6

analysis of coordination problems in call handling software

"Man is a tool making animal."
P.H. Kylstra (1976)

In: Registratie, Vol. 8, No. 1
March 1976, (in Dutch).

"Very little happens in today's complex society that does not generate a piece of paper."
A. Kent (1962)

Information retrieval - review and prospective. In: Proc. IFIP 1962, Munich, N.H. Publ. Co. 1963, pp. 267-272.

Contents Chapter Six

CHAPTER SIX

ANALYSIS OF COORDINATION PROBLEMS IN CALL HANDLING SOFTWARE

6.1. INTRODUCTION

Up to this point we have studied coordination problems mainly in the context of the general purpose multiprocessing systems. In this chapter we will apply the analysis and design techniques developed in this thesis to the study of coordination problems in call handling software for telephone systems. We will try to describe and solve those problems systematically for an imaginary s.p.c. telephone system with (potentially) an extreme amount of concurrency. The analysis is performed on a relatively high level of abstraction, and, as a result, is largely implementation-independent. We make hardly any assumptions about specifics in the call processing, nor any about the specific type of system in which the solution should be implemented. We do make assumptions about the general structure of the telephone call handling processes.

Coordination rules

The analysis yields a formal specification of the coordination requirements of the system per functional step in the call processing. This formal specification is given in the form of initiation conditions for functional sections, in what can be called an 'abstract solution'. The correctness of this abstract solution can be verified on a number of criteria, like absence of deadlocks, reachability of system states, etc.
The abstract solution can subsequently be transformed (automatically) into a concrete solution with (more) standard coordination primitives, e.g. dependence operations.

Real-time aspects

By abstracting from the implementation specifics we also abstract from the real-time aspects of the call processing. It may seem odd that the real-time constraints can largely be ignored in this study; however, it should be noted that these requirements are, in call processing, primarily concerned with the execution of functions as such (e.g. i/o processing) and not with their coordination. There are many real-time requirements which are merely related to 'service-quality' measures (e.g. 'dial-tone should be returned within 3 seconds after the generation of an off-hook signal for 99% of the originating calls). These 'service-quality' requirements clearly depend more on system-capacity than on process coordination aspects. It would, therefore, only be diffusing to include such criteria in an analysis of coordination problems.
Although the real-time aspects can indirectly be the cause of coordination problems (e.g. in a multi-programmed system with priority interrupting, see chapter 5), there is still no need to consider these causes in this study.
As argued in chapter 1, there is no principal difference between coordination problems caused by virtual and actual concurrency.
In this chapter we abstract from these aspects, and base the analysis on a sort of 'worst case' in which each single processing function (like 'digit reception', or 'call routing') for each subscriber line may be executed concurrently with any other function. Each function distinguished here is thought to be executed on a separate processing unit, with direct access to all shared data.

More precisely: each function, or part of a function, for each *party*[1], may be executed on a separate processing unit. The number of processing units is not restricted in any way. The resulting (chaotic) concurrency in this imaginary system is to be coordinated by the set of coordination rules which will be derived.

Implementation aspects
For all clarity we add here that the 'worst case' assumption, mentioned above, does not imply that the only possible implementation of the abstract solution would be an actual multi-processor system with as many (or more) processing units as the product of the number of parties and the number of processing functions. Such a system would, most probably, be impractical as the delays caused by processor contention may well exceed the time gained by parallel executions. On the contrary, any implementation in which the execution of the functions is intelligently combined into a smaller number of processing units is possible. Some combinations may relieve the coordination requirements and make the implementation much simpler than the abstract solution. At some point there must be an optimum for the time gained by the prevention of delays in combinations, and the time lost by the sequential execution in a single processor. A more detailed study of such optimal load-sharing strategies is however not attempted here.

Levels of abstraction
In the analysis we distinguish between 5 levels of abstraction:

(1) The level of complete *subscriber processes*.
(2) The level of the *call processes*, of which there are at least two, namely one for the *calling party* and one for the *called party*.
(3) The level of the *call phases*, of which there are four: the *pre-call phase*, the *initial-call phase*, the *terminal-call phase* and the *post-call phase*.
(4) The level of the *call functions*, like digit reception and call routing.
(5) The level of the *primitive operations*, like the macro 'release cross-point'.

The first four levels are discussed in detail. A refinement to the fifth level is briefly outlined.

Restrictions
Clearly, the design and analysis of a 'complete' software system for a modern s.p.c. telephone exchange is far too comprehensive to be treated here. The first restriction we make is that we consider only the coordination problems in the call handling software as such. This implies that many interesting aspects of telephone systems with multi-control are not treated here. We mention:

- system maintenance, fault detection, and fault tracing software;
- system recovery methods, and take-over software;
- memory organization and data structures;
- signalling and timing problems;
- hardware organization and network structures.

Instead, we concentrate on the analysis of a simple call processing scheme. Initially we will make two further restrictions on the generality of that scheme.

— — — — —

(1) The word *party* is used to indicate a terminal (subscriber line) or a trunk (in general a two-way line between two exchanges).

(1) The first restriction is that we consider only *local calls* at length, and only briefly outline the extensions for the non-local variants. These extensions are fairly straightforward and do not add essentially different coordination problems to the ones treated here. There is one interesting problem that does only occur for non-local telephone calls: the usage of the two-way lines between exchanges. This problem is studied separately.

(2) The second restriction is that we assume the existence of an *ideal* system, in the first part of the analysis. With the term 'ideal' we mean that:

   a. The system considered is virtually blocking free, has an ample resource of service-circuits, and will not break-down, neither in its hardware, nor in its software.
   b. The system has 'ideal' subscribers, which implies that these subscribers will never make a premature cleardown, will not dial non-existing numbers, will observe all time limits, and will answer all calls. The only handicap the ideal subscriber is allowed to have is that he may be occupied (engaged) when called.

These restrictions may seem rather severe, still it can be argued that they are not really essential in view of the analysis of coordination problems. The extension of the call processing schemes to a non-ideal system is again fairly straightforward. It merely implies the addition of a large number of *escape-routes* to these schemes. The only problem added to the ones considered here is the problem of coordinating the escapes: if one branch of a parallel path is escaped, all the other paths should be aborted. The latter problem is however not a 'new' problem in the schemes. We have allowed for subscribers to be occupied when called, which yields one specific escape-condition. As we shall see this escape can cause precisely the same problems as sketched above.
A more detailed analysis of an extended call processing scheme would add little to the analysis performed here. We will outline the extensions in brief, and study some of the more interesting coordination problems related to these extensions separately.

By concentrating on a 'simple' call processing scheme, which nevertheless exhibits all principal types of coordination problems, we hope to keep the analysis transparent and yet complete. In the analysis we will encounter many curious problems. Under unfavorable circumstances, for instance, it may even in an 'ideal' system be possible that two subscribers will never reach each other, even when they both explicitly desire to do so.
We will consider special subscriber facilities, like recording services, follow-me, ring-back, etc. in detail, and we will discover several undesirable blocking effects for these. We will, finally, study methods of preventing the blocks.

Top down approach
At present, the usual way to study coordination problems in call processing schemes is to study them on a rather low level of abstraction, that is: on the implementation level, by trial and error. Coordination problems are often not recognized as such. The correctness of the resulting coordination schemes then depends largely on the skill and insight of the programmers. This skill and insight must however stand a severe test. The programmer should be able to 'foresee' where concurrency may lead to errors, from a very unfavorable position. **Boute '78** made similar observations, when he wrote:

> "... the traditional designer still holds an equivalent mechanistic view, namely of bit patterns being exchanged or modified by devices, CPU or memory. This is probably the main reason for the high design

> and software cost, because the major cost factor is the mastering of complexity."
> **(Boute '78, pg. 18).**

The design procedure followed here is essentially 'top down'. The call processing scheme is refined stepwise. We will not attempt to prove a specific implementation correct; instead we will try to analyze under what conditions *any* implementation can be called 'correct' for specific criteria.

6.2. SYSTEM REQUIREMENTS

The first difficulty one encounters is the problem of giving a specification of the system requirements. When do we call a telephone system correct, or which criteria should one use?
The ultimate correctness criterion for a telephone system is clearly that it should employ a specific type of behavior in response to subscriber actions. As we know, the subscribers are rather limited in the way in which they can communicate with a telephone system. Their options are: to lift the handset if it is on-hook, to lower the handset if it is off-hook, and in between, in a specific time-interval and with a specific speed, to send digits to the system. The system as a whole (the entire network of automatic exchanges) on the other hand may respond theoretically in an infinite number of ways. But, traditionally its options are also rather limited. It can send special tones, and possibly also taped messages, but most importantly: it can establish connections from one subscriber to another.

One usually distinguishes between the following four types of calls per exchange:

(1) Local calls (from one local subscriber to another local subscriber).
(2) Incoming calls (from a non-local subscriber to a local one).
(3) Outgoing calls (from a local subscriber to a non-local one).
(4) Transit calls (from one non-local subscriber to another non-local one).

The first two types of calls are also indicated as *terminating calls*; the last two types as *non-terminating calls*. Similarly, the first and the third type of call can be indicated as *originating calls*; the second and the fourth type as *non-originating calls*.
In the first phase of the analysis we concentrate on local calls. We first give an outline of the processing of a local call in a common telephone system.
As an example we take the space-divided system shown in figure 6.1.

Typical call process
When subscriber i lifts his handset, an off-hook signal is detected in the subscriber line unit SLU(i). The state of SLU(i) is scanned at regular intervals. If subscriber i is not occupied (for instance by a calling subscriber), the off-hook signal is interpreted by the control as a call request. The subscriber line is then connected to a free line-feed unit LF, via *C*oncentration/*E*xpansion network C/E (see figure 6.2.a). In this case the network C/E serves to concentrate a large number of subscriber lines on a small number of line-feed units. Then the line-feed unit is connected to a free digit receiver annex dial-tone sender R, via network E/I/C (*E*xpansion/*I*nterconnection/*C*oncentration). (See figure 6.2.b). Circuit R generates the dial tone. The subscriber acknowledges that signal by sending routing information into R. The digit receiver R is disconnected on completion of the digit reception. The control performs a number analysis, determining which subscriber corresponds to the chosen number.
If the chosen subscriber exists, and is presently not engaged in a call-process, then that subscriber is occupied and connected to a free line-feed unit, via C/E. (Observe, that network C/E, in effect, now serves as an expansion network.) A free route between the two line-feeds is selected and reserved, but not yet occupied. (See figure 6.2.c.) The B-subscriber receives a ringing-tone via LF; the A-subscriber receives a ring-back tone via his own line-feed LF[1].
When the B-subscriber lifts his handset, this is not interpreted as a call request but as a call answer. The ring toning functions are completed, and the

— — — — —

(1) The calling party is named the 'A-party', 'A-subscriber', or 'caller'; the called party is named the 'B-party', 'B-subscriber', or 'callee'.

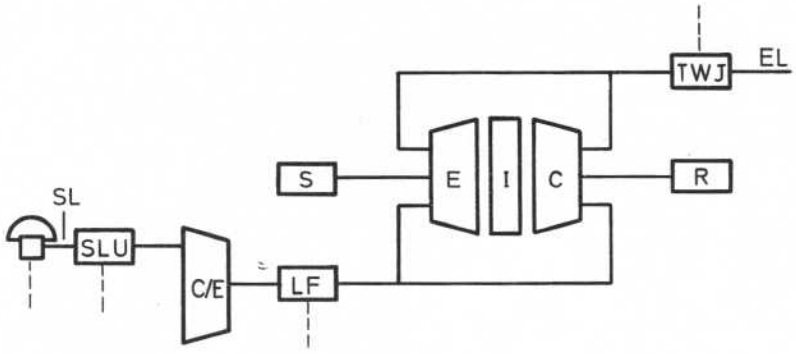

Figure 6.1.
Network Organization
(without Interface and Control Level)

SL = Subscriber Line
EL = Exchange Line
SLU = Subscriber Line Unit
TWJ = Two Way Junctor
C/E = Concentration/Expansion Network
LF = Line Feed Unit
S = Digit Sender
R = Digit Receiver
E = Expansion Network
I = Interconnection Network
C = Concentration Network

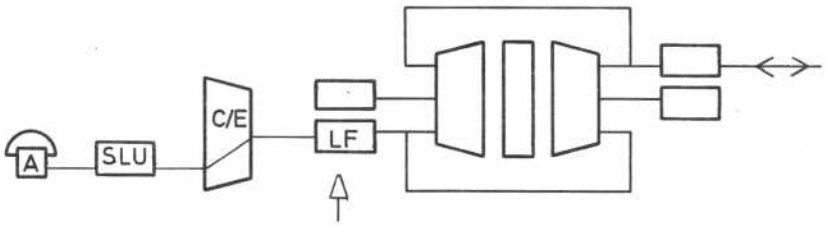

Figure 6.2(a).
Dial-Tone is returned from LF.

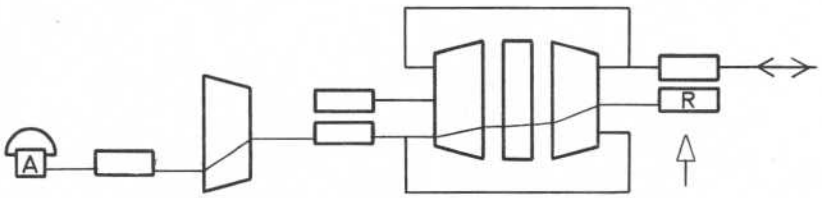

Figure 6.2(b).
Digits are received in R.

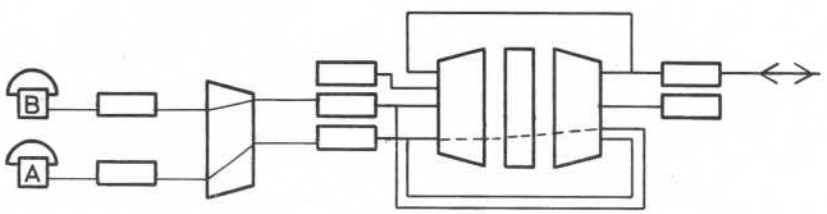

Figure 6.2(c).
A speech-path connection is reserved.

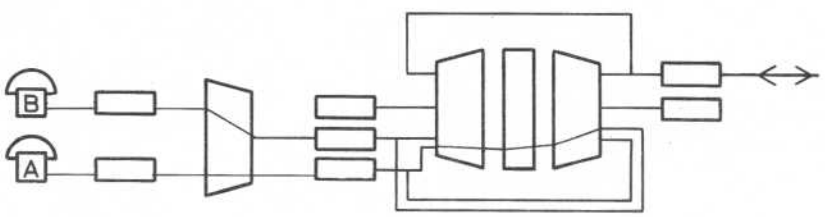

Figure 6.2(d).
The speech-path is established.

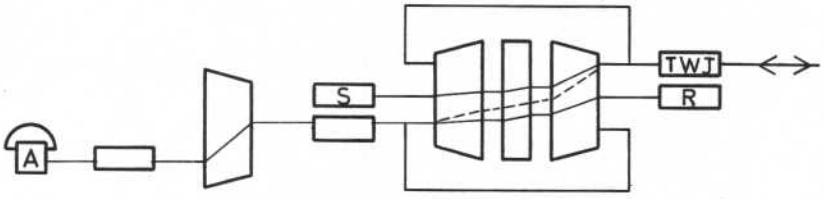

Figure 6.3(a).
Outgoing Call

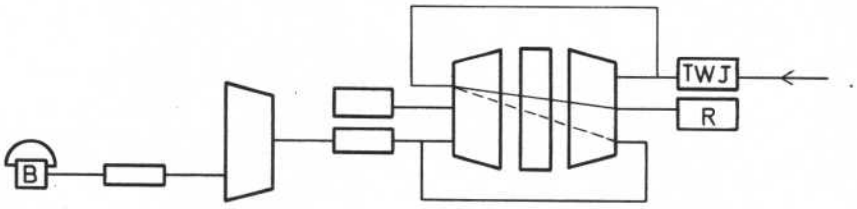

Figure 6.3(b).
Incoming Call

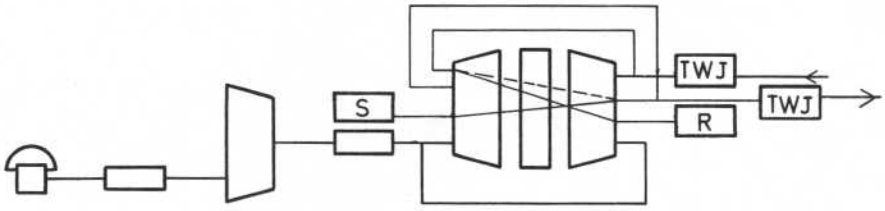

Figure 6.3(c).
Transit Call

reserved route is established. (See figure 6.2.d.)
On call ending (see below) the established route is released.
The party which is still off-hook receives a congestion tone from the line-feed unit. As soon as the subscriber returns his handset on-hook, the congestion toning is terminated and the line-feed unit is released, the route form the corresponding SLU via C/E to LF is released, and the subscriber is returned to the free-state.

In figure 6.3.a a typical call process state for an outgoing call is indicated. The call is routed via a two-way junctor and a trunk, to another exchange. First the route from LF to TWJ is selected and reserved; then a route from a free digit sender S to TWJ is selected and established. S will pass the routing information on to the succeeding exchange (link-by-link system). The call answer signal is passed on backwardly (from B-subscriber to A-subscriber) via the TWJ's.

In figure 6.3.b a typical call process state for an incoming call is indicated. The routing information is received in R. After the detection of a call answer signal, TWJ is used to send this message to the preceding exchange.

In figure 6.3.c, finally, a typical call process state for a transit call is indicated. Routing information is received in R, and passed on via S.
We have indicated the signalling strategy which is known as the 'link-by-link' method. The routing information is then passed on step by step, from originating exchange to terminating exchange. Another method is the 'end-to-end' system, where routing information is only sent from the originating exchange. The transit exchanges then merely establish a connection (speech path connection, or data line connection) between two exchanges. Each exchange in line receives its routing information directly from the originating exchange.

Call ending
There are four different ways to define call ending:

(1) The *called party release* strategy, implying that only a B-subscriber can terminate a call and remove the connection.
(2) The *calling party release* strategy, implying that only an A-subscriber can terminate a call and remove the connection.
(3) The *one party release* strategy, where either party can end the call and remove the connection.
(4) The *two party release* strategy, implying that a call is only terminated when *both* parties have cleared down.

Signals and layers
The signals generated by subscribers reach the control level in three steps.
The *first* step is the *generation* of the *primary* signal by the subscriber, for instance, by lifting the handset.
The *second* step is the *detection* of this primary signal in the system peripherals, for instance, by scanning the SLU's. The peripherals generate a *secondary* signal to inform the control level of the event.
The *third* step is the *recognition* of the secondary signal on the control level. On the control level the signal is interpreted and acted upon.
Our analysis of coordination problems is performed entirely on the control level.
The modern s.p.c. system are sometimes divided into three conceptual layers, corresponding to the three steps described here. The first layer then is the *network*-circuitry and the subscriber lines. (See figure 6.1.) The second layer is the *interface* between network and control. The third level is the *control* itself. A crosspoint, line-feed unit, or subscriber line unit, belongs to the

first layer. A scanner or marker belongs to the second layer. The processing units and memories belong to the third layer.

Occupation of circuits

The occupation of circuits, like digit receivers or line-feed units, will generally follow this pattern:

```
select a free circuit and occupy it in memory
   select a free route to the circuit and occupy it in memory
      establish the route in the network (via markers)
         initiate the circuit
         terminate the circuit or await its termination
      disconnect the established route
   release the route in memory
release the circuit in memory
```

In the third and fourth step one has to account for the hardware response times, which are on the order of 5 - 7 msec. In general one will first send a command and then check whether it has been executed or not, some 10 msec. later.

This outline of the general call processing requirements will serve as a guideline for the analysis to follow. We start with a high level description in terms of subscriber processes, which is then refined stepwise into more specific call processing functions and their mutual relations.
The initiation conditions which will be derived are divided into two parts: a *sequence* clause (sic), which formalizes the instant in the call process where a specific function is to be executed, and zero or more *proviso* clauses (pric) which formalize the additional coordination requirements. Each initiation condition (ic) is then composed as follows:

$$ic_i = sic_i \cap \left(\underset{j}{\Delta}\ pric_i^j\right),$$

where $\underset{j}{\Delta}$ represents disjunction over j.

6.3. COORDINATION REQUIREMENTS

6.3.1. Subscriber level

At a relatively high level of abstraction a telephone exchange can be described as a collection of subscriber processes. If τ is the set of subscriber lines and trunks, then there are precisely $|\tau|$ subscriber processes. (More precisely we should call them 'party processes', but as we restrict ourselves mainly to local calls, we may ignore the trunks for the time being (see footnote pg. 254). We name the subscriber process for the i-th party in the system S(i), where $i \in \tau$. Section S(i) can now be modelled as a structured section (see chapter 4). The subscriber processes are related to a class of structured sections. The formal parameter i is used to identify the parties.
We can, already at this level, formulate some general coordination requirements. Observe that it is implicitly assumed that always, no more and no less than, one subscriber process is executed at a time per party. This can be made explicit in the form of an exclusion clause for section S(i). We can derive that clause via system invariant:

$$(\forall\, i)\ \left[i \in \tau \rightarrow I_{S(i)} - T_{S(i)} \leqslant 1 \right]. \qquad (I)$$

The invariant can be falsified by the initiation of section S(i). This yields the exclusion clause:

$$pric^{1}_{S(i)} = \left[I_{S(i)} - T_{S(i)} = 0 \right].$$

On a still higher level of abstraction we may even consider the suspending and restoring of subscriber processes. The subscribers who do not pay their telephone bills can then be placed in an 'out-of-service' state which we call $\bar{S}(i)$. In a general graph we can then represent the coordination requirements on the subscriber level with simple exclusion and ordering relations, as shown in figure 6.4.

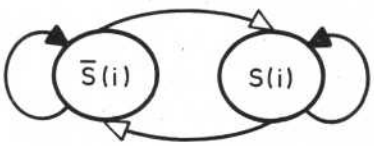

Figure 6.4.
Relations on the Subscriber Level

6.3.2. Call process level

Within the subscriber process we can distinguish between a FREE state, in which line scanning actions are to be performed (to detect call requests), and a BUSY state, in which the actual call processes are to be executed.
Within the BUSY state we can further distinguish between an A-side call process, and a B-side call process, which are executed respectively when the subscriber is calling or is called. We name the A-side call process A(i), and we name the B-side call process B(h,i). To each process we relate a structured section.

Section A(i) has one formal parameter which identifies the caller.
Section B(h,i) has two formal parameters, which identify, respectively, the caller (h) and the callee (i). If the identity of the caller is irrelevant we indicate B(-,i).

We may formulate the following coordination requirements on the call process level.
At no time should it be possible to execute an A-side process simultaneously with a B-side process, for a single party. It may not be possible to execute more than one A- or B-side process at a time, per party.

Observe that this requirement implies, specifically for the non-local variants of the call processes that the two-way junctors and the trunks may never be occupied by two exchanges at the same time in opposite directions, nor by more than one A-side process or B-side process within a single exchange. We return to this problem in a later section.

The coordination requirements described above can be formalized in a second system invariant, as follows:

$$(\forall\ i)\ \Big(i \in \tau \rightarrow \Big(I_{A(i)} - T_{A(i)}\Big).\Big(I_{B(-,i)} - T_{B(-,i)}\Big) = 0 \quad \text{AND} \quad \Big(I_{A(i)} - T_{A(i)} \leqslant 1\Big) \quad \text{AND} \quad \Big(I_{B(-,i)} - T_{B(-,i)} \leqslant 1\Big)\Big) \qquad \text{(II)}$$

From invariant II we can derive initiation clauses for sections A(i) and B(-,i). The clause is equal for both sections:

$$pric^1_{A(i)} = pric^1_{B(-,i)} = \Big(\Big(I_{A(i)} - T_{A(i)} = 0\Big) \cap \Big(I_{B(-,i)} - T_{B(-,i)} = 0\Big)\Big).$$

This relation between A(i) and B(-,i) is represented in the general graph of figure 6.5.

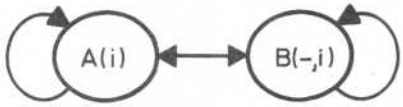

Figure 6.5.
Relation between A(i) and B(-,i)

Section S(i) can now be refined into a cyclic selection structure, as illustrated in figure 6.6. Subsection F(i) models the free state. The unlabeled section is needed to transform S(i) into a structured section. We will see that the gate variables from this dummy section can be eliminated from the coordination rules. (Observe that S(i) is still an A-graph, even without the dummy section. See appendix C, chapter 4.)
From system invariant I we can derive the 'capacity' of the cyclic structure. This 'capacity' is precisely 1. It then follows that:

$$sic_{F(i)} = \left(I_{F(i)} - T_{dummy} = 0\right)$$

$$sic_{A(i)} = \left(\left(T_{F(i)} - I_{A(i)} + I_{B(-,i)}\right) = 1\right)$$

$$sic_{dummy} = \left(\left(T_{A(i)} + T_{B(-,i)}\right) - I_{dummy} = 1\right).$$

If we replace T_{dummy} by $\left(T_{A(i)} + T_{B(-,i)}\right)$ we can omit section 'dummy' entirely.
The exclusion rules formalized in $pric^1_{A(i)}$ and $pric^1_{B(-,i)}$ are implicit in the sequence clauses derived here.
They can therefore be omitted as well.

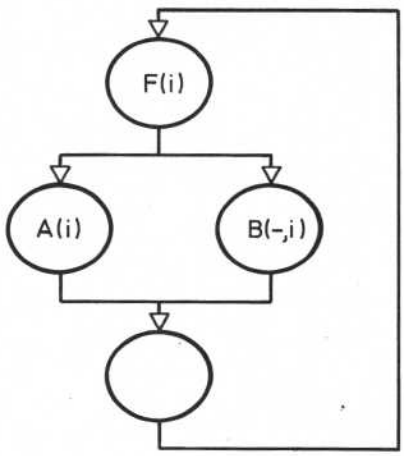

Figure 6.6.
Cyclic Selection Structure

Selection between A(i) and B(-,i)
Section A(i) should only be initiated after the recognition of a call request signal. B(-,i) should only be initiated after a 'seizure' of party i. We will elaborate the second condition in another place. To formalize the first condition we need a communal variable, which can be used to signal a call request from within F(i) to A(i). We name this communal variable STS(i), short for *status* i. In fact, we have introduced a whole vector of communal variables of length $|\tau|$.
STS(i) is defined as a binary variable, with initial value 0.
STS(i) = 1 means: party i wants to start or continue a call process.
STS(i) = 0 means: party i wants to return to or remain in the FREE state.
We have thus derived an additional proviso-clause for section A(i):

$$pric^2_{A(i)} = (STS(i) = 1).$$

The FREE state: section F(i)
In brief, the task of section F(i) is:

- First, to (re)start line scanning actions on subscriber line (i), for the detection of new call request signals, when the FREE state is (re)entered.

Secondly, to terminate those scanning actions when the FREE state is to be left, after the recognition of a call request signal (off-hook).

To avoid ambiguity we assume (require) that the call request signals are irrevocable. Whenever an off-hook signal is detected and STS(i) := 1, the line-scanning for party i is to be suspended until it is explicitly restarted.

Remark:

Actually it is not really necessary to interrupt the line scanning as such, during the call processing. One may consider the line scanning as an independent task, which is never interrupted. One may then 'simulate' the suspension of line scanning on specific lines by bit masking techniques.

Section F(i) can be divided into two subsections $F_1(i)$ and $F_2(i)$, one for each task mentioned. The second section is initiated when a call request has been recognized or when the considered party has been seized as a callee.
The first of these two initiation conditions for $F_2(i)$ can now be formalized as follows:

$$pric^1_{F_2(i)} = (STS(i) = 1).$$

The second condition will be elaborated on a lower level. The FREE state as such can also be formalized with the aid of the gate variables. Depending on one's view-point it can be defined quite narrowly as:

$$\left(T_{F_1(i)} - I_{F_2(i)} = 1\right),$$

or somewhat wider as:

$$\left(T_{A(i)} + T_{B(-,i)} = I_{A(i)} + I_{B(-,i)}\right).$$

We will use the second definition in the sequel.

6.3.3. **Call phase level**

Both the A-side and the B-side call process can be divided still further into four phases:

(a) *Pre-call phase*. Including call initiation and (for the A-side process) digit reception and number analysis[1].
(b) *Initial call phase*. Including the seizure of a B-party, path searching, ringing and connecting[1].
(c) *Terminal call phase*. Including the releasing of the connections between A-party and B-party.
(d) *Post-call phase*. Including congestion toning and call ending[1].

The coordination between the A-side and B-side call processes is concentrated in the phases (b) and (c).
When the required B-party is found to be occupied, the phases (b) and (c) should be skipped in the A-side process. To this purpose we define an 'escape section' $E_1(i,j)$ in the A-side process. Section $E_1(i,j)$ is executed if subscriber i calls subscriber j, while j is busy.
We will name the other sections as follows (see also figure 6.7):

— — — — —

(1) See appendix F.

	A-side	B-side
Pre-call phase	$A_1(i)$	$B_1(h,i)$
Initial call phase	$A_2(i,j)$	$B_2(h,i)$
Terminal call phase	$A_3(i,j)$	$B_3(h,i)$
Post-call phase	$A_4(i)$	$B_4(i)$

The symbol i identifies the subscriber considered.
The symbol j identifies the party called by i.
The symbol h identifies the party calling i.
If the identity of j or h is irrelevant we notate this as follows: $A_2(i,-)$ or $B_2(-,i)$, respectively.
It follows from invariant II that:

$$(\forall\ i)\ \left[i \in \tau \rightarrow I_{A_2(i,i)} = T_{A_2(i,i)} = 0\right],$$

and similarly for $A_3(i,i)$ and $B_1(i,i)$, $B_2(i,i)$, $B_3(i,i)$, as no subscriber can ever establish a call with itself[1]. It is not necessarily so that also:

$$I_{E_1(i,i)} = T_{E_1(i,i)} = 0,$$

as subscriber i may well *attempt* to call itself. Before detailing the coordination rules on the call phase level, we give an overview of the more trivial sequence relations between the call-phase sections. See figure 6.7. We can combine section $A_4(i)$ and $B_4(i)$ into a single section $AB_4(i)$ as their function is clearly the same in both processes.

The sequence clauses can now be derived directly from the definition of the structured sections, as given in chapter 4. The graph consists of a nesting of processing lines and selection structures. The only points which deserve special attention are the transfers:

$$A_2(i,-) \rightarrow A_3(i,-), \text{ and}$$
$$B_1(-,i) \rightarrow B_2(-,i) \rightarrow B_3(-,i).$$

Although the identity of, respectively, the caller and the callee, is not as such relevant, it is still important that this identity cannot change during the execution of a call process (at least not without special precautions, which we will consider later). This implies that $A_2(i,j)$ must be succeeded by $A_3(i,j)$ and not by $A_3(i,k)$ with $k \neq j$. This requirement necessitates the specification of the identity of the caller *and* the callee in the sections mentioned. Observe further that the sequence clauses for $A_1(i)$ and $B_1(-,i)$, and similarly for sections $A_2(i,-)$ and $E_1(i,-)$ are equal, as these are the initial sections in the branches of a selection structure. The actual selection between the branches is to be made via the proviso-clauses, which will be discussed later.

Below we will use the following specific abbreviations. We consider a section X with two formal parameters: X(i,j).

$$I_{X(i,-)} \stackrel{\text{def}}{=} \sum_{j \in \tau} I_{X(i,j)} \quad \text{and} \quad T_{X(i,-)} \stackrel{\text{def}}{=} \sum_{j \in \tau} T_{X(i,j)}$$

$$I_{X(-,j)} \stackrel{\text{def}}{=} \sum_{i \in \tau} I_{X(i,j)} \quad \text{and} \quad T_{X(-,j)} \stackrel{\text{def}}{=} \sum_{i \in \tau} T_{X(i,j)}$$

— — — — —

(1) Those cases in which *several* subscribers can be reached under *one* number are not considered here.

An initiation condition (or a coordination clause) $ic_{X(i,j)}$ (respectively, $pric_{X(i,j)}$) that holds for all values of j, is written as: $ic_{X(i,-)}$ (respectively, $pric_{X(i,-)}$), and similarly for i.
The following equivalences will be evident:

$$I_{A(i)} \hat{=} I_{A_1(i)};$$

$$I_{B(-,i)} \hat{=} I_{B_1(-,i)};$$

$$T_{A(i)} + T_{B(-,i)} \hat{=} T_{AB_4(i)}.$$

The list of all sequence clauses on the call phase level can now be written.

$$sic_{A_1(i)} = sic_{B_1(-,i)} = \left(T_{F_2(i)} - \left(I_{A_1(i)} + I_{B_1(-,i)}\right) = 1\right);$$

$$sic_{A_2(i,-)} = sic_{E_1(i,-)} = \left(T_{A_1(i)} - \left(I_{A_2(i,-)} + I_{E_1(i,-)}\right) = 1\right);$$

$$sic_{A_3(i,j)} = \left(T_{A_2(i,j)} - I_{A_3(i,j)} = 1\right);$$

$$sic_{B_2(h,i)} = \left(T_{B_1(h,i)} - I_{B_2(h,i)} = 1\right);$$

$$sic_{B_3(h,i)} = \left(T_{B_2(h,i)} - I_{B_3(h,i)} = 1\right);$$

$$sic_{AB_4(i)} = \left(T_{A_3(i,-)} + T_{B_3(-,i)} - I_{AB_4(i)} = 1\right);$$

$$sic_{F_1(i)} = \left(T_{AB_4(i)} - I_{F_1(i)} = 0\right);$$

$$sic_{F_2(i)} = \left(T_{F_1(i)} - I_{F_2(i)} = 1\right).$$

Note that $F_1(i)$ is the initial section of a cyclic structure (see figure 6.7), which explains why its sequence clause differs slightly from the others.

The sequence clauses listed here clearly do not suffice to solve all coordination problems in the call processes at this level. The proviso-clauses, which were derived on the call-process level, can be translated to the call-phase level, with the aid of the equivalences noted above.
We have noted that the clauses $pric^1_{A(i)}$ and $pric^1_{B(-,i)}$ are implicit in the sequence clauses and can be disregarded. This leaves us with only two clauses:

$pric^2_{A(i)}$ which yields $pric^1_{A_1(i)}$ = (STS(i) = 1), and

$pric^1_{F_2(i)}$ = (STS(i) = 1), which is unaffected by the refinements.

But, there are many more coordination requirements on this level.

(1) The A-side process must 'trigger' a B-side process. By the execution of section $A_2(i,j)$, subscriber j is seized as a B-party, and the B-side call process must be initiated via $F_2(j)$ and $B_1(i,j)$.
(2) For every single execution of $A_2(i,j)$ there must be no more and no less than one execution of the sequence $F_2(j)$; $B_1(i,j)$; $B_2(i,j)$; ...
(3) The identity of the B-party must be registered unambiguously. (Observe that the sequence clauses are indeed ambiguous at this point.)
(4) A subscriber may only be seized as a B-party if its subscriber process is in the FREE state, *and* if it has not been seized already.

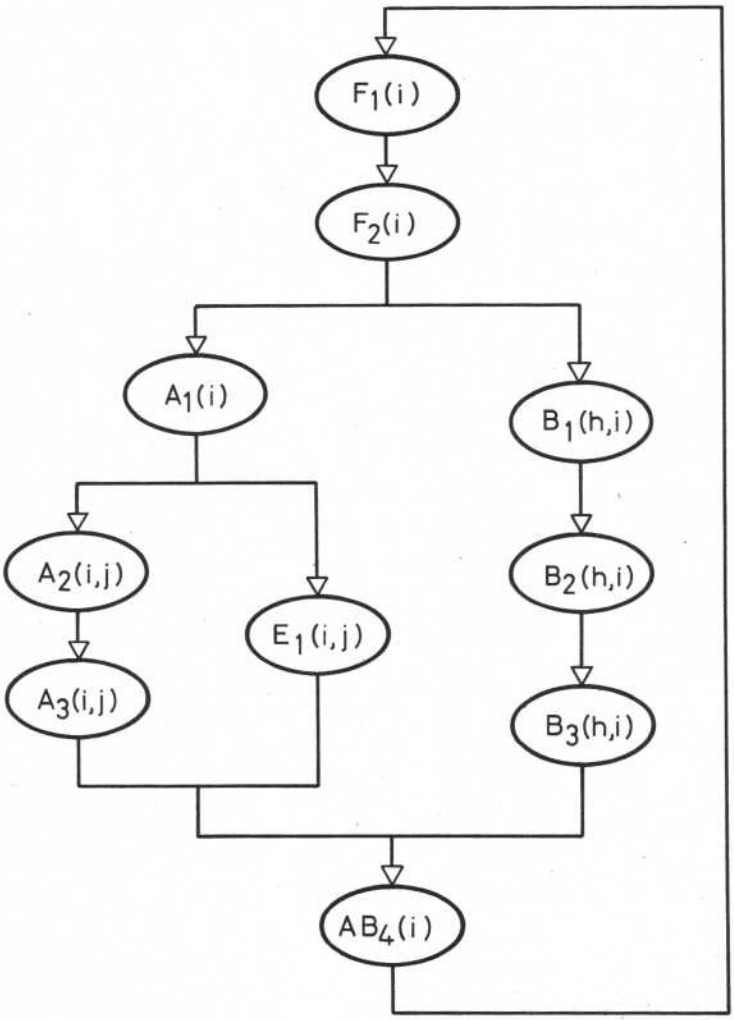

Figure 6.7.
Call Phases in Subscriber Process S(i)

(5) The call process must be ended unambiguously with (in view of the coordination requirements, the most difficult strategy) the one-party-release strategy.

The first requirement leads to the following proviso-clauses:

$$pric^2_{F_2(j)} = \left(I_{A_2(-,j)} - T_{A_2(-,j)} \neq 0\right);$$

$$pric^1_{B_1(i,j)} = \left(I_{A_2(i,j)} - T_{A_2(i,j)} \neq 0\right);$$

$$pric^2_{A_1(j)} = \left(I_{A_2(-,j)} - T_{A_2(-,j)} = 0\right).$$

The last clause is needed to block the initiation of an A-side process when a B-side process is to be initiated. The seizure of a B-party is, according to the proviso-clauses just derived, released when section $A_2(i,j)$ terminates. Clearly, section $A_2(i,j)$ may not be completed before the call answer signal has been received. This call-answer signal is generated in the B-side process, and therefore the execution of $A_2(i,j)$ cannot be terminated before at least the B-side process has been initiated. (We formalize the specific requirements at a lower level, when refining $A_2(i,j)$ itself.)
To avoid the danger of the B-side process being executed repeatedly on a single 'seizure', we must include a condition of the following type:

$$pric^1_{B_3(i,j)} = \left(I_{A_2(i,j)} - T_{A_2(i,j)} = 0\right).$$

The B-side process can, with this clause, no longer be completed before the seizure has been removed. Thus we have also obeyed the second requirement from the list given above.
To guarantee the observance of the third requirement, we introduce a second communal variable ISB(i) (*I*dentity *S*ubscriber *B*). The initial value of ISB(i) is zero. The variable is set to the desired value in section $A_1(i)$, and it is reset to its initial value in section $AB_4(i)$. We thus arrive at the following proviso-clause for sections $A_2(i,j)$ and $E_1(i,j)$:

$$pric^1_{A_2(i,j)} = pric^1_{E_1(i,j)} = (ISB(i) = j).$$

The fourth requirement leads directly to the following clauses:

$$pric^2_{A_2(-,j)} = \left(T_{AB_4(j)} - \left(I_{A_1(j)} + I_{B_1(-,j)}\right) = 0\right);$$

(see section 6.3.2)

$$pric^3_{A_2(-,j)} = \left(I_{A_2(-,j)} - T_{A_2(-,j)} = 0\right);$$

$$pric^2_{E_1(-,j)} = \overline{pric^2_{A_2(-,j)}} \cup \overline{pric^3_{A_2(-,j)}}.$$

Obeying the fifth requirement (unambiguous call ending) is less straightforward. The problems encountered here are characteristic for 'escaping' problems in general (see **Introduction**). Clearly, the on-hook signal of one or both subscribers is to be interpreted as the call-ending signal. The relevant signals are then STS(i) = 0 and STS(j) = 0, where i is the A-party and j the corresponding B-party. To detect these signals, the line scanning functions which were suspended in, respectively, section $F_2(i)$ and $F_2(j)$, must be resumed in the final part of, respectively, $A_2(i,j)$ and $B_2(i,j)$. To avoid ambiguity it is, however, necessary that the call ending signal be irrevocable. This implies that we must require the line scanning functions to be again suspended directly after the detection of an on-hook signal.

It is a more specific implementation problem to decide *when* an on-hook signal should be interpreted as a serious call ending signal. In the peripheral system one may well observe a certain time margin before subscriber signals are passed on to the control level via STS(i) and STS(j). We do however not elaborate this further here.

When cleardown signals can indeed not be revoked, the call ending signal is unambiguous within the call process of the corresponding subscriber. But, there may still exist ambiguity in the coordination of the A-side and B-side call processes. Observe that one of the two subscribers may have reached the FREE state, resumed the line-scanning, and restarted a call process on a new call request before the cleardown in the original process has even been noticed in the other call process. Subscriber i can therefore not use the value of STS(j) as a reliable indication for cleardowns of subscriber j, and neither can subscriber j use the value of STS(i) as a reliable indication for cleardowns of i. To use communal variable STS(i) or STS(j) in this manner would be a violation of the design rules phrased in chapter 4, section 4.5.

To solve the problem we introduce two boolean communal variables CLD(i) and CLD(j) (*Cl*ear-*D*own). CLD(j) is used to signal a call ending from i to j, and CLD(i) is used to signal a call ending from j to i. The initial value of CLD(i) and CLD(j) is FALSE. We require that CLD(j) be set to TRUE in section $A_3(i,j)$, but only if at that moment CLD(i) is FALSE. Similarly, CLD(i) must be set to TRUE in $B_3(i,j)$, but only if at that moment CLD(j) is FALSE. In this manner we can guarantee that CLD(j) is not made TRUE if the B-side process B(-,j) has been completed already, and similarly that CLD(i) is not made TRUE if the A-side process A(i) has already been completed. CLD(j) can then always be reset to its initial value in $AB_4(j)$, and CLD(i) can be reset in $AB_4(i)$.

Assume that section $A_3(i,j)$ contains the selection structure given in figure 6.8.a, and that $B_3(i,j)$ contains the selection structure in figure 6.8.b.

We can formalize the notions on call ending argued above, as follows:

$$pric^1_{A_3(i,-)} = (STS(i) = 0) \cup (CLD(i) = TRUE);$$

$$pric^2_{B_3(-,j)} = (STS(j) = 0) \cup (CLD(j) = TRUE);$$

$$pric^1_{CA_3(i,j)} = (CLD(i) = FALSE) \cap \left(I_{CB_3(i,j)} - T_{CB_3(i,j)} = 0\right);$$

$$pric^1_{CB_3(i,j)} = (CLD(j) = FALSE) \cap \left(I_{CA_3(i,j)} - T_{CA_3(i,j)} = 0\right);$$

$$pric^1_{EA_3(i,j)} = (CLD(i) = TRUE) \cup \left(I_{CB_3(i,j)} - T_{CB_3(i,j)} \neq 0\right);$$

$$pric^1_{EB_3(i,j)} = (CLD(j) = TRUE) \cup \left(I_{CA_3(i,j)} - T_{CA_3(i,j)} \neq 0\right).$$

Section $CA_3(-,j)$ contains the instruction 'CLD(j) := TRUE'.
Section $CB_3(i,-)$ contains the instruction 'CLD(i) := TRUE'.
Section $EA_3(-,-)$ contains the dummy instruction 'skip'.
Section $EB_3(-,-)$ contains the dummy instruction 'skip'.

Observe that variable CLD(j) is set in the A-side call process, and reset in the B-side call process. Similarly, CLD(i) is set in the B-side call process, and reset in the A-side call process.

To refine sections $A_3(i,j)$ and $B_3(i,j)$ completely into the structures of figure 6.8, we must translate the proviso-clause $pric^1_{A_3(i,-)}$ into clause:

$$pric^2_{CA_3(i,-)} = pric^2_{EA_3(i,-)} = (STS(i) = 0) \cup (CLD(i) = TRUE).$$

And similarly, we must translate $pric^1_{B_3(i,j)}$ into:

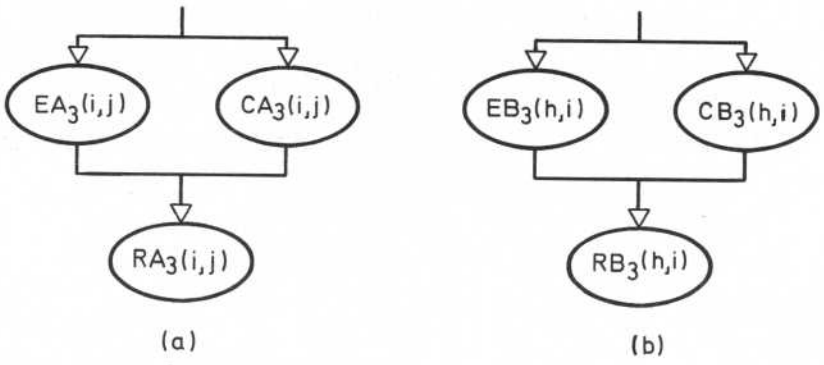

Figure 6.8.
Refinement of $A_3(i,j)$ and $B_3(h,i)$

$$pric^2_{CB_3(i,j)} = pric^2_{EB_3(i,j)} = \left(I_{A_2(i,j)} - T_{A_2(i,j)} = 0\right).$$

And we must translate $pric^2_{B_3(-,j)}$ into:

$$pric^3_{CB_3(-,j)} = pric^3_{EB_3(-,j)} = (STS(j) = 0) \cup (CLD(j) = TRUE).$$

The refined sequence clauses are as follows:

$$sic_{CA_3(i,j)} = sic_{EA_3(i,j)} = \left(T_{A_2(i,j)} - (I_{CA_3(i,j)} + I_{EA_3(i,j)}) = 1\right);$$

$$sic_{CB_3(i,j)} = sic_{EB_3(i,j)} = \left(T_{B_2(i,j)} - (I_{CB_3(i,j)} + I_{EB_3(i,j)}) = 1\right);$$

$$sic_{RA_3(i,j)} = \left(T_{CA_3(i,j)} + T_{EA_3(i,j)} - I_{RA_3(i,j)} = 1\right);$$

$$sic_{RB_3(i,j)} = \left(T_{CB_3(i,j)} + T_{EB_3(i,j)} - I_{RB_3(i,j)} = 1\right).$$

An overview of the coordination clauses derived until now is given below.

$$ic_{F_1(i)} = sic_{F_1(i)};$$

$$ic_{F_2(i)} = sic_{F_2(i)} \cap \left(pric^1_{F_2(i)} \cup pric^2_{F_2(i)}\right);$$

$$ic_{A_1(i)} = sic_{A_1(i)} \cap pric^1_{A_1(i)} \cap pric^2_{A_1(i)};$$

$$ic_{A_2(i,j)} = sic_{A_2(i,-)} \cap pric^1_{A_2(i,j)} \cap pric^2_{A_2(-,j)} \cap pric^3_{A_2(-,j)};$$

$$ic_{E_1(i,j)} = sic_{E_1(i,-)} \cap pric^1_{E_1(i,j)} \cap pric^2_{E_1(-,j)};$$

$$ic_{CA_3(i,j)} = sic_{CA_3(i,j)} \cap pric^1_{CA_3(i,j)} \cap pric^2_{CA_3(i,j)};$$

$$ic_{EA_3(i,j)} = sic_{EA_3(i,j)} \cap pric^1_{EA_3(i,j)} \cap pric^2_{EA_3(i,j)};$$

$$ic_{RA_3(i,j)} = sic_{RA_3(i,j)};$$

$$ic_{B_1(h,i)} = sic_{B_1(-,i)} \cap pric^1_{B_1(h,i)};$$

$$ic_{B_2(h,i)} = sic_{B_2(h,i)};$$

$$ic_{CB_3(h,i)} = sic_{CB_3(h,i)} \cap pric^1_{CB_3(h,i)} \cap pric^2_{CB_3(h,i)} \cap pric^3_{CB_3(h,i)};$$

$$ic_{EB_3(h,i)} = sic_{EB_3(h,i)} \cap pric^1_{EB_3(h,i)} \cap pric^2_{EB_3(h,i)} \cap pric^3_{EB_3(h,i)};$$

$$ic_{RB_3(h,i)} = sic_{RB_3(h,i)};$$

$$ic_{AB_4(i)} = sic_{AB_4(i)}.$$

A more detailed analysis of this abstract description of the coordination rules on the call phase level is postponed until section 6.4. First we will study the refinements to the call-function level.

6.3.4. Call-function level

6.3.4.1. *A-side call process*

6.3.4.1.1. Pre-call phase
The pre-call phase of the A-side call process A (i) may be divided into three call functions, namely:

- Call Initiation, performed in a subsection named $CI_a(i)$.
- Digit Reception, performed in subsection DR(i).
- Number Analysis, performed in subsection NA(i).

For local calls digit reception should be completed before a number analysis can be performed. Still, we may split section DR(i) into two lower level sections $DR_1(i)$ and $DR_2(i)$: one for the *initiation* of a peripheral digit reception process, which includes hunting for a free digit receiver, building a connection to this receiver and initiating it; one for the *termination* of the digit reception process, which includes the removal of the connection to the receiver and the releasing of that receiver. Section $DR_2(i)$ may only be initiated if a complete subscriber number has been received (that is: complete routing information). We introduce a new boolean communal variable for this purpose, named DCP(i) with initial value FALSE. DCP is short for *D*ialling *C*ompleted. DCP(i) can be set to TRUE within (the peripheral process initiated by) $DR_1(i)$.
As sections $DR_2(i)$ and NA(i) are largely independent, they may be executed in concurrency. The resulting structure of the pre-call phase in the A-side process is illustrated in figure 6.9. We recognize a concatenation of a processin line ($CI_a(i)$; $DR_1(i)$) and a parallel path ($DR_2(i)$/NA(i)).
The only 'new' coordination clauses are:

$$pric^1_{DR_2(i)} = pric^1_{NA(i)} = (DCP(i) = TRUE).$$

Observe that communal variable ISB(i) is set in section NA(i).
The refinements of the sequence clauses and the proviso clauses derived on t call-phase level is rather straightforward, and is omitted for brevity.

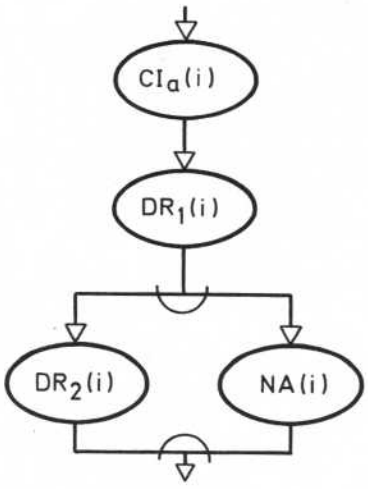

Figure 6.9.
Structure of the A-side Pre-Call Phase

6.3.4.1.2. Initial and terminal call phase
Section $A_2(i,j)$ can be divided into four call functions, as follows:

- The execution of *R*ing-*B*ack toning functions: RB(i,j).
- *C*all *R*outing (path searching) for the A-side of the network: $CR_a(i,j)$.
- *E*stablishing the reserved *R*oute for the A-side: $ER_a(i,j)$.
- *I*nitiation of call *S*upervision and call charging for the caller: $IS_a(i,j)$.

In section $IS_a(i,-)$ the line scanning actions for subscriber i are to be re-initiated to allow for the detection of the call ending signal from the A-party. Note that the line scanning actions were assumed to be suspended directly after the detection of a clear-down signal. As each party must always clear-down before the call process can be ended (and the party can return to the FREE state), the line scanning actions do not need to be suspended at the control level: they can be suspended or masked in the (peripheral) line scanning process, as soon as the party goes on-hook.

From the terminal call phase we have to refine section $RA_3(i,j)$. We arrive at two call functions for this section:

- *T*ermination of call *C*harging for the caller: TC(i,j).
- *Dis*[illegible]*necting* the established *R*oute for the A-side of the network: $RD_a(i,j)$.

The state between the initial and the terminal call phase:

$$\left(T_{IS_a(i,j)} - I_{TC(i,j)} = 1\right),$$

[illegible] be called the *talking state*. In the talking state there is a speech-path

connection between the A-party (the caller; subscriber i) and the B-party (the callee; subscriber j).

We have allowed for independent path searching functions for the A-side and the B-side of the speech-path network. Of course, these functions are not entirely independent. The two selected routes will have to meet somewhere. Let us consider the network configuration of figure 6.1. The A-side call routing function may select a path through the expansion network. The B-side call routing function , which we shall name $CR_b(i,j)$, can then concurrently select a path through the concentrator. When both partial searches have been completed, the two routes can be linked with a route in the interconnection network. If we assume that the latter function is performed in the A-side call process, we must divide section $CR_a(i,j)$ into two subsections $CR_{a1}(i,j)$ and $CR_{a2}(i,j)$, one for the search through the expansion network (E), and one for the search through the interconnection network (I). The call routing on the B-side of the network must clearly be completed before $CR_{a2}(i,j)$ can start. This leads to the coordination clause, for $CR_{a2}(i,j)$:

$$pric^1_{CR_{a2}(i,j)} = \left(T_{CR_b(i,j)} - I_{CR_{a2}(i,j)} = 1 \right).$$

Just like section DR(i) in the pre-call phase, we can divide section RB(i,j) into two subsections: an initial part $RB_1(i,j)$, and a terminal part $RB_2(i,j)$. The terminal part must be executed only *after* the reception of a call-answer signal from the B-side process. For this purpose we introduce another boolean communal variable named CAN(j) (from *C*all *An*swered). The initial value of CAN(j) is FALSE. CAN(j) should be reset to its initial value in $AB_4(j)$.

All in all we have now defined 10 sections in the initial and terminal call phases of the A-side call process (including $EA_3(i,j)$ and $CA_3(i,j)$).
It seems worthwhile to consider how much parallelism can be allowed for in the sequence clauses. The specific nature of the call process requires that some functions must strictly be executed in one given order. We can recognize two such sequences:

$CR_{a1}(i,j)$; $CR_{a2}(i,j)$; $ER_a(i,j)$; $IS_a(i,j)$; $TC(i,j)$; $RD_a(i,j)$, and

$RB_1(i,j)$; $RB_2(i,j)$.

It is further plausible not to initiate $IS_a(i,j)$ before $RB_2(i,j)$ has terminated (that is, before ring-back toning has been completed). This leads to the structure illustrated in figure 6.10. The full refinement of the sequence clauses, and the 'translation' of the proviso-clauses derived earlier is fairly straightforward, and is therefore not detailed here. The only 'new' coordination clauses are $pric^1_{CR_{a2}(i,j)}$ given above, and:

$$pric^1_{ER_a(-,j)} = pric^1_{RB_2(-,j)} = (CAN(j) = TRUE).$$

6.3.4.2. *B-side call process*

6.3.4.2.1. Pre-call phase
In the pre-call phase of the B-side process there is just one function to be performed: call initiation. Section $B_1(h,i)$ is thus merely replaced by a section $CI_b(h,i)$.

6.3.4.2.2. Initial and terminal call phase
Section $B_2(h,i)$ can be refined in four functions, analogous to $A_2(h,i)$.

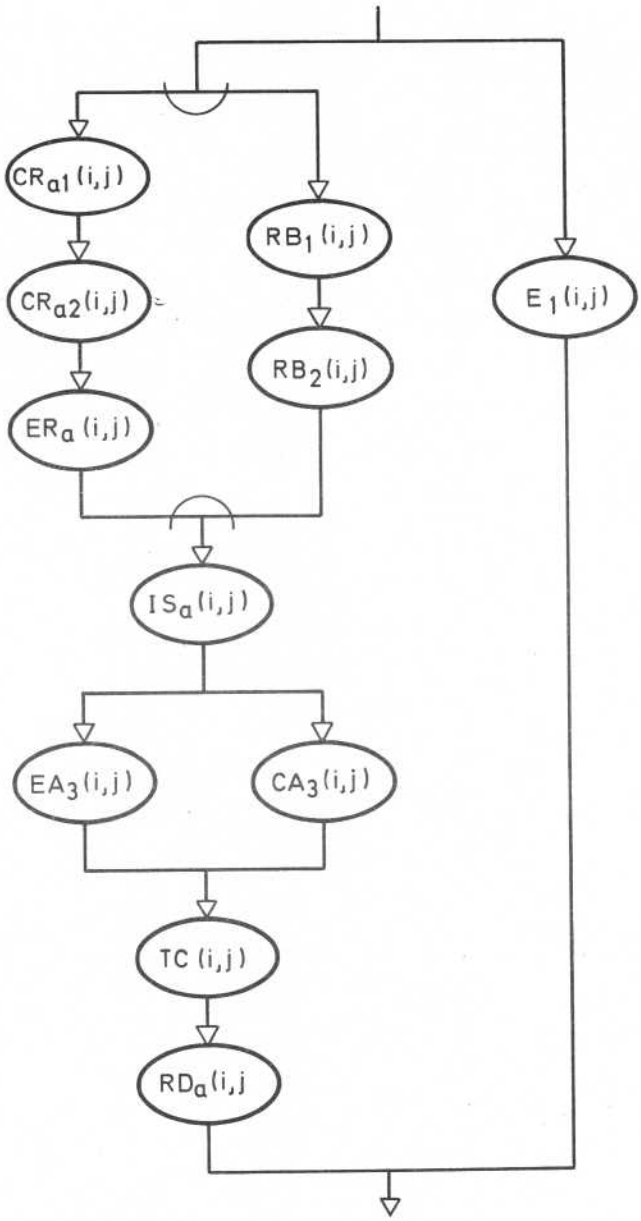

Figure 6.10.
A-side Call Functions

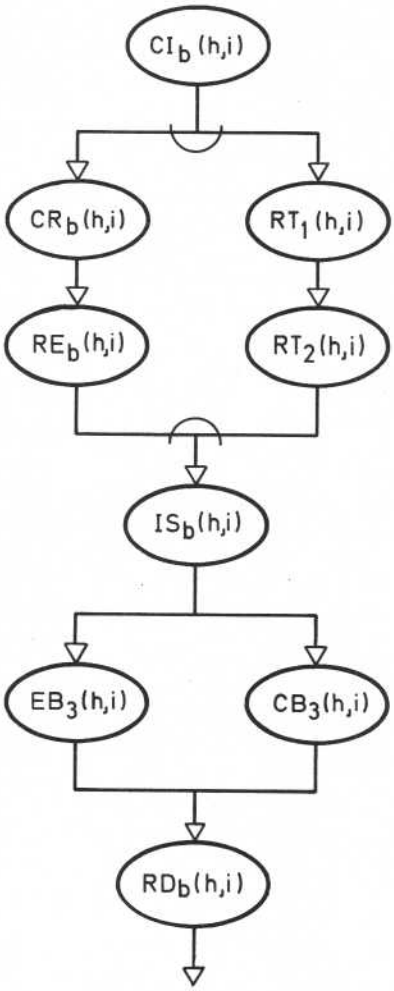

Figure 6.11.
B-side Call Functions

These four functions and the corresponding new section names are:

- The execution of *R*ing *T*oning functions: RT(h,i).
- *C*all *R*outing for the B-side of the network: $CR_b(h,i)$.
- *E*stablishing the selected *R*oute on the B-side: $ER_b(h,i)$.
- *I*nitiation of call *S*upervision, to detect call ending signals from the B-party: $IS_b(h,i)$.

In the terminal call phase we must refine section $RB_3(h,i)$. There is only one function to be performed here: disconnecting the established route on the B-side of the network $RD_b(h,i)$.
Like sections DR(i) and RB(h,i) we will divide section RT(h,i) into an initial part $RT_1(h,i)$ and a terminal part $RT_2(h,i)$. There are then 8 sections in the call phases of the B-side process (including $CB_3(h,i)$ and $EB_3(h,i)$). The resulting structure is illustrated in figure 6.11. The derivation of the refined sequence clauses and proviso-clauses for these 8 sections is again straightforward. The only 'new' clauses are:

$$pric^1_{ER_b(-,i)} = pric^1_{RT_2(-,i)} = (CAN(i) = TRUE).$$

6.3.4.3. *Post-call phase*

The post-call phases of the A-side and B-side processes were combined into one section $AB_4(i)$. There are two functions to be performed in that section:

- *C*ongestion *T*oning: CT(i).
- *C*all *E*nding (restoring the FREE state): CE(i).

The congestion toning function must be skipped if the subscriber considered is already on-hook, that is, if STS(i) = 0. We must therefore include yet a third section in the post-call phase: the 'escape' section $E_2(i)$. The structure is indicated in figure 6.12. The only 'new' clauses are:

$$pric^1_{CT(i)} = \overline{pric}^1_{E_2(i)} = (STS(i) = 1).$$

The expansion of the remaining clauses is deleted (see above).

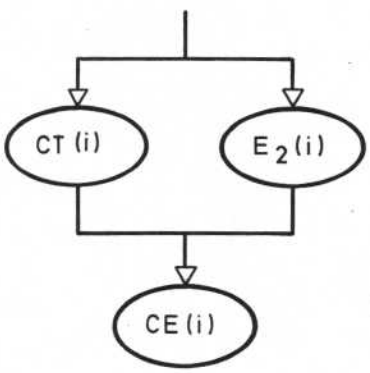

Figure 6.12.
Post-Call Phase

6.3.4.4. *Overview call function level*

There are 29 sections on the call-function level (including $E_1(i,j)$ and $E_2(i)$, $F_1(i)$ and $F_2(i)$). (See figure 6.13.)
The larger part of the rules merely formalize the desired ordering properties of the executions of these functions. Of the initiation conditions 19 do contain proviso-clauses, which formalize more specific coordination rules. The proviso-clauses can be composed from the following 10 conditions:

(1) $STS(i) = 1;$

(2) $ISB(h) = i;$

(3) $CAN(j) = TRUE;$

(4) $CLD(i) = TRUE;$

(5) $DCP(i) = TRUE;$

(6) $T_{CE(i)} - (I_{CI_a(i)} + I_{CI_b(-,i)}) = 0;$

(7) $T_{CR_b(i,j)} - I_{CR_a(i,j)} = 1;$

(8) $I_{CR_{a1}(h,i)} - T_{IS_a(h,i)} = 1 \cup I_{RB_1(h,i)} - T_{IS_a(h,i)} = 1;$

(9) $I_{CR_{a1}(-,i)} - T_{IS_a(-,i)} = 1 \cup I_{RB_1(-,i)} - T_{IS_a(-,i)} = 1;$

(10) $I_{CA_3(i,j)} - T_{CA_3(i,j)} = 1 \cup I_{CB_3(i,j)} - T_{CB_3(i,j)} = 1.$

The compositions are as follows:

$pric_{F_2(i)} = (1) \cup (9);$

$pric_{CI_a(i)} = (1) \cap (\bar{9});$

$pric_{DR_2(i)} = pric_{NA(i)} = (5);$

$pric_{CR_{a1}(h,i)} = pric_{RB_1(h,i)} = (2) \cap (\bar{9}) \cap (6);$

$pric_{E_1(h,i)} = (2) \cap ((9) \cup (\bar{6}));$

$pric_{CR_{a2}(i,j)} = (7);$

$pric_{ER_a(-,j)} = pric_{RB_2(-,j)} = pric_{ER_b(-,j)} = pric_{RT_2(-,j)} = (3);$

$pric_{CA_3(i,-)} = (\bar{1}) \cap (\bar{4}) \cap (\overline{10});$

$pric_{EA_3(i,-)} = (4) \cup (10);$

$pric_{CI_b(h,i)} = (8);$

$pric_{CB_3(h,i)} = (\bar{1}) \cap (\bar{4}) \cap (\overline{10}) \cap (\bar{8});$

$pric_{EB_3(h,i)} = (\bar{8}) \cap ((4) \cup (10));$

$pric_{CT(i)} = (1);$

$pric_{E_2(i)} = (\bar{1}).$

This initial examination of the coordination requirements in a call processing scheme was based on many simplifying assumptions. It will be clear that in any realistic telephone system the speech-path network is not blocking free, and that subscribers are not 'ideal'. For instance, the coordination between sections $DR_1(i)$ and $NA(i)$, $DR_2(i)$, is in practice much more comprehensive than assumed here, as subscribers may dial non-existing numbers, or violate time-limits. Similarly, the ring-toning and congestion-toning functions will be more complex, as the expected subscriber responses (respectively, call answering and the replacing of the receiver on-hook) may stay away. We will study the corresponding extensions and refinements of the coordination schemes in a later section. First of all, however, we will verify the design work performed until now.

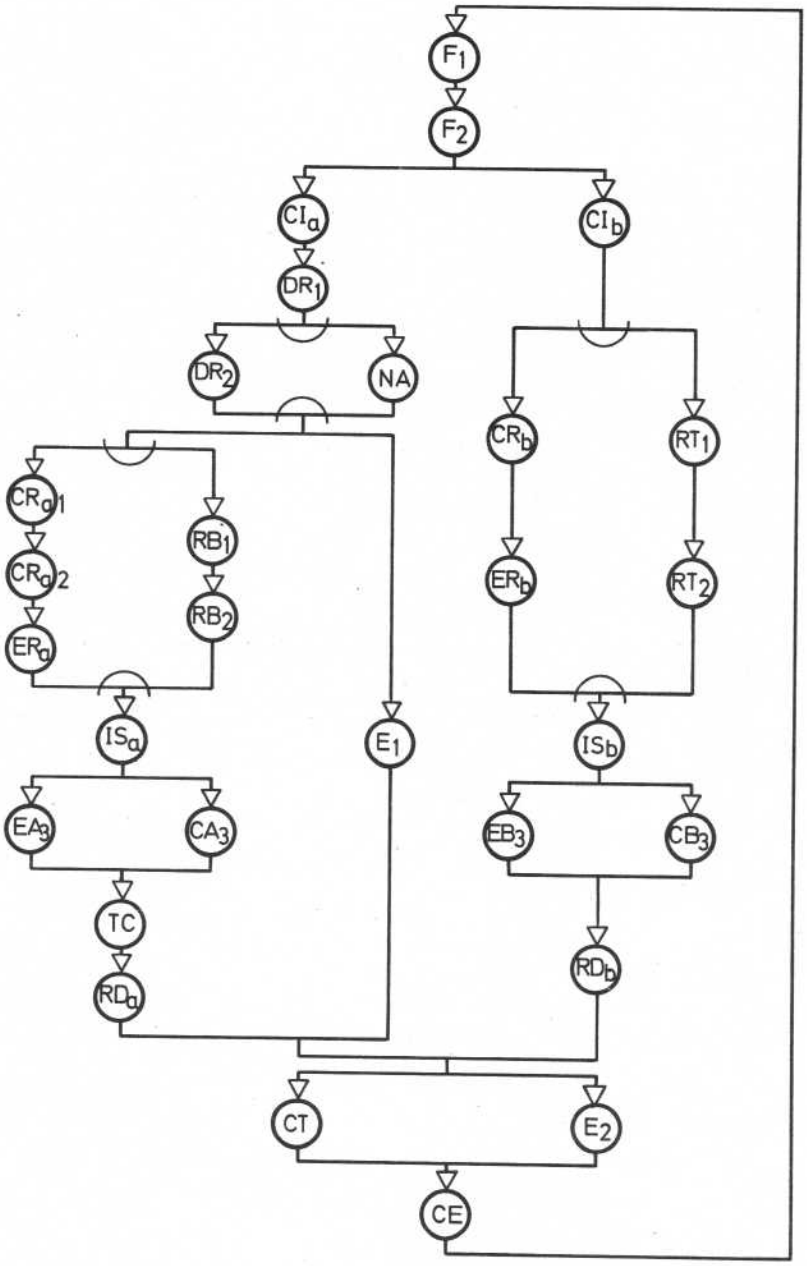

Figure 6.13.
Overview Call Functions

6.4. CORRECTNESS ANALYSIS

We will analyze the abstract solution on an intermediate level of abstraction. The analysis is kept informal. Still, many interesting properties can be established quite convincingly by mere disciplined reasoning. The accuracy of the analysis depends to a great extent on the exactness of the correctness criteria. For instance, we may verify that the three system invariants in which part of the coordination requirements has been formalized are never violated. This would be a trivial task, as the relevant initiation conditions were derived from precisely these three invariants. We may also verify that after each completed call process all gate-variables and all communal variables accessed in that process are restored to a legal initial state. Such a proof would also be simple. Note, that it is a property of structured sections (see chapter 4) that all gate variables are restored to a legal initial state. It is not difficult to prove that the subscriber process as it is refined here is equivalent to a structured section. The initial state of the communal variables is easily restored in section $AB_4(i)$ (or, one level lower, in section CE(i)). In that section ISB(i) is set to zero, CAN(i) is set to FALSE, CLD(i) is set to FALSE, DCP(i) is set to FALSE, and if necessary STS(i) can be set to zero. These proofs will not be detailed here. We can still formulate more specific correctness criteria, of which the truth is less trivial. We will analyze three of these criteria in the next three sections. The three criteria are:

(1) The two design rules for coordination schemes, given in chapter 4, may not be violated.

(2) There may be no *dead-ends* in the call processes. (If a call process is initiated it must terminate via CE(i), within finite time.)

(3) All desirable system states must be reachable. Any number of non-conflicting calls must be able to exist simultaneously. Every subscriber must be able to reach every other subscriber. The latter implies more specifically, that section $A_2(i,j)$, where $i \neq j$, may never be blocked indefinitely.

6.4.1. Observance of the design rules

We need only examine the proviso-clauses here. The sequence-clauses do not violate the design rules as they were derived from the definition of the structured sections (see chapter 4).
First, we will consider the effect of terminations on proviso-clauses, then we will consider the effect of the updating of communal variables.

Terminations
With the aid of the overview in section 6.3.4.4 (see also figure 6.13) we find that the termination of the following four sections may falsify a proviso-clause:

$CE(i)$, can falsify $pric_{E_1(-,i)}$;

$IS_a(h,i)$, can falsify $pric_{F_2(i)}$, $pric_{E_1(-,i)}$, and $pric_{CI_b(h,i)}$;

$CA_3(i,j)$, and

$CB_3(i,j)$, can falsify $pric_{EA_3(i,j)}$ and $pric_{EB_3(i,j)}$.

We consider these four cases one by one.

(1) Terminations of CE(i)

The initiation of section $E_1(-,i)$ is surpassed by the termination of CE(i) if the desired B-party completes a call process directly after the caller has established that this B-party is occupied. The caller will 'escape' via the post-call phase. Clearly, no harm is done by this surpassing. The caller can try to occupy the B-party in a new A-side call process.

(2) Terminations of $IS_a(h,i)$

The termination of this section will remove the 'seizure' of a B-party. It can thereby falsify $ic_{CI_b(h,i)}$, that is, the initiation of the B-side call process of the occupied B-party. Before $IS_a(h,i)$ can be completed however, section RB(h,i) must have been executed. The latter can only initiate if CAN(i) = TRUE, and this can only occur after the initiation of RT(h,i). The initiation of RT(h,i) can only occur if $CI_b(h,i)$ has been completed, which makes the falsification of $ic_{CI_b(h,i)}$ irrelevant. Observe that CAN(i) cannot be true before $CI_b(h,i)$ executes, as it is always reset to FALSE in CE(i).
The termination of $IS_a(h,i)$ can also falsify one clause in $pric_{E1(-,i)}$. This falsification is also irrelevant as this falsification can at that moment not affect the truth of $ic_{E1(-,i)}$. Observe that when $IS_a(h,i)$ terminates subscriber i is running a B-side process which implies that condition $(\bar{6})$ from the list in section 6.3.4.4. is TRUE. If $(\bar{6})$ is TRUE, the effect of a falsification of condition (9) is suppressed in $pric_{E_1(-,i)}$.
Finally, the termination of $IS_a(h,i)$ can falsify a clause in $pric_{F_2(i)}$. As was argued above, section $F_2(i)$ be completed long before this falsification can occur. Note, that subscriber i cannot complete the B-side process and restart another one before $IS_a(h,i)$ has indeed been completed, as a result of condition $(\bar{8})$ in $ic_{CB_3(h,i)}$ and $ic_{EB_3(h,i)}$.

(3) Terminations of $CA_3(i,j)$ and $CB_3(h,i)$

Upon termination of $CA_3(i,j)$ communal variable CLD(j) is TRUE, and similarly, upon termination of $CB_3(h,i)$ communal variable CLD(h) is TRUE.
This implies that condition (4) in the proviso-clauses of $EB_3(i,j)$, respectively, $EA_3(h,i)$, will be TRUE. The latter suppresses the effect of the falsification of condition (10).

Communal variables
We have introduced five communal variables: STS(i), ISB(i), CAN(i), CLD(i), and DCP(i). We will consider them one by one.

(1) STS(i)

The setting of STS(i) to zero can falsify condition (1), and thereby the initiation conditions of sections $F_2(i)$, $CI_a(i)$, and CT(i).
The setting of STS(i) to one can falsify condition $(\bar{1})$, and thereby the initiation conditions of sections $CA_3(i,-)$, $CB_3(-,i)$ and $E_2(i)$.
STS(i) can however only be set to zero if, in the A-side process at least section $IS_a(i,-)$ has been executed, or if, in the B-side process at least $IS_b(-,i)$ has been executed and section CE(i) has not yet been executed. This implies that $F_2(i)$ and $CI_a(i)$ cannot be active at that time. STS(i) can however be set to zero directly after the initiation of CT(i). This need not cause any errors though. Congestion toning can then be completed in a minimal time, and the call process can be terminated normally.
The setting of STS(i) to one can only occur in the FREE state, or in the pre-call and initial-call phase of the B-side process. None of the sections $CA_3(i,-)$, $CB_3(-,i)$, or $E_2(i)$ can be active at that time.

(2) **ISB(i)**

Only the resetting of ISB(i) to FALSE in section CE(i) may falsify an initiation-condition, namely the initiation condition of an initial call phase ($A_2(i,j)$). Clearly, the initial call phase is always completed before the post-call phase, in which CE(i) occurs.

(3) **CAN(i)**

As with ISB(i), only the resetting of CAN(i) to FALSE in CE(i) can falsify condition (3) and thereby the initiation conditions of four initial call phase sections. These cannot be active when CE(i) is being executed.

(4) **CLD(i)**

The setting of CLD(i) to TRUE can falsify condition ($\bar{4}$) and thereby $pric_{CA_3(i,-)}$ or $pric_{CB_3(-,i)}$. However, as CLD(i) can only be set to TRUE in section $CA_3(-,i)$ or $CB_3(i,-)$, condition ($\overline{10}$) must be FALSE at that time, which suppresses the effect of the falsification of ($\bar{4}$).
The setting of CLD(i) to FALSE in CE(i) can falsify $pric_{EA_3(i,-)}$ or $pric_{EB_3(-,i)}$. But clearly, when CE(i) executes, sections $EA_3(i,-)$ and $EB_3(-,i)$ cannot be active.

(5) **DCP(i)**

Only the setting of DCP(i) to FALSE in CE(i) can falsify condition (5) and thus $pric_{DR_2(i)}$ and $pric_{NA(i)}$. Sections $DR_2(i)$ and NA(i) are part of the pre-call phase and can therefore not be active when the post-call phase is being executed.

We conclude that the two design rules from chapter 4 are not violated.

6.4.2. Dead ends

We will consider the effects of the proviso-clauses. (The sequence clauses cannot introduce dead ends, see chapter 4.) Here we will concentrate on a single subscriber process. Higher level blockings will be considered in the reachability analysis. We first examine under what conditions proviso-clauses can introduce dead ends in the general control-flow structures: processing lines, selection structures, and parallel paths[1].

(1) **Processing lines**

The proviso-clause of each subsequent section in the line must either *be* true when the corresponding sequence clause becomes true, or it must *become* true within a finite time after this event.
Let $T_j(sic_x)$ denote the j-th time at which sic_x becomes TRUE after system initialization. Further, let $sic_x(t)$ be the value of sic_x at time t.
A processing line of m sections, s_1, s_2, ..., s_m, is then free of dead ends if the following condition is fulfilled:
For all sections s_i with $1 \leq i \leq m$ and all positive values of j:

$$\left(\exists \Delta t_j^i\right)\left[\Delta t_j^i \geq 0 \rightarrow pric_{s_i}\left(t_j(sic_{s_i}) + \Delta t_j^i\right) = \text{TRUE}\right],$$

— — — — —

(1) Iteration structures do not occur in the coordination scheme to be verified. To phrase a comparable condition for iteration structures requires the formalization of the notion of 'termination'. We do not elaborate this here.

where Δt_j^i is finite for all values of i and j (but not bounded).

(2) **Selection structures**

At least one of the proviso-clauses of the initial sections in the branches of the selection structure must be true, or become true within finite time, when the corresponding sequence clauses are true. If the initial sections of the branches are named $s_1, s_2, \ldots, s_m$, the following condition must hold[1]:

$$(\exists i)\ (\forall j)\ \left(1 \leqslant i \leqslant m \text{ AND } j > 0 \rightarrow \left(\exists \Delta t_j^i\right)\ \left(\Delta t_j^i \geqslant 0 \rightarrow pric_{s_i}\left(t_j(sic_{s_i}) + \Delta t_j^i\right) = TRUE\right)\right) \text{ AND}$$

$$(\forall j)\ \left(j > 0 \rightarrow \left(\exists \Delta t_j\right)\ \left(\Delta t_j \geqslant 0 \rightarrow pric_{s_{m+n}}\left(t_j(sic_{s_{m+n}}) + \Delta t_j\right) = TRUE\right)\right),$$

where section s_{m+n} is the terminal section of the selection structure. Note, that for selection structures we have:

$$(\forall i)\ \left(1 < i < m \rightarrow sic_{s_i} = sic_{s_{i+1}} \rightarrow (\forall j)\ \left(t_j(sic_{s_i}) = t_j(sic_{s_{i+1}})\right)\right).$$

(3) **Parallel paths**

If the sections $s_1, \ldots, s_m$ are the initial sections in the branches of a parallel path, and s_{m+n} is the corresponding terminal section, the following condition must hold[2]:

$$(\forall i)\ (\forall j)\ \left(\left(1 \leqslant i \leqslant m \text{ OR } i = m + n\right) \text{ AND } j > 0 \rightarrow \left(\exists \Delta t_j^i\right)\ \left(\Delta t_j^i \geqslant 0 \rightarrow pric_{s_i}\left(t_j(sic_{s_i}) + \Delta t_j^i\right) = TRUE\right)\right).$$

Application

The absence of dead-ends in the subscriber processes is now readily established. 10 of the 29 sections on the call function level contain no proviso-clauses at all (which is equivalent to the 'default-clause' TRUE). These 10 sections can therefore not cause dead-ends. We will however perform the analysis on the call phase level. On the call phase level, there were 10 sections. Of these 10 sections only 7 contain proviso-clauses. We will consider these 7 sections below.

Sections $A_1(i)$ and $B_1(h,i)$ are initial sections in the branches of a selection structure. (See figure 6.13.a.) The condition is then that each time when:

$$sic_{A_1(i)} = sic_{B_1(-,i)} = TRUE, \text{ either } pric_{A_1(i)} = TRUE, \text{ or}$$

$$pric_{B_1(-,i)} = TRUE,$$

within finite time (or both). A pre-condition for both events is the execution of $F_2(i)$, which implies:

$$STS(i) = 1 \text{ OR } \left(I_{A_2(-,i)} - T_{A_2(-,i)} \neq 0\right).$$

If either of these conditions is TRUE, it cannot become FALSE before either $A_1(i)$ or $B_1(-,i)$ has been executed. The truth of the second clause implies the truth of $pric_{B_1(-,i)}$. If the second clause is not TRUE then the first clause

(1) See also figure 4.9 in chapter 4.
(2) See also figure 4.11 in chapter 4.

must be TRUE, which implies the truth of $pric_{A_1(i)}$.

Sections $A_2(i,-)$ and $E_1(i,-)$ are the initial sections in the branches of a selection structure. Under the assumption of an ideal subscriber behavior:

$$pric^1_{E_1(i,j)} = pric^1_{A_2(i,j)} = (ISB(i) = j),$$

is always TRUE for at least one j, after the execution of $A_1(i)$. The remaining clauses in $pric_{E_1(i,j)}$ and $pric_{A_2(i,j)}$ are complementary, which implies that at least one of these clauses must be TRUE.
Sections $F_2(i)$, $A_3(i,j)$ and $B_3(h,i)$ remain to be considered. The duration of the interval Δt in these cases depends to a large extent on specific subscriber behavior. These intervals correspond to the duration of the FREE state, respectively, the TALKING state. Dead-ends in the subscriber processes may then occur at these points. These dead-ends are however not caused by the system, but by its environment (the subscribers). The possibility that a subscriber remains in the FREE state, or in the TALKING STATE, forever is present, but not undesirable.

6.4.3. Reachability

In the first part of this section we study the reachability of system states. In a second part we study the reachability of specific subscriber states.

6.4.3.1. *System states*

We abstract from individual subscribers and concentrate on classes of subscribers in equivalent states. First, we introduce a few symbols and abbreviations.

N : the number of subscribers ($N = |\tau|$);

α_p : the set of subscribers currently executing an A-side pre-call phase:

$$\alpha_p = \left\{ i:\ i \in \tau \text{ AND } \left(I_{A_1(i)} - (I_{A_2(i,-)} + I_{E_1(i,-)}) = 1 \right) \right\};$$

α_c : the set of subscribers currently executing an A-side initial call-phase:

$$\alpha_c = \left\{ i:\ i \in \tau \text{ AND } \left(I_{A_2(i,-)} - I_{A_3(i,-)} = 1 \right) \right\};$$

$\alpha\beta_e$: the set of subscribers currently executing the A-side or B-side terminal call phase, or the escape part ($E_1(i,j)$) of the A-side call process, or the post-call phase:

$$\alpha\beta_e = \left\{ i:\ i \in \tau \text{ AND } \left(\left(I_{A_3(i,-)} - T_{A_3(i,-)} = 1 \right) \text{ OR } \left(I_{E_1(i,-)} - T_{E_1(i,-)} = 1 \right) \text{ OR } \left(I_{B_3(-,i)} - T_{B_3(-,i)} = 1 \right) \text{ OR } \left(T_{A_3(i,-)} + T_{E_1(i,-)} + T_{B_3(-,i)} - I_{F_1(i)} = 1 \right) \right) \right\}$$

β_p : the set of subscribers currently executing the B-side pre-call phase, or the B-side initial call phase, plus the subscribers which were seized as a B-party but which have not yet initiated a B-side call process:

$$\beta_p = \left\{ i:\ i \in \tau \text{ AND } \left(\left(I_{B_1(-,i)} - I_{B_3(-,i)} = 1 \right) \text{ OR } \right. \right.$$

$$(\exists h)\left(h \in \tau \text{ AND } \left(I_{A_2(h,i)} - T_{A_2(h,i)} = 1\right)\right)\Big)\Big)\Big\}$$

Two other sets, which differ only slightly from sets α_c and β_p, will prove helpful.

$$\alpha'_c = \left\{i: i \in \tau \text{ AND } \left(I_{A_2(i,-)} - T_{A_2(i,-)} = 1\right)\right\}$$

$$\beta'_p = \left\{i: i \in \tau \text{ AND } \left(\left(I_{B_1(-,i)} - T_{B_2(-,i)} = 1\right) \text{ OR } (\exists h)\left(h \in \tau \text{ AND } \left(I_{A_2(h,i)} - T_{A_2(h,i)} = 1\right)\right)\right)\right\}$$

$$\alpha\beta = \alpha_p \cup \alpha_c \cup \beta_p \cup \alpha\beta_e.$$

The set of FREE subscribers can be found by subtracting set $\alpha\beta$ from set τ[1]. Note, that all these sets are mutually disjoint.

$\psi(i)$: the set of all subscribers that have chosen to establish a connection to subscriber i:

$$(\forall i)\ (i \in \tau \rightarrow \psi(i) = \{j: j \in \tau \text{ AND } ISB(j) = i\}).$$

In the system studied here, each subscriber can establish or request only one connection at a time, and therefore:

ψ: the set of all subscribers which occur in one of the sets $\psi(i)$ is a subset of τ:

$$\psi = \underset{i \in \tau}{\Omega}\ \psi(i) \subseteq \tau.$$

Using set ψ we can define another useful set $\bar{\psi}$ as follows:

$$\bar{\psi} = \underset{i \in \tau}{\Omega}\ \bar{\psi}(i), \text{ where}$$

$$\bar{\psi}(i) = \{h: h \in \tau \text{ AND } i \in \psi(h)\} \hat{=} \{h: h \in \tau \text{ AND } ISB(i) = h\}.$$

Observe that set $\bar{\psi}(i)$ can contain at most one element, unlike set $\psi(i)$. Observe also that:

$$(i \in \psi(j)) \rightarrow (j = \bar{\psi}(i)) \rightarrow (i \in \psi(\bar{\psi}(i))).$$

The following conditions hold invariantly:

$$|\alpha\beta| \leqslant N \text{ AND } \alpha'_c \subseteq \alpha_c \text{ AND } \beta'_c \subseteq \beta_c \text{ AND } |\alpha'_c| \leqslant \tfrac{1}{2}N \text{ AND } |\beta'_c| \leqslant \tfrac{1}{2}N$$

$$(i \in \alpha'_c \rightarrow \bar{\psi}(i) \in \beta_p) \text{ AND } (j \in \beta'_p \rightarrow (\exists h)\ (h \in \psi(j) \rightarrow h \in \alpha_c) \text{ AND }$$

$$(\forall h_1)\ (\forall h_2)\ (h_1 \in \psi(j) \text{ AND } h_2 \in \psi(j) \text{ AND } h_1 \in \alpha'_c \rightarrow h_2 \notin \alpha'_c).$$

Finally, we have:

$$|\alpha'_c| > 0 \rightarrow |\beta_p| > 0 \text{ AND } |\alpha_c| > 0, \text{ and}$$

$$|\beta'_p| > 0 \rightarrow |\alpha_c| > 0 \text{ AND } |\beta_p| > 0.$$

— — — — —

(1) For set subtraction we will use the symbol "-" (minus operation).

We can now define the following 8 system states:

(1) Empty state: $|\alpha\beta| = 0$;

(2) $|\alpha_p| > 0$ AND $|\alpha\beta - \alpha_p| = 0$;

(3) $(|\alpha_c| > 0$ OR $|\beta_p| > 0)$ AND $|\alpha\beta - (\alpha_c \cup \beta_p)| = 0$;

(4) $|\alpha\beta_e| > 0$ AND $|\alpha\beta - \alpha\beta_e| = 0$;

(5) $|\alpha_p| > 0$ AND $(|\alpha_c| > 0$ OR $|\beta_p| > 0)$ AND $|\alpha\beta - (\alpha_p \cup \alpha_c \cup \beta_p)| = 0$;

(6) $|\alpha_p| > 0$ AND $|\alpha\beta_e| > 0$ AND $|\alpha\beta - (\alpha_p \cup \alpha\beta_e)| = 0$;

(7) $(|\alpha_c| > 0$ OR $|\alpha_p| > 0)$ AND $|\alpha\beta_e| > 0$ AND $|\alpha\beta - (\alpha_c \cup \beta_p \cup \alpha\beta_e)| = 0$;

(8) $|\alpha_p| > 0$ AND $(|\alpha_c| > 0$ OR $|\beta_p| > 0)$ AND $|\alpha\beta_e| > 0$.

Observe that this set of states is complete, and that each state is unique (the states are mutually disjoint).
With these 8 states we can make a state diagram. The complete diagram is given in figure 6.14. In order not to complicate the diagram we have used annotations on the arcs, which specify the type of transition and the conditions under which it can occur. (A system diagram without such annotations would contain many more states.) The types of transitions are motivated and described below. A rigid proof that these transitions are indeed possible under the conditions specified, and no more than these, is not included, but should be straightforward.

Transition-types

- Type 1 : *new call request*
  Effect : $(\exists i)$ $(i \in \tau$ AND $i \notin \alpha\beta$ AND STS(i) = 1: $\alpha_p := \alpha_p \cup i)$
  Condition : $|\alpha\beta| < N$
  Motivation: There must be at least one FREE subscriber in the system, which can generate the call request.

- Type 2 : *return to the FREE state*
  Effect : $(\exists i)$ $(i \in \alpha\beta_e$: $\alpha\beta_e := \alpha\beta_e - i)$
  Condition : $|\alpha\beta_e| > 0$
  Motivation: There must at least be one subscriber in the post-call phase. Observe, that the post-call phase contains no dead-ends.

  We distinguish between two cases:

  Type 2a: $|\alpha\beta_e| = 1$, and
  Type 2b: $|\alpha\beta_e| > 1$.

  A transition of type 2a leads from macro-state {4,6,7,8} to {1,2,3,5} (see figure 6.14).

- Type 3 : *establish new call*
  Effect : $(\exists i)$ $(i \in \alpha_p$ AND $\bar{\psi}(i) \notin \alpha\beta$: $\alpha_p := \alpha_p - i$ AND $\alpha_c := \alpha_c \cup i$ AND $\beta_p := \beta_p \cup \bar{\psi}(i))$
  Condition : $|\alpha_p| > 0$ AND $|\alpha\beta| < N$
  Motivation: The desired B-party must be FREE. A call can only be established via an A-side call process in the pre-call phase. Observe also that $\bar{\psi}(i) \notin \alpha\beta$ implies $(\forall j)$ $(j \in \alpha'_c \rightarrow \bar{\psi}(j) \neq \bar{\psi}(i))$

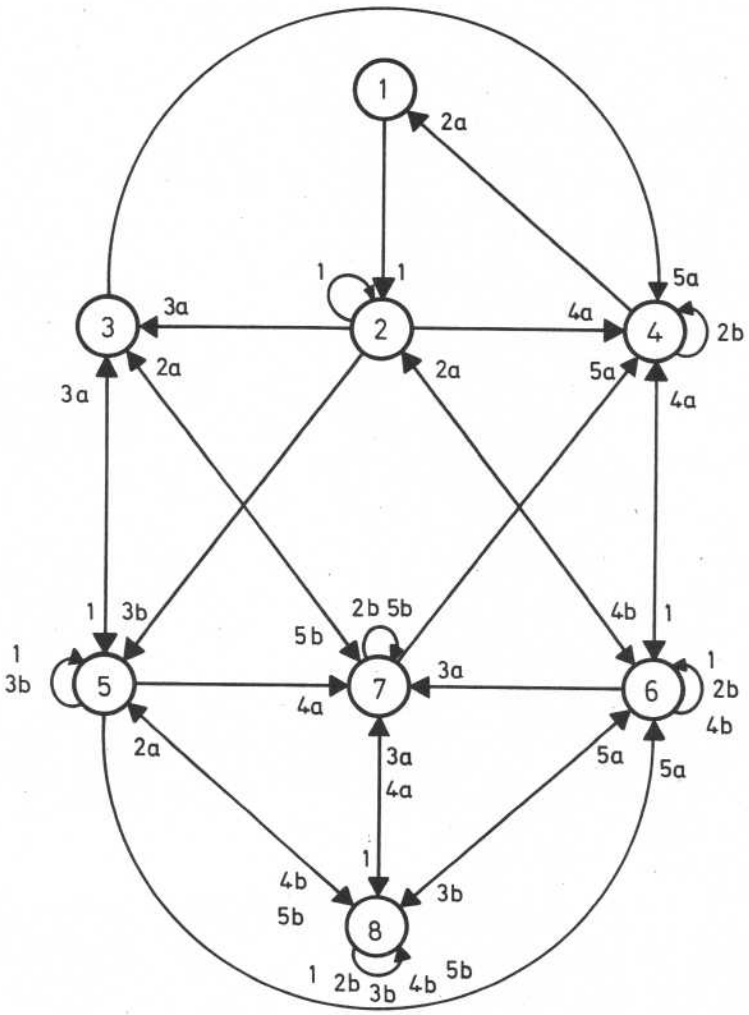

Figure 6.14.
System State Transition Diagram

We distinguish between two cases:

Type 3a: $|\alpha_p| = 1$, and
Type 3b: $|\alpha_p| > 1$.

- Type 4 : *escape*
  Effect : $(\exists i)\ (i \in \alpha_p$ AND $\bar{\psi}(i) \in \alpha\beta: \alpha_p := \alpha_p - i$ AND $\alpha\beta_e := \alpha\beta_e \cup i)$
  Condition : $|\alpha_p| > 0$
  Motivation: Note that: $\bar{\psi}(i) \in \alpha\beta$ implies $(\exists h)\ (h \in \psi(\bar{\psi}(i)) \rightarrow h \in \alpha'_c)$.

We distinguish between two cases:

Type 4a: $|\alpha_p| = 1$, and
Type 4b: $|\alpha_p| > 1$.

- Type 5 : *call ending*
  Effect : $(\exists i)\ (i \in \alpha_c: \alpha_c := \alpha_c - i$ AND $\alpha\beta_e := \alpha\beta_e \cup i)$ OR
  $(\exists i)\ (i \in \beta_p: \beta_p := \beta_p - i$ AND $\alpha\beta_e := \alpha\beta_e \cup i)$
  Condition : $|\alpha_c| > 0$ OR $|\beta_p| > 0$
  Motivation: The caller and the callee can end a call independently.

We distinguish between two cases:

Type 5a: $|\alpha_c \cup \beta_p| = 1$, and
Type 5b: $|\alpha_c \cup \beta_p| > 1$.

The important thing to note for the state diagram in figure 6.14 is that a sequence of transitions between any two states in the system exists. None of the 8 system states is unreachable; none of the system states corresponds to a deadlock-state.

6.4.3.2. *Subscriber states*
The reachability of all desirable system states does not imply the reachability of all desirable specific subscriber states. As a result of the inter-process relations the subscriber processes may well block each other in undesirable ways. With the term 'inter-process relations' we mean the relations between the call processes of different subscribers. Fortunately, there are only a few of these inter-process relations. They were defined in two situations:

- To prevent two or more subscribers from establishing a call to the same callee, at the same time, and
- To coordinate the A-side call process of a caller with the B-side call process of the corresponding callee.

The first type of relation was formalized in conditions 6 and 9 (see section 6.3.4.4) which are evaluated in the initiation conditions of sections:

$$F_2(i),\ CI_a(i),\ CR_{a1}(-,i),\ RB_1(-,i),\ E_1(-,i).$$

The second type of relation was formalized in conditions 3, 4, 7, 8, and 10, which are evaluated in the initiation conditions of sections:

$$CR_{a2}(-,j),\ ER_a(-,j),\ RB_2(-,j),\ ER_b(-,j),\ RT_2(-,j),$$
$$CA_3(i,-),\ EA_3(i,-),\ CI_b(h,i),\ CB_3(h,i),\ EB_3(h,i).$$

To study the effect of each type of relation one can draw blocking and releasing graphs (for instance, for 3 subscriber processes i, j, and h, where i and h attempt to call j, while j attempts to call yet another subscriber k).

The number of attainable states (and potential blocking situations) rises quite rapidly with the number of subscriber processes studied. We will therefore follow another procedure here, which is independent of the value of N. Instead of considering whether undesirable blockings exist in a system of arbitrary size, we consider under what conditions *specific* sections may be blocked. By way of example we analyze the possibility of undesirable blockings of one of the most interesting sections: section $A_2(i,j)$ (corresponding to section $CR_{a1}(i,j)$ or $RB_1(i,j)$ on the call function level).
The interesting conditions are then conditions 6 and $\bar{9}$. The blocking condition for section $A_2(-,i)$ is (with respect to these conditions):

$$\left(T_{CE(i)} - (I_{CI_a(i)} + I_{CI_b(-,i)}) = 1\right) \cup$$

$$\left(I_{CR_{a1}(-,i)} - T_{IS_a(-,i)} = 1 \text{ OR } I_{RB_1(-,i)} - T_{IS_a(-,i)} = 1\right) \text{ permanently.}$$

We first consider the second clause in the blocking condition.
The only inter-process relation which can cause the permanent truth of this second clause is formalized in condition 3 (via the initiation conditions of sections $ER_a(h,i)$, and $RB_2(h,i)$, where h is the subscriber which seized i). The permanent truth of CAN(i) = FALSE, however, implies that also $B_1(h,i)$ is blocked. This contradicts the assumption that h has seized i, and thereby made the second clause in the blocking condition of $A_2(-,i)$ TRUE. The conclusion is that this second clause cannot be TRUE permanently.

Next, we consider whether the first clause in the blocking condition of $A_2(-,i)$ can be TRUE permanently. This would imply that the subscriber process S(i) contains a dead-end. We have seen earlier that there is indeed one such potential dead-end corresponding to the TALKING state. Another special case is the case in which subscriber i attempts to call itself. Each time subscriber i will then find section $A_2(i,i)$ blocked on the first clause. The conclusion must be that section $A_2(-,i)$ can indeed be blocked indefinitely on the first clause of its blocking condition, but not in an undesirable way.

Alternate and overlapping blockings of the two clauses are however not possible, as the first clause cannot become FALSE while the second clause remains TRUE (condition $\bar{8}$ in $pric_{EB_3(-,i)}$ and $pric_{CB_3(-,i)}$ prevents this). (See section 6.3.4.4.)
Apart from the potential blockings which we have encountered here, there are some time-dependent blocking effects between subscriber processes. As an example of such a blocking one may consider the case in which two subscribers attempt to call each other repeatedly, and fail each time as they run their call processes simultaneously. We will study a number of these cases in detail, in section 6.6.

6.5. FURTHER REFINEMENTS AND EXTENSIONS

We will now consider some interesting aspects of further refinements in the call processing scheme, and the extension of that scheme for more 'realistic' assumptions about the system and its subscribers. We shall consider how the scheme can be adapted to erroneous subscriber behavior (violation of time-limits, premature cleardowns, dialling of non-existing numbers) and to less ideal systems (congestion in hardware, hardware- and software faults). We then consider the extension of the call processing scheme to non-local call types, and the extensions for a number of special subscriber facilities (ring-back, waiting-queue, follow-me, etc.).

6.5.1. Further refinements

Let us consider some of the more interesting coordination problems which occur only at still lower levels of abstraction than the call-function level.
A nearly classic type of coordination problem occurs when two concurrent processes access the same shared data. (A sort of 'musical chairs' problem with just one chair.) Ideally one should refine all call-functions to the implementation level and make a 'system table' of all shared data, indicating which sections share which data. Instead of giving such a full elaboration of the example studied here, we will make some general assumptions about an implementation, which are then used to illustrate the major types of coordination problems involved.

The most important shared (hardware) objects in a telephone system are:

- The network circuitry (links, crosspoints, lines).
- The service circuits (tone senders, digit receivers, etc.).
- Peripheral equipment (scanners, drivers, markers, etc.).

The status of these shared objects is represented in specific shared data fields and data structures. There are still other shared data structures which do not represent physical objects but 'shared information' on the call processing in general. An example of the latter type is the *transient call record* (t.c.r.) which contains all relevant information for one specific call. The t.c.r. will specify the links and crosspoints occupied for a specific call, the service circuits in use, etc. The access to this shared data must be coordinated to avoid unintended interferences. The access to the physical objects mentioned earlier is generally protected via the coordination schemes on the shared data. Fortunately, the types of coordination required in each case is quite similar. Only one process (section) may occupy and use a shared object at a time. We can model this type of coordination with simple exclusion rules, which can in turn be implemented with the simplest primitives from the 'semaphore class' (see chapter 1: **The test&set operations**). Observe that in the 'ideal system' which we study in this phase, there will be sufficient physical objects to service all requests without undue delays. (We consider the extensions for less ideal systems later.)
In a structured software package for telephone systems we may expect to find a *linked list* of shared objects. (More precisely: a linked list of the data structures representing the shared physical objects.) Plus the appropriate *extract* and *insert* operations. The latter operations are mutually exclusive per list, in all combinations. For instance, if L is the name of the linked list of free t.c.r.'s, then the operation *extract(L)* will occur in each section $CI_a(i)$ and $CI_b(-,i)$, and the operation *insert(L)* will occur in each section CE(i).

For proper system management the execution of the insert operations should have a priority over the execution of extract operations, especially under *overload*

conditions. A strategy in which the extract operation would have priority over the insert operation could lead to a trivial deadlock situation.

Remark:

The implication of this observation is that the tasks which are traditionally seen as *high* priority tasks in the telephone system in view of their real-time requirements (e.g. line scanning actions in the free state and the pre-call phase) should in view of the coordination requirements be considered as *low* priority tasks. To prevent system overload it is more important to detect call ending, than it is to detect new call requests. This point confirms that real-time requirements should not be taken as a guide-line for system design (cf. section 5.1, and 6.1).

Unlike the service circuits, or the peripheral equipment, it is rather difficult to subdivide the network circuitry into distinct shared objects. One may regard the crosspoint as an entity; one may equally well regard the switches as entities. (A switch is a module of crosspoints and links.)
The usual way to perform call routing in telephone networks is by performing exclusive logical AND operations on binary words in memory, which contain the busy/free bits of the crosspoints per switch. (One binary word corresponds to one specific switch.) The status of all crosspoints that are required in a specific route through the switch considered is then tested via a single AND operation. In the call routing strategy outlined earlier we may (for simplicity) consider the networks E, I, and C (see figure 6.1) as entities and formalize the corresponding coordination requirements in an invariant, as follows:

$$(\forall i)\ (\forall j)\ \left(I_{CR_{a1}(i,j)} - T_{CR_{a1}(i,j)} \leqslant 1 \right),$$

and similarly for $CR_{a2}(i,j)$ and $CR_b(i,j)$. From these invariants one can then derive simple exclusion rules for the sections concerned.

One can also allow for more concurrency in the path searching actions per switch. If, by chance, two concurrent path searches claim the same cross point(s), only one of these conflicting claims can be granted, and the other path searches will have to be repeated. The searching of a route is then concurrent; the actual occupation of a single crosspoint is exclusive. It is not possible to predict which of the two strategies would be more efficient in time. It would however be interesting to compare the two methods under realistic traffic conditions. We leave this as a subject for further research.

6.5.2. Extensions

6.5.2.1. *Non-ideal systems*

We will start this section by examining the main causes of fallibility in the telephone system. Then we still study the appropriate extensions of the digit reception and number analysis functions, of the ring toning and ring-back toning functions, and of congestion toning. The extensions for the other functions are far less comprehensive, and therefore not detailed here.

Causes

Each function essentially has two possible outcomes in a realistic system: either the function can be completed as intended, or it cannot be completed. The latter can be the result of one (or more) of the following 5 causes:

(1) A *premature cleardown*. The calling subscriber goes on-hook before the call is established.
(2) *Congestion* in the network, the circuitry, or the memory, due to overload.

(3) The violation of a *time-limit*. The calling subscriber may take too much time chosing a number, the called subscriber may take too much time to answer a call, a subscriber may wait too long with returning his receiver on-hook after the reception of congestion tones, etc.
(4) The reception of an *invalid number*. The system may detect a 'decimal' number of more than 10 digits. The subscriber may dial a non-existing number.
(5) The occurrence of hardware or software *faults*.

An other cause of premature call ending has been considered before:

(6) The caller dials the number of an occupied subscriber.

In the post-call phase one may generate different types of congestion tones, to give the subscriber some insight into the cause of the call ending. Three different types of congestion tones should suffice:

- Type 1: The cause lies with the system (cause 2 and 5).
- Type 2: The cause lies with the caller (causes 1, 3, and 4)[1].
- Type 3: The cause lies with the callee (callee is occupied (6) or not answering (3)).

The subscriber can interpret these three signals in the following way:

- Type 1: Blocked: try again. If this tone is received repeatedly, the system is overloaded (peak traffic): try again in an hour.
- Type 2: Mistake: reconsider and try again.
- Type 3: *Busy*: try again after 3 or more minutes, or (after ring-back tones). *Absent*: try again in one or more hours.

We use a single integer communal variable, with initial value zero, to signal the type of failure. The variable is named BRK(i) (from *break*).
As a result of the concurrency, there may be more than one cause for a call ending. Each cause can be specified in the single word BRK(i), for instance by associating each single bit in BRK(i) with one specific cause. The exclusive set operation BRK(i) **plusab** 2↑n would then record unambiguously that the n-th cause for call ending is applicable.
Each elementary control-flow structure in the call processing scheme can now be extended into a selection structure by adding a branch with an escape-section. The selection criterion is the value of BRK(i). As long as BRK(i) = 0, the main stream in the scheme can be followed. When BRK(i) ≠ 0, the escape branches are followed. The escapes lead (perhaps in more than one step on different levels in the nesting hierarchy) to the post-call phase, where the appropriate congestion tone can be generated.
In the next sections we take a closer look at the extensions of the digit-reception, number analysis, and toning functions, for non-ideal systems.

Digit reception and number analysis
One of the tasks of the digit receiver in non-ideal systems is the observance of time limits. Time limits may be violated on two occasions:

- for inter-digit intervals, and
- for pulse lengths (rotary dialling sets).

A violation of the pulse-length limit for rotary dialling sets is interpreted

— — — — —

(1) The subscriber may cleardown, and immediately afterwards repeat its request, thus finding himself still in the post-call phase (cause 1).

as a premature cleardown. For MFC signalling (*m*ulti-*f*requency-*c*ode) a violation of the pulse-length limit is interpreted as the reception of an invalid number. The violation of an inter-digit pause limit is interpreted as the termination of the dialling actions. If at that time the received number is still incomplete, the number is invalid, and the call is ended.
The digit receiver generally keeps count of the number of received digits. If this number reaches a certain maximum (for instance, 7 digits for local calls, 10 for national calls, 12 for inter-national calls), the dialling actions can be considered complete, and all further digits may be disregarded as non-significant. All in all, the digit reception can be completed on 4 occasions:

- the occurrence of congestion, or hardware/software faults;
- the occurrence of a premature cleardown;
- the reception of an invalid number;
- the completion of the dialling actions.

We have assumed that the number analysis starts as soon as the digit receiver has been initiated. Evidently, section NA(i) should be completed whenever one of the above situations has been detected.

The number analysis can be organized such that it is repeated each time when a new digit has arrived, until it can be completed with one of the two verdicts:

- obtainable number, or
- non-obtainable number, cleardown or congestion.

The digit receiver keeps count of the number of received valid digits in communal variable NRD(i), with initial value 0 (*n*umber of *r*eceived *d*igits). The number analysis cycle keeps count of the number of completed analysis cycles in internal variable NCC(i), with initial value x (*n*umber of *c*ompleted *c*ycles)[1]. An initiation condition for the analysis cycle then is:

$$((NRD(i) > NCC(i)) \cap (BRK(i) = 0)) .$$

If at the end of an analysis cycle it is found that:

$$((DCP(i) = TRUE) \cap (NRD(i) \leqslant NCC(i))^{2},$$

while the number received is still incomplete, the verdict should be that an unobtainable number has been received, and the call can be ended after recording this fact in BRK(i). If a received number is recognized in the number analysis as a complete and existing number, this will be recorded in communal variable ISB(i), introduced earlier, by setting it to the appropriate value. Finally, to avoid the analysis of non-significant digits, which are not recognized in the digit receiver, we can then extend the initiation condition for the analysis cycle to:

$$((NRD(i) > NCC(i)) \cap (BRK(i) = 0) \cap (ISB(i) = 0)).$$

The resulting structure of the digit reception and number analysis functions is illustrated in figure 6.15.

— — — — —

(1) By setting x to a larger value than 0, the number analysis can be started after the reception of the first x digits, instead of directly.
(2) Observe that the subscriber may decide to complete dialling before he has dialled x digits.

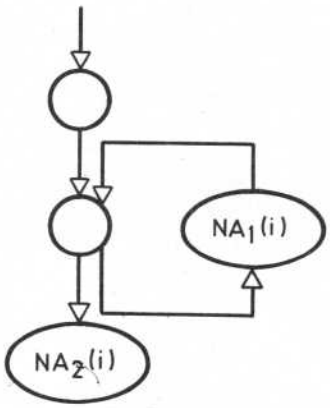

Figure 6.15.
Digit Reception and Number Analysis

The relevant proviso clauses are:

$$pric_{NA_1(i)} = (NRD(i) > NCC(i)) \cap (BRK(i) = 0) \cap (ISB(i) = 0))$$

$$pric_{NA_2(i)} = (DCP(i) = TRUE) \cap ((NRD(i) \leqslant NCC(i)) \cup (ISB(i) \neq 0) \cup (BRK(i) \neq 0)).$$

Within section $NA_1(i)$ communal variable NCC(i) is updated; within section $NA_2(i)$ communal variable BRK(i) can be updated.

Toning functions

The duration of the sending of toning signals is restricted. If the desired response of the subscriber is suspended for too long, the toning will be completed. For instance, if the maximal time for call answering has expired after approx. one minute, the ring tone circuits are released, and an escape to the post-call phase is made. If call routing actions were performed, they must be undone (the reserved path can be cleared). The time-out count for ring toning is kept in the B-side process (in the peripheral process initiated by RT(i,j)). A violation of that count is signalled to the A-side process via a special communal variable, in much the same way as call answering or call ending.

A curious situation occurs when the time limit for congestion toning is violated. The congestion toning as such can be completed, but the call process itself cannot be completed, as the persistent off-hook signal would be interpreted immediately afterwards as a new call request. A forgetful subscriber (or a short-circuited telephone line) may thus force the system to run unsuccessful call processes in rapid succession. After the completion of the congestion toning by a time-out, the subscriber is therefore placed in a so-called *park* or *stall* state. With regular intervals the line status can be checked, and only after the subscriber has finally returned his receiver on-hook will the park state be left via the execution of section CE(i). These observations lead trivially to the inclusion of an additional proviso-clause to section CE(i), namely:

$$pric_{CE(i)} = (STS(i) = 0).$$

The park state can be formalized as:

$$\left(T_{CT(i)} - I_{CE(i)} \neq 0 \quad \cap \quad STS(i) = 1\right).$$

The PARK state is one of the few states of which the duration depends only on subscriber behavior. We have encountered two similar states before: the FREE state, and the TALKING state. In the sequel we will encounter one more such state in a more extensive call processing scheme: a WAITING state (waiting queues).

6.5.2.2. *Non-local variants*

The extension of the call processing scheme to non-local types of calls is straightforward. The TWJ's can be considered as a special type of subscriber-line. One of the differences with the subscriber lines is that one must distinguish between 'line' call-answer signals and 'subscriber' call-answer signals. The line call-answer signals that a connection to another exchange has been established. It triggers digit sending actions via S. The connection in the network between A-party and B-party is however only established after the reception of a subscriber call-answer signal.
In modern telephone systems, the s.p.c. exchanges may also communicate via special fast data lines, instead of via the speech-path network. The coordination problems as such are however not different there[1].

An interesting coordination problem arises for the so-called two-way lines which carry the traffic from one exchange to another. The two-way line is the only shared object in the system of exchanges which is not only shared among the processes of one exchange, but even among the processes of *two* exchanges. In both exchanges the line can be considered as a special type of subscriber. 'Special' implies that the two-way line requires another type of signalling than the normal subscriber lines do. The two-way line can make 'call' requests, and it can be occupied as a B-party, just like the regular subscribers. Again, no more than one call process may be executed at a time per line. The principal difference between a two-way line and a subscriber line is that the initiative to occupy the line, for instance as a B-party, can be taken by two independent actors, which do not have access to shared memory modules in which they can coordinate their deeds.
We can make the following observations:

1. **Signalling requirements**

1.1. To resolve simultaneity conflicts one needs at least some basic exclusion tools on the hardware level. (See chapter 1.) In multi-processor systems we need at least (the means to implement) indivisible read and write operations on shared data.

1.2. There may be simultaneity conflicts between the two ends of a two-way line (call them E and W)[2], concerning the occupation of that line. Observation 1.1 then implies that E and W should at least be able to communicate, that is to speak and listen to each other, in unambiguous ways, in order to resolve their mutual conflicts.

1.3. Because of the nonzero signal propagation times on the line, one cannot realize an effective exclusion on signals in either direction before the line has been assigned unambiguously to one specific party: E or W.

Observations 1.2 and 1.3 imply that a mere 'semi-duplex' channel cannot be used for this type of communication. Depending on the relative speeds of the processes executed in E or W, the two parties may well decide to 'speak' and 'listen' in the same order and at the same instants, and thus be unable to communicate.

(1) There may be some new problems though, in the communication itself (especially with respect to messages which are mutilated in the transmission, and have to be repeated).
(2) *E*ast and *W*est.

1.4. We conclude that one needs, at least for signalling purposes, a full duplex channel. It is however not necessary that every two-way line in the speech-path network be implemented as a full duplex channel. The signalling tasks can be concentrated on a small(er) number of full-duplex data lines.

2. **Priorities**

2.1. Two telephone exchanges connected by two-way lines will in general not have access to a single shared memory where the status information of the two-way lines could be stored. Each exchange will have its own copy of the status bits.

2.2. The two copies of the status of a line must always be consistent, clearly. What type of signalling can guarantee this consistency? If each exchange would first test its resident copy of the status bits, occupies these when the line is found free, and then signals the other exchange in order to reach agreement over the new status of the line, a deadlock is bound to occur. The deadlock occurs if the two exchanges set their resident copies simultaneously, and then both wait for the non-resident copies to be adjusted.

2.3. A solution then is to assign a priority to exchange E on half the number of lines which connects it to W, and a priority to W on the remainder of these lines. In case of a simultaneity conflict (a possible deadlock) the side with the low priority withdraws and hunts another free line.
As the two-way lines are not rigidly assigned to specific subscribers, the subscribers cannot benefit or be impaired systematically by the high or low priority of specific lines.
The following table gives an example of such a strategy. In each state only one side can make a move, except in state 1.

State no.	E acknow. occ. by W	W acknow. occ. by E	E requests occupation	W requests occupation	Transitions to other states
1	-	-	-	-	2/3/4
2	-	-	-	x	8
3	-	-	x	-	6
4	-	-	x	x	2
5	-	x	-	-	1
6	-	x	x	-	5
7	x	-	-	-	1
8	x	-	-	x	7

6.5.2.3. *Subscriber facilities*

Subscriber facilities like 'follow-me', 'ring-back', and 'waiting-queue' can cause curious coordination problems. These problems are sometimes quite difficult to solve, notably for the non-local variants of call processing.
In this section we outline the relevant extensions of the call processing scheme. In a later section we study the coordination problems in more detail. (See section 6.6.) First we consider the facilities more generally.

There is a great variety of services that a telephone system may offer its subscribers, in addition to the regular call handling services with which we are familiar. The CEPT [1] made a list of 90 possible services. Not all of these services are 'new' though. Some 20 of them are rather basic and are available in most existing systems. The feasibility and the desirability of the other services is in many countries still being investigated. The Dutch PTT, for instance, has selected 12 services for feasibility studies. Towards the end of 1978 an

— — — — —

(1) *C*ommunauté *E*uropéen de *P*oste et de *T*elecommunication

experiment was started with 6 of these 12 services in PRX-205 exchanges in rural and large-city areas (respectively, Heereveen-Centre and Amsterdam-North). The initial findings are that the technical feasibility of many new services is limited, especially because the larger part of the telephone network still consists of electro-mechanical exchanges. (In 1978 only 20% of the Dutch exchanges were s.p.c. systems. The annual rise of this percentage is estimated at 7%. In the Netherlands, therefore, the number of s.p.c. systems will probably surpass the number of electro-mechanical systems in 1981/1982. On a world-scale, however, this is unlikely to occur within the next 15/20 years (cf. section 5.1))
The costs of implementation are relatively high. The required communication procedures between subscriber and system are considered difficult, and most importantly: there seems to be little interest on the part of the subscribers for the availability of the new facilities **(Maltha '78)**.
The 6 services selected by the Dutch PTT are not very revolutionary though. They are: waking-service, abbreviated dialling, an optional cost specification after each completed call, number repetition, a 'do-not-disturb' state, and an absence-notification. With the latter service, subscribers can ask the system to return a special message to incoming calls during a specified period of time. The special message contains the telephone-number where the absent subscriber may be reached, or where further information may be obtained.
None of these facilities will present principally different types of coordination problems from those studied in this chapter. Below we will discuss other types of subscriber services which do present new problems: follow-me (also called 'call-transfer'), ring-back (not to be confused with 'number repetition', as will be illustrated in the sequel), and waiting-queue facilities. First we will however consider still another type of service, which has gained relatively little attention until now: extensive recording services.

(1) Recording services
A promising new type of service in telephone systems is the 'recording and forwarding of spoken messages' **(De Kroes '78)**.
Subscriber A may desire to deposit a spoken message for subscriber B in the following situations:

- A calls B but B is occupied or absent;
- A does not want to disturb B, or knows that B is absent.

A straightforward method, and possibly the most transparent from the subscriber's point of view, is to introduce a special code for the recording services. The subscriber who wants to deposit a message dials the code followed by the number of the subscriber addressed. The code gives access to a recording unit in the local or private branch exchange of the caller. Instead of the ring-back tone the caller receives taped instructions on the usage of the recording facilities. Then follows the classic tone pulse which indicates the start of the actual recording. The recording is then either timed out or ended by an on-hook of the caller. In due time the system can transmit the message to its destination (as low-priority 'fill-up' traffic). When the message has arrived in the local or private branch exchange of the addressee, this subscriber can be informed of that event by an optical signal on his receiver. The addressee can request a replay of the message by dialling another special code. To avoid 'orphan messages' the sender of the message should specify:

- how many days the message must be stored, and
- what should happen when this period expires without the addressee accepting the message: (a) to notify the addressee and the sender via a printed message, or (b) to erase the message (default).

The sender can be charged in accordance with his choice.

When the addressee accepts the message, that is: when the addressee requests a replay, he should specify whether: (a) the message can be erased immediately, or (b) the message should be printed and sent to him, or (c) the message should be stored for maximally x days. As soon as the addressee has accepted the message, the storage instructions of the sender can be disregarded. The addressee becomes the owner of the message and can again be charged in accordance with his choices.

With a strategy as outlined here, the call processing structures need hardly be changed, and no new coordination problems will be introduced. One needs special recording processes which respond to the dialling of the chosen codes, etc. Another strategy, in which the sender of a message is allowed to record directly in the private branch exchange of the addressee, would not only create more coordination problems, but would also make the service less transparent for the subscribers.

(2) Follow-me

With follow-me (call transfer) facilities the subscriber can be allowed to route the calls addressed to him to another set. As the connection to the exchange serving the original subscriber will have to be established (or reserved) anyway, it is plausible to ring the subscriber first on his own set, for a brief period of time (10 seconds), and to reroute the call only if the original call is not answered[1]. The follow-me facilities are usually restricted to only local subscribers, to avoid the more complex coordination problems and the call-charging problems for non-local call-transfers. (Clearly one cannot charge a subscriber for a long-distance call if that subscriber requested a local call and is rerouted without his consent.) We consider the non-local variants below. The follow-me facilities (for non-local variants) can be included in the call processes on the call-phase level. The entire initial and terminal call phases are then embedded in an iteration loop, preceded by a section $FM_1(i,j)$ and succeeded by a section $FM_2(i,j)$, as illustrated in figure 6.16. In the return jump we find section $FM_3(i,j)$. The operation is as follows:

It is the task of $FM_1(i,j)$ to investigate whether the call is terminating or not. If it is terminating it is checked to see whether the callee has specified a follow-me reference. If so, this reference is indicated in the transient call record tcr(i). The call phase is then entered via $A_2(i,j)$, and a B-side process is initiated. The presence of the follow-me reference in tcr(i) makes function RT(i,j) observe a stricter time-limit for call answering than usual (10 seconds). If the call is not answered, the initial call phases $A_2(i,j)$ and $B_2(i,j)$ are completed via respectively, $A_3(i,j)$ and $B_3(i,j)$. It is the task of $FM_2(i,j)$ to check whether the call succeeded or not, and if it failed to check if a follow-me reference is indicated in tcr(i). If a reference is found and it is local (i.e. within the same terminating exchange), the sequence can be repeated via $FM_3(i,j)$. In the latter section the identity of addressee j is replaced by the reference.
If a non-local refenrence is found, the post-call phase is entered and the identity of the reference is signalled via CT(i) to the preceding exchange(s). If, however, the call originated in the exchange considered, section $FM_3(i,j)$ is to replace j for the reference, and the call phase is re-entered for the new (non-terminating) call. The information on references sent via CT(i) to preceding exchanges is received via TWJ (see figure 6.1). It is indicated in the tcr of the A-side process in that exchange, and the A-side process of that exchange can continue via $FM_2(i,j)$ to $FM_3(i,j)$ or to $AB_4(i)$, and so on.

— — — — —

(1) Note also that a subscriber may forget to remove a call-transfer request. The brief ringing periods will remind him of such an omission.

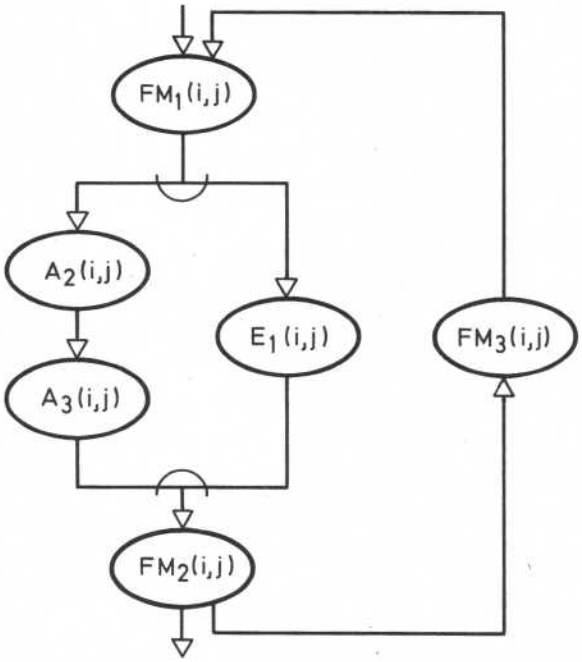

Figure 6.16.
Follow Me Extension

(3) Ring-back
When subscriber A calls subscriber B while B is occupied, A can request a ring-back service. The system will then occupy B as soon as the occupation is released, reserve a connection from B to A and occupy also A, after which both A and B receive ring-tones. Subscriber A is allowed to request or receive new calls in the meantime. (Other strategies have also been applied.)
The number of subscribers that can request ring-back service with respect to a single callee is usually restricted to just one.
The request for ring-back service can best be separated from the actual call processes. To include the service in, for instance, the terminal call phase of an A-side process would not only damage the structure of the call processes, but would also destroy the transparency of the service to the users.
The subscriber can dial a special code, as with the recording services, followed by the number of the desired B-subscriber. The caller then awaits acknowledgement (e.g. the spoken message 'you will be called') and clears down. The system then initiates special call processes, first for the callee then for the caller.

(4) Waiting queues
Incoming calls for a subscriber with waiting-queue facility are listed in a queue. The head of the queue occupies the subscriber. On call ending a new caller becomes head of queue and occupies the callee. Clearly, the manipulations of the queue must be mutually exclusive.
In still more sophisticated systems one can inform the subscriber by an audible or optical signal of the presence of waiting callers. He may then be allowed

to switch from one caller to another by dialling special codes. To implement such a facility one should redefine the call phase function $IS_b(i,j)$. Whenever calls are waiting one must now scan for coded signals. The reception of a code implies a rearrangement of the queue (round-robin). It may be wise to inform both the caller and the callee about the length of the queue.

6.6. PROBLEMS, INCONSISTENCIES, PARADOXES

In this section we will study some curious coordination problems that may occur in modern telephone systems. The problems described here have probably all been noted before. They have however not always been recognized as principal coordination problems. To develop solutions to the problems which do not destroy the structure of the call processes is by no means trivial. On the other hand, to discover the problems as such by non-automated analysis is perhaps even harder. It is therefore uncertain that no inconsistencies other than the ones discussed here, will exist.
We shall encounter two undesirable types of blocking, which we call, respectively, the *lovers' paradox* and the *circle trap*.

Mutual blocking: *The lovers' paradox*
Every subscriber to a telephone system knows the following problem: two subscribers try to reach one another simultaneously, and they both fail repeatedly until one of them stops trying.
It is clearly unsatisfactory that neither of the two routes which have been reserved in the two call attempts can be used, even though both subscribers explicitly desire to do so. Note that the principal task of a telephone system is to establish connections on request. It is therefore rather odd that this system cannot cope with situations in which two such requests merely confirm one another. We call this coordination problem the 'lovers' paradox'. Three other versions of this problem will be encountered in the sequel.
The present version is represented in the coordination graph of figure 6.17.

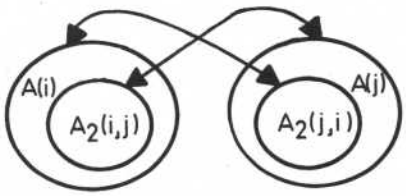

Figure 6.17.
Lovers' Paradox

Indicated are the initial call processes of two subscribers, with the lower level sections $A_2(i,j)$, respectively $A_2(j,i)$.
One immediately recognizes the hierarchical inconsistency, which would indeed lead to a deadlock if the escape-sections $E_1(i)$ and $E_1(j)$ had not been included in the call processing scheme.

Outline of a solution
A pre-requisite for the solution of this coordination problem is that its occurrence can be recognized. One method is then to store a specification of the called subscriber in the (semi-permanent) subscriber record of the caller, for the duration of the A-side call process. In a caller's A-side process in the terminating exchange it can then be checked to see whether the stored number in the called subscriber's record matches the identity of the present caller, whenever the called party is found to be occupied. If this is the case, one has recognized the block. It must then be decided which of the two connec-

tions will be used, or equivalently: which of the two parties will be the caller, and which will be the callee. The choice is arbitrary. Let us assume that the subscriber with the 'lowest' full subscriber number is given priority. The A-side call process of the other subscriber is then completed via $E_1(i,j)$, when $i > j$.
The other A-side call process is suspended until the callee has cleared down. This solution does require some additional signalling between caller and callee, as the identity of the caller must be transmitted to the exchange of the callee. This information can in turn be made available to the subscribers themselves via a display on their set on which the number of a caller can be indicated. The same display can be used as a dialling register. The subscriber can then fill the register at his own pace, before making a call request. After the request the contents of the register is then called in by the system.

The circle trap

A second type of problem, which resembles the lovers' paradox, is the *circle trap*. If three subscribers A, B, and C call one another simultaneously (A calls B, B calls C, C calls A) they will all fail. In the general case we have n subscribers ($n \geqslant 2$) trapped in the circle when each of these n subscribers attempts to call his right (left) neighbour in the circle. If n is even, half of the blocked calls could however well be established simultaneously, as they are non-conflicting. If n is odd $\frac{1}{2}(n - 1)$ could be established. This time a solution would require quite elaborate signalling strategies, if only because the number if subscribers that can included in the circle is virtually unbounded. The occurrence of a circle trap is however also much more unlikely than an occurrence of the lovers' paradox (which is in fact a circle trap with $n = 2$).

Ring-back problems

The lovers' paradox and the circle trap show clearly what dangers there may be for naïve implementations of automatic ringing facilities. For instance, the mere repetition of a dialled number when the called subscriber is found occupied could well lead into an everlasting trap. With the simpler 'number repeat' facilities (as introduced by the Dutch PTT in the field trial discussed earlier) the subscriber is therefore asked to request each repetition of the required number separately, by dialling a short code. The alternative would be to adopt a solution similar to the one described for the lovers' paradox.

Follow-me problems

Assume that subscriber A routes his calls to B, while B routes his calls to A. What happens if A calls B, and B does not answer within 10 seconds?
Clearly, the call is routed back to A, but A is occupied and thus the call is transferred to A's reference, which is B. This sequence can be repeated forever, or more likely: until subscriber A gives up. The system is meanwhile working overtime. Similarly, if A and B call each other simultaneously, they are both entangled in the call transfer trap. Neither A nor B will receive ring tones or congestion tones, while the processors 'are running in circles'.
In the general case there may be a call transfer circle of n references.
Consider also the following problem. Subscriber A routes his calls to B, and subscriber B routes his calls to C. If B calls A and finds him occupied, B will end up ringing his own reference C, which may well surprise B.

The problem of these call transfer traps then is to:

- detect referencing circles, and/or to
- limit the number of references per call. (Observe that the number of references per subscriber is assumed to be restricted to 1, already.)

Note that the only permanent call process throughout the whole referencing

procedure is the A-side call process in the originating exchange. It is then plausible to store in the transient call record of that process:

- each reference made, and/or
- the number of transfers executed.

In the first case the call process can be ended whenever a reference is made back to the caller, or whenever a reference occurs for the second time.
In the second case the call process is ended whenever the maximum number of permitted transfers is surpassed.

Waiting queue problems
The possibilities for undesirable blockings in waiting queues are evident. Suppose A, with waiting queue facility, erroneously calls himself; or consider two subscribers with waiting queues, calling each other simultaneously. In each case the subscribers enter a queue which they cannot leave again, unless they cleardown. A solution would be to prevent the enqueueing of parties with a non-empty waiting queue, *and* to prevent the enqueueing *for* parties which are themselves enlisted in a queue. Again these solutions require more elaborate signalling procedures between callers and callees.

6.7. IMPLEMENTATION ASPECTS

The abstract solution derived in this chapter can be translated into a concrete solution with dependence operations, along the lines indicated in chapter 4. Each initiation condition is then replaced by a composition of semaphore passages, such that each clause corresponds to one specific d-semaphore. Negative effects on the value of an initiation condition are represented by semaphore DOWN operations, positive effects are represented by UP operations. The manipulation of communal variables can be treated likewise. Instead of elaborating the full translation here we give an example, together with a proof of equivalence of the original and the translated description.

Consider the sequence clause for section $CI_a(i)$:

$$\left(T_{F_2(i)} - (I_{CI_a(i)} + I_{CI_b(-,i)}) = 1 \right). \tag{1}$$

The initial value of the gate-variables should satisfy the condition:

$$I_{CI_a(i)} + I_{CI_b(-,i)} = T_{F_2(i)}. \tag{2}$$

As a property of a structured section we further have, invariantly:

$$I_{CI_a(i)} + I_{CI_b(-,i)} \leq T_{F_2(i)}. \tag{3}$$

Clearly, the value of expression:

$$\left| T_{F_2(i)} - (I_{CI_a(i)} + I_{CI_b(-,i)}) \right| \tag{4}$$

determines the truth of (1). The effect of an initiation of either $CI_a(i)$ or $CI_b(-,i)$ is always that the value of (4) is increased by 1. Analogously, the effect of a termination of $F_2(i)$ is always that the value of (4) is decreased by 1. The value of (4) is stored in a D-semaphore named D(i). The effect of the initiation of $CI_a(i)$ and of $CI_b(-,i)$ can then be represented by the D-operation:

```
DOWN D(i);
```

and the effect of the termination of $F_2(i)$ can be represented by:

```
UP D(i);
```

We now have created the following 1 to 1 correspondence between the value of (4) and the value of D(i):

$$\left| T_{F_2(i)} - (I_{CI_a(i)} + I_{CI_b(-,i)}) \right| = -(D(i) - \delta).$$

Where δ is the initial value of D(i). Invariant (3) corresponds to:

$$|D(i)| \leq \delta.$$

$sic_{CI_a(i)}$ = FALSE implies:

$$I_{CI_a(i)} + I_{CI_b(-,i)} = T_{F_2(i)} \text{ or } D(i) = \delta. \tag{5}$$

$sic_{CI_a(i)}$ = TRUE implies similarly:

$$D(i) = \delta - 1. \qquad (6)$$

In the first case (5) semaphore D(i) should block in passages. In the second case (6) it should not block.

(5) then implies $D(i) < 1 \rightarrow \delta < 1$, and

(6) implies $D(i) \geqslant 1 \rightarrow \delta \geqslant 0$.

This leads to the conclusion that the translation is only equivalent to the abstract solution if the initial value of D(i): $\delta = 0$.

6.8. CONCLUDING REMARKS

We have analyzed the coordination problems in an imaginary call processing system on several levels of abstraction. The analysis was independent of possible implementations and of call processing specifics. It is especially interesting that the analysis was independent of specific coordination primitives, with which the eventual solution is to be implemented.
We have encountered essentially four types of coordination problems in this analysis:

(1) ordering problems within and between call processes;
(2) exclusion problems;
(3) escape or 'fallibility' problems: the problems related to the abortion of unsuccessful call attempts, and
(4) higher level inconsistencies, like the circle traps, and the lovers' paradoxes.

On the basis of the complete analysis one will be able to improve the call processing scheme.
The description and analysis method developed in chapter 4 leads to a clear and uniform description of the coordination requirements, which can be analyzed on varying levels of abstraction. Still, to conclude that we have proven the abstract solution 'correct' at this point would be presumptuous: the analysis was based on only a finite number of correctness criteria. Other criteria are certainly thinkable.

6.9. REFERENCES

Boute, R.T. (1978), *Logical models for computer control of telephone exchanges*, Third Int. Conf. on Software Eng. for Telecomm. Switching Systems, June 1978, Helsinki, Finland, IEE Conf. Publ. No. 164, pp. 18-24.

De Kroes, J.L. (1978), *Recording and forwarding of spoken messages*, Unpublished paper, 1978.

Maltha, R.A.D. (Dutch PTT) (1978), Personal Communication, December 8, 1978.