{11}a_1 + \epsilon_{12}a_2 \cdots \epsilon_{1q}a_q$

and $\lambda_2 = \epsilon_{21}a_1 + \epsilon_{22}a_2 \cdots \epsilon_{2q}a_q$

and $\lambda_n = \epsilon_{n1}a_1 + \epsilon_{n2}a_2 \cdots \epsilon_{nq}a_q$