MASTER

Requirements model for the control system of a digital telephone exchange

Verhoof, P.H.W.

Award date:
1990

Link to publication
REQUIREMENTS MODEL FOR THE
CONTROL SYSTEM OF A
DIGITAL TELEPHONE EXCHANGE

by P.H.W. Verhoof

Master thesis report
Coaches: Prof. Ir. M.P.J. Stevens
Ir. M.J.M. van Weert

Eindhoven, February 1990

The department of Electrical Engineering of Eindhoven University of technology does not accept any responsibility regarding the contents of student project and graduation reports.
Summary

This report describes the development of the control system for a digital telephone exchange. The structured analysis method described by [Hatley] and supported by the CASE-tool ProMod is used to state a Requirements Model. Development of the Architecture Model and implementation are still to be done.

The switch array of the telephone exchange is a non-blocking TST-network and will be integrated on one chip. Quality constraints are ensured with the use of a (4,2)-redundancy system.

The resulting Requirements Model is satisfactory. Its development revealed some flaws of ProMod and some errors in the design of the exchange hardware.
## Contents

**Summary** .......................... 1

1. **Introduction** .......................... 5

2. **Context description: The telephone exchange** .......................... 6
   2.1 The TST-switch .......................... 6
       2.1.1 General structure .......................... 6
       2.1.2 Structure of the switches .......................... 9
       2.1.3 Problems caused by this structure .......................... 9
       2.1.4 Communication with the control system .......................... 11
   2.2 The (4,2)-redundancy-system .......................... 13
       2.2.1 The principle .......................... 13
       2.2.2 Communication with the control system .......................... 15
       2.2.3 The CODEC hardware .......................... 15
           2.2.3.1 The error/modus vector .......................... 16
   2.3 The concentrator and the deconcentrator .......................... 17
       2.3.1 Functional description .......................... 17
       2.3.2 Communication with the control system .......................... 17

3. **The control system** .......................... 19
   3.1 Orientation, feasibility, alternatives .......................... 19
   3.2 Design of the interfaces .......................... 19
   3.3 The need for a formal method .......................... 20
       3.3.1 Choosing the method .......................... 20

4. **The Hatley & Pirbhai method and ProMod** .......................... 22
   4.1 The Hatley & Pirbhai method .......................... 22
       4.1.1 Background .......................... 22
       4.1.2 The overall strategy .......................... 22
       4.1.3 The requirements model .......................... 23
           4.1.3.1 DFDs, CFDs and their components .......................... 24
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.3.2</td>
<td>Interpretation of flows</td>
<td>27</td>
</tr>
<tr>
<td>4.1.3.3</td>
<td>Interpretation of processes</td>
<td>27</td>
</tr>
<tr>
<td>4.1.4</td>
<td>The architecture model</td>
<td>28</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Transformation from requirements to architecture model</td>
<td>29</td>
</tr>
<tr>
<td>4.2</td>
<td>ProMod</td>
<td>30</td>
</tr>
<tr>
<td>4.3</td>
<td>Some other aspects of the method</td>
<td>32</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Coupling</td>
<td>32</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Cohesion</td>
<td>33</td>
</tr>
<tr>
<td>5</td>
<td>The requirements model of the control system</td>
<td>34</td>
</tr>
<tr>
<td>5.1</td>
<td>Orientation</td>
<td>34</td>
</tr>
<tr>
<td>5.1.1</td>
<td>A model for the entire exchange</td>
<td>34</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Call set up procedure</td>
<td>34</td>
</tr>
<tr>
<td>5.2</td>
<td>Structural description</td>
<td>35</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Functions of the control system</td>
<td>35</td>
</tr>
<tr>
<td>5.2.2</td>
<td>First level partitioning</td>
<td>36</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Lower level partitioning</td>
<td>37</td>
</tr>
<tr>
<td>5.3</td>
<td>Functional description</td>
<td>39</td>
</tr>
<tr>
<td>5.3.1</td>
<td>The terminators</td>
<td>39</td>
</tr>
<tr>
<td>5.3.1.1</td>
<td>The TST switch</td>
<td>39</td>
</tr>
<tr>
<td>5.3.1.2</td>
<td>The CODECs</td>
<td>40</td>
</tr>
<tr>
<td>5.3.1.3</td>
<td>The concentrator</td>
<td>41</td>
</tr>
<tr>
<td>5.3.1.4</td>
<td>The supervisor</td>
<td>42</td>
</tr>
<tr>
<td>5.3.2</td>
<td>The error localisation strategy</td>
<td>43</td>
</tr>
<tr>
<td>5.3.3</td>
<td>The data stores</td>
<td>44</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion and recommendations</td>
<td>47</td>
</tr>
<tr>
<td>7</td>
<td>List of references</td>
<td>48</td>
</tr>
<tr>
<td>7.1</td>
<td>Reports of the telephony project</td>
<td>48</td>
</tr>
<tr>
<td>7.2</td>
<td>Yourdon literature</td>
<td>50</td>
</tr>
<tr>
<td>7.3</td>
<td>Other references</td>
<td>52</td>
</tr>
</tbody>
</table>
Glossary of terms ................................................. 54

annex I  Version 0 of the TST-switch .......................... 59

annex II  Read and write protocols for the TST-switch ........ 60

annex III Lay-out of a CODEC ................................. 61

annex IV  Estimation of the calculation time on a PC .......... 63

annex V   Interfaces for the communication with TST-switch and
          CODEC .................................................. 66
          1. Interface between PC and TST-switch .................. 66
          2. Interface between PC and CODEC ..................... 72

annex VI  Flow diagrams of the entire system .................. 78
          1. Context flow diagrams ............................... 78
          2. First level flow diagrams .......................... 79
          3. Enhanced requirements model ......................... 80

annex VII Set up procedure in pseudo-Pascal .................. 81
1. Introduction

In 1986 a project group was set up at the Digital Systems section of the department of Electrical Engineering, here at Eindhoven University of Technology, with the task to design and build a digital telephone exchange. Since that time the group has been composed of five students on an average (trainees or graduation students). The project group started under supervision of Prof. Ir. A. Heetman and was taken over by Ir. M.J.M. van Weert because of the retirement of the professor.

When I joined this telephony group in May 1989, a first implementation of the designed hardware was expected shortly, giving rise to the need of a system, which allowed the modules to be tested. My assignment therefore was to make a control system for the exchange. Its task was in the first place to allow the designed system to be tested. In later versions it would be extended to a full-size control system.

During design of the control system the need arose for a formal method. The choice was made of using a combination of the method proposed in [Hatley] and the CASE-tool ProMod. Because time was limited, my graduation project was confined to the development of the Requirements Model, the first of two models to be developed before implementation.

Chapter 2 of this report describes the hardware of the exchange and the way it was designed to communicate with the control system. Chapter 3 accounts for the choice to use the method described in chapter 4. The resulting requirements model is discussed in chapter 5. For the convenience of the reader a glossary of terms is included at the end of the report.
2. Context description: The telephone exchange

In this chapter a description will be given of the telephone exchange of which the control system is a part.

The exchange is designed to have the following properties:

- It has a (4,2)-redundancy system (see §2.2) to ensure the quality constraints.
- The switch array will be integrated on one chip. Because fault separation is needed for the redundancy system (see §2.2), the exchange contains four of those chips, which are identical.
- The switch array has a TST-structure. This means that it is composed of three stages. The first and the third ones will be a time switch, the second one a space switch (see §2.1.1).

These result in a structure as described in figure 2.1.

The different parts of figure 2.1 have been explicitly described (as far as already designed) in the reports of other participators in the telephony project (see §7.1). For the convenience of the reader these parts will be reviewed in the following sections, especially those parts, which communicate with the control system. In some cases changes were made to the designs in earlier reports. Important changes and some considerations made by the project group are also indicated in the following sections of this chapter.

2.1 The TST-switch

2.1.1 General structure

The TST-switch is composed of three stages:
Figure 2.1 Block diagram of the telephone exchange

In the first stage 64 time switches each have one input line and one output line. On the input line the time division multiplexed (TDM) signal has 64 slots of 4 bits per frame. Each slot represents an individual channel, which carries a pulse code modulated (PCM) voice-signal. By changing its sequence, slots on the input line are switched to slots on the output line by the time switch. On its output the TDM-signal has 128 slots of 4 bits per frame.

The output line of each first-stage time switch (T1-switch) is connected to a space switch in the second stage.

In the second stage 64 space switches each have one input line and 64 output lines, all carrying a TDM-signal with 128 slots of 4 bits per frame. In a space switch every slot of the input line is switched to one of the output lines. (For each slot the output line being selected independently.) The sequence of the slots is not changed, so they leave the space switch in the same order as they came in.

Each of the output lines of a space switch (S-switch) is connected to a different time switch in the third stage.
In the third stage 64 time switches (T2-switches) each have one input line and one output line. The input line is connected to one of the output lines of all the space switches. The input line carries a TDM signal of $128 \times 4$ bits per frame. The output line carries one of $64 \times 4$ bits per frame.

The complete structure of the TST-switch is given in figure 2.2

![Diagram of TST-switch](image)

Figure 2.2 The TST-switch

According to [Ronayne p.74] this array is non-blocking, because the number of internal slots (to or from the S-switches) is not smaller than $2n-1$, where $n = $ the number of external slots.
2.1.2 Structure of the switches

Time switches have two memories: a speech or data memory and a connection memory. The data on the incoming line is written sequentially into the data memory, so each incoming slot number has its own memory cell. The order in which the data memory is read is stored in the connection memory. This connection memory is read sequentially; Each outgoing slot number has its own memory cell. These cells contain the address of the data memory, which corresponds to an incoming slot number.

Due to timing delays slot numbers are not identical to addresses: The address of the data memory corresponds to incoming slot number \((\text{address}-1) \mod 64\). The address of the connection memory corresponds to outgoing slot number \((\text{address}+1) \mod 128\).

Space switches are composed of a connection memory and a demultiplexer. No data memory is needed because the sequence of the slots is not changed. Each slot number has a memory cell, which contains the number of the outgoing line, where upon that slot is sent out. Like in the time switch, the address of the connection memory corresponds to outgoing slot number \((\text{address}+1) \mod 128\). However, it corresponds to incoming slot number \(\text{address}\) due to the internal delay of the space switch.

In figure 2.3 the internal structure of the switches is shown. It shows a connection from \{incoming line 0, incoming slot 10\} to \{outgoing line 63, outgoing slot 41\}. For this connection time slot 20 is used from T1- to S-switch. Due to the delay the slot number used for the connection from S- to T2-switch is one higher: 21. The path of the data of the connection is marked by `##`.

2.1.3 Problems caused by this structure

The structure as described in section 2.1.2 causes a problem: If two different space switches have the same data on the same address of their connection memory, a conversation interference can occur: The two space switches will be writing on the
same line at the same time. Normally this situation will not occur, because such a connection would be useless, but in practice this situation will be inevitable.

There are three possible ways to get this situation:

1. During the start up process, before the initialization by the control, the contents of the memories have a random value.

2. When a memory error occurs, the contents of a memory cell will be different from the value that was written into it.

3. During switching, when setting up a connection, the S-switch must pass through the situation that two switches write on the same line at the same time. This can be seen as follows:

For a given slot number all space switches must be connected to a different outgoing line, in order to avoid conversation interference. Because there are as many space switches as there are outgoing lines, all outgoing lines
will be connected to a space switch. Because the memories of two different space switches cannot be written at the same time (see §2.1.4), a situation as described above cannot be avoided.

Some suggestions have been done by the project group to prevent a short circuit being caused by conversation interference:

- Implementing an S_ENABLE line, which can block the output of the S-switches. This will solve the problem in situation 1, but cannot prevent situation 2 from happening. If this method is used during switching (situation 3) all connections will be blocked during some microseconds, an unwanted side-effect.

- Using logical circuits to connect the space switches to the same line. This solves the problem in all three situations, but causes a different problem: Each S-switch has 64 output lines. Each of those lines must be connected to a different logical circuit. As shown in figure 2.4 an array is needed which is 64 x 64 lines large and 64 x 64 lines wide. If each line is 2\mu m wide and between each line a space of 2\mu m is left, the surface needed for this array is 1.6 cm x 1.6 cm. This does not fit on one chip.

For the time being, the problem is handled by decreasing the number of external lines. In Version 0, the TST-switch will only have four incoming and outgoing lines and consequently four T1-, S-, and T2-switches. The array of lines will only take 0.64 mm x 0.64 mm.

The structure for Version 0 is shown in annex I. For later versions the problem is being studied by John Muijselaar.

**2.1.4 Communication with the control system**

The control system can read and write data to and from the connection memories of the switches. By writing, it can make a connection, by reading, it can check the contents of the connection memory.
Figure 2.4 The line problem
Because all switches are integrated on one chip, first one switch has to be selected, before the read or write operation can take place. This selection is done by making the SEL-line of the desired switch high.

The protocol for the read and write operation is described in [Van Lier pp.21-22], the timing diagrams are copied in annex II.

2.2 The (4,2)-redundancy-system

2.2.1 The principle

The (4,2)-concept for fault tolerant computing was developed by Th. Krol for Philips, who gives lectures on this subject at Eindhoven University of Technology. According to this concept an 8-bit data-word is coded into four data-words of 4 bits. The system (in our project the TST-switch) consists of four independent slices. Each one of them performs identically the same function as the other ones. Each of the four 4-bit words will be presented to the input of a different slice. Every slice generates its own 4-bit data-word on its output. The four data-words coming out of the system are translated by a decoder, into one 8-bit data-word. Thanks to the fact that the data on the input of the decoder contains 50% of redundant information, the decoder is capable of correcting the following errors:

- Complete failure of one of the slices
- Two random bit-faults in two different slices
- If the decoder knows about the failure of one of the slices, it can correct a random bit-error in one of the other slices.

The TST-switch with (4,2)-redundancy coding is depicted in figure 2.5. The four CODECs (decoders/coders) first decode the data coming out of the TST-switch and then encode it for further transport. The reason why these CODECs were included in the system was the fact that the distance between the switch array and the last decoder may be some kilometres. The thought was to protect this traject by means of the redundancy coding.
Figure 2.5 The TST-switch and the redundancy system

This solution, however, demands considerably more cabling and does not really improve the fault-recovery characteristics of the system. For instance, it hardly protects the system against the breaking of a cable. In case a cable should break, it is most probable that all four of them will break, as it is too expensive to give each cable its own trajectory through the city. When the CODECs cannot prevent errors, they must be eliminated, because they are themselves a source of errors (like all devices are).

The decision not to implement the CODECs was taken after a discussion, which took place some months after the modelling of the control system was started. For this reason this change is not taken into account for the model. In fact no definitive
decision has been taken, due to the fact that no-one can recall the reason why these CODECs should be implemented.

2.2.2 Communication with the control system

The CODECs are the only parts of the redundancy system which communicate with the control system. For this reason a brief look inside them will be given here. More explicit information can be found in [Aggenbach]. Very recently another report has been written on this subject by Ralph van Duin. He is a trainee, who finished his work in June 1989, but has written his definitive report only just now, due to his training abroad, which started in July.

Each CODEC has four incoming lines and one outgoing line. Apart from those, it has some communication lines to and from the control system, which will be discussed later.

The incoming lines carry a TDM-signal of 64 x 4 bits per frame, each one originating from a different slice of the TST-switch, having all four the same output line number. From the input signals the CODEC reconstructs for each slot the original 8-bit word which entered the coder. This is subsequently re-coded for further transport. Hence the output line of the CODEC carries a 64 x 4 bits per frame TDM-signal.

For each slot the CODEC has a register, in which it stores the so-called 'error/modus vector' of that slot. This vector indicates the errors that have been found in the slot. These registers can be read by the control system, so that it can locate defects in the TST-switch. Thanks to the fact that there are four CODECs with the same input lines (see figure 2.3) the control system can detect defects in the CODECs. Therefore its registers must have the same contents. If not so, an error has occurred in (one of) the CODECs.

2.2.3 The CODEC hardware

The hardware of the CODEC is described in [Aggenbach]. For the design of the control system only its communication with the outside world and the interpretation
of the error/modus vector are important. In annex III a copy is given of the lay-out of a CODEC as can be found in [Aggenbach pp.17-18].

2.2.3.1 The error/modus vector

An error/modus vector contains one out of 31 possible values. 12 of them may be written into it by the control system. The control system does this writing either to set the CODEC for one slot in the 'idle' mode, or to reset the vector when a connection has just been set up. These 12 values are:

- 1 non-correcting mode ('idle')
- 1 random mode ('no slices defect')
- 4 erase modes ('one slice defect')
- 6 single modes ('two slices defect')

When the vector reaches a single mode, no errors can be detected and so the CODEC cannot change the vector any more.

During operation the value of an error/modus vector is changed whenever the CODEC detects an error. Before a connection is broken down the error/modus vector is read by the control system, which makes it in the long run possible for the control system to localise errors.

The CODEC decides that a slice is defect when two single bit errors occur in two consecutive frames. If more than two not-correctable errors occur during a connection, the CODEC will generate an interrupt. The control system must check the contents of the error/modus vectors of the CODEC to determine which slot was responsible for the interrupt.
2.3 The concentrator and the deconcentrator

2.3.1 Functional description

Task of the concentrator is to multiplex a number of digital subscriber input lines on one TDM-PCM line, which is the input line of the coder. The output line of a concentrator thus carries 64 x 8 bits per frame. (See also figure 2.5.) As not all subscriber lines are occupied at the same time, the number lines entering a concentrator may be greater than 64. In [Van Engelen] a maximum concentration factor of 11 is determined, which comes down to 704 subscriber lines on one concentrator. This is a maximum and in practice the number will probably be 512.

The deconcentrator performs the task of distributing the 64 channels on the output of the decoder to the digital subscriber output lines.

The concentrator and the deconcentrator will be placed in a remote part of the exchange, close to the subscribers. This remote part, the so-called Digital Subscriber Stage, is a local exchange, which can handle local calls without interference of the central control. For the reasons indicated in section 2.2 the coder and decoder used to be part of this remote part, but they will probably be placed in the central part.

In figure 2.6 the digital subscriber stage is depicted. It contains, except for the parts already named, a local processor and for each subscriber a subscriber line interface card (SLIC). The local processor serves to handle the local calls. The SLIC is an interface between an (analog) telephone set and the digital subscriber lines (input and output).

2.3.2 Communication with the control system

About the communication between the digital subscriber stage and the control system much is still uncertain. It is not yet decided whether channel associated signalling or common channel signalling (CSS) will be used. [Van Engelen] assumes channel associated signalling, but cannot give convincing arguments for that choice.
Figure 2.6 The digital subscriber stage

[Van Engelen pp. 25-26] defines two control signals for the concentrator: The control system can give the command to make a connection from a subscriber line to a time slot and it can read which connections are currently set in the concentrator. As will follow from the requirements model, which will be treated in chapter 5, these commands are not sufficient: The concentrator must pass the set-up and break-down requests through from the SLIC to the control. To set up a connection, for instance, the control needs to know the subscriber number of the person who is called.

Because the deconcentrator has not yet been designed, its control signals are not yet determined.
3. The control system

3.1 Orientation, feasibility, alternatives

As described in chapter 1, a control system must be made to fit in the system, described in chapter 2. This control system could be implemented in software on a personal computer (PC). To make sure that a PC is fast enough an estimation of the calculation time is made (see annex IV). The conclusion is, that it must be possible to implement the control system on a personal computer, however calculation time must not be used too lavishly. An alternative would be to use a special purpose processor, which would demand a design of both software and hardware.

The question on what operating system the control system will be implemented is not yet answered. An option is to write it with LEX (Local EXecutive), a real-time executive, developed at Eindhoven University of Technology (see [Duijkers], [Van Etten]). As far as could be judged in this early stage, LEX seems to be suited for this system.

3.2 Design of the interfaces

To make software control of the telephone exchange possible, interfaces with the environment (see figure 2.1) are needed. In annex V these interfaces for the TST-switch and the CODEC are shown. Procedures are included which describe the communication from the software point of view.

These interfaces are not copied from [Hense], but are my own design. Unfortunately the design of M. Hense was overlooked, because his report was not yet available, and an own design was made. After discovering his report, both solutions were compared. My design was found to be the better one, because of its less complicated structure.
As described in section 2.3, the communication with the concentrator and the decon­centrator is not yet defined, so interfaces can not yet be designed.

3.3 The need for a formal method

Designing a system as this control is not an easy job. For that reason it is important to have all thoughts on paper. In this way thinking errors can be corrected and vague ideas can be developed into concepts.

The need for a clear overview makes the notes develop from a little list of terms to a figure with arrows, blocks and commands. Flow charts are introduced very soon, but they cannot give the overview, because they are not made for real-time systems.

At this moment the need arises for a method which does give the overview. This method must both be standardized and formal. Standardized, because other people have to be able to co-operate and check or continue the work. Formal, because that is the only way to cope with vagueness. And vagueness is an important source of errors.

3.3.1 Choosing the method

Because the CASE-tool ProMod is present at the section of Digital Systems, the first step in choosing a method was easy: As the ProMod documentation [ProMod] refers to the method described in [Hatley], this method was studied first. The outcome was positive: the method appeared to be suited.

As alternatives SDL and the method described in [Ward] were considered. SDL seemed not to be very suited and had as major disadvantage that there was no software support present for this method. [Ward] was not chosen, because it differed too much from what ProMod expects. Implementing the [Ward]-method on ProMod would be working with two standards, which violates both standardization and formalism.
So, at first, the choice was made for [Hatley]. During work with ProMod however, the tools did not appear completely compatible with [Hatley]. This problem could be solved by making the description acceptable to ProMod, without violating the rules of [Hatley]. Unfortunately this came at the expense of some decreased legibility. The alternatives of working with two standards or abandoning ProMod were considered, but eventually rejected.
4. The Hatley & Pirbhai method and ProMod

In this chapter the principles of the method described in [Hatley] will be explained. Of course it is almost impossible to capture all of its 412 pages in this report. For this reason the goal of this chapter is restricted to give the reader a notion of what the method comprises.

In section 4.2 the software-tool ProMod will be discussed, which is based on the method of Hatley & Pirbhai. In some points however, ProMod does not exactly cover the method. These differences will be indicated in section 4.2 and its effects will be discussed.

In section 4.3 some other relevant subjects concerning this method will be discussed.

4.1 The Hatley & Pirbhai method

4.1.1 Background

The method by D.J. Hatley and I.A. Pirbhai, presented in [Hatley], is an adaption of earlier methods for structured analysis and structured design, which can be found in the so-called Yourdon-literature. E. Yourdon is the writer of one of the first books on this subject ([Yourdon]) and publisher of a series of books which presented extensions, adaptions and improvements on his method. An overview of the most important works in Yourdon-literature is presented in a special section of the list of references (see §7.2).

4.1.2 The overall strategy

According to the method a requirements model and an architecture model are developed before implementation of the system.
The requirements model must describe what the system must do, and must leave as much space as possible to the designer's choice how to implement this. The architecture model must describe how the design will be structured. Development of these two models should not be a sequential, but rather an iterative process. Because the two models are interrelated, they must be developed in parallel.

The method contemplates traceability of functions and units, so that changes in the system become more manageable and critical parts can more easily be prototyped or otherwise be evaluated before the implementation of the complete system.

In the following two sections both models will be described.

4.1.3 The requirements model

The whole system requirements model is a cohesive statement about the system's data flow (shown in the data flow diagrams), data processing (shown in the process specifications), control flow (shown in the control flow diagrams), and control processing (shown in the control specifications). These components interconnect with each other as shown in figure 4.1. The interconnections are formed either by data flows (shown as drawn lines) or by control flows (shown as dashed lines).

The components illustrated are described below:

- **Data flow diagrams** (DFDs) decompose the system and its functions into a hierarchical structure of processes. They show the transfer of data from one process to an other.
- Using the same decomposition as the DFDs, the control flow diagrams (CFDs) show the transfer of control from one process to an other.
- **Process specifications** (PSPECs) specify in concise terms each detail of the system's functional requirements. Control flows generated inside PSPECs through tests on data flows are called "data conditions".
- **Control specifications** (CSPECs) specify the processing of control through tests on control flows.
Figure 4.1 The structure of the requirements model

- The requirements dictionary contains a listing of all the data and control flows on the DFDs and CFDs together with their definitions.
- The response timing specification defines the limits on response time allowed between events at the system input and the resulting events at the system output.

DFDs and CFDs are the most complicated components of the model. They will be explained in the following section.

4.1.3.1 DFDs, CFDs and their components

The system requirements are described in an hierarchical structure of processes. On the top level only one process exists, namely the entire system itself. Going down in the structure each (sub)process is decomposed into subprocesses, until the lowest level is reached. Processes on this level are called "primitive processes". Each primitive processes is described by a PSPEC, each non-primitive process by a flow
A flow diagram describes the decomposition of the process it represents, and the structure of data and control flows between its subprocesses. It may be drawn as one diagram, but can also be drawn as two separate diagrams; the DFD showing the data flows and the CFD showing the control flows between the subprocesses.

Figure 4.2 shows the elements of a flow diagram. They are described and listed below.

**Process (circle):** Processes are part of the hierarchical structure as described above. They represent an action, which produces its outgoing flows from its incoming
flows. Non-primitive processes are described by a flow diagram one level lower in the hierarchy, primitive processes by a PSPEC.

Data Flow (directed arc): Data flows may represent anything from a single item of information to a group of any number of items. Data flows may be split or merged, as shown in figure 4.2.

Control Flow (directed dashed arc): What is stated for data flows is valid for control flows too, except that they carry control information. The difference between data and control is difficult to describe and depends on the interpretation of the model maker.

Store (two horizontal lines): Stores may either be data stores or control stores. Their function is to sustain the information on its incoming flows even after its source stopped generating it.

CFD to CSPEC Interface (bar): Control flows starting or ending on a bar indicate outputs or inputs of the associated CSPEC. Different bars within the same flow diagram are all associated with the same CSPEC. For this reason bars do not need labels.

Input and Output Flow ("loose flow"): Flows that seem to start or end "nowhere" refer to a connection to the parent process. This is the process one level higher in the hierarchy. The in- and outgoing flows of a process should be the same both in parent and in child diagram.

Terminator (square): Terminators may only appear in a context diagram (Data Context Diagram, or Control Context Diagram). They represent connections to the outside world, i.e. to objects outside of the system.
4.1.3.2 Interpretation of flows

Flows can either have destructive or non-destructive read-out. Flows with destructive read-out will lose their information once they have been read by the receiving process. The flow can only be reread after the sending process has generated new information. Flows with non-destructive read-out, on the other hand, can be read and reread any number of times.

Unlike other methods, Hatley & Pirbhai do not distinguish between the two kinds of flows in flow diagrams. The only place in the model where they are treated differently, is in their originating PSPEC or CSPEC: A flow with destructive read-out is generated by the statement "Issue <flow name> = ...", flows with non-destructive read-out by the statement "Set <flow name> = ...". Flows which originate from stores have always non-destructive read-outs.

4.1.3.3 Interpretation of processes

A set of DFDs and CFDs is a model of the requirements of a system, not a representation of the system's implementation. The model is highly idealized in that it is assumed to be data triggered and infinitely fast: Whenever data sufficient for a given process to perform its task appears at its inputs, then it will perform that task, and do so instantaneously. This means that in order to know at what moment a process is activated, it is important to know if its incoming data flows have destructive read-out. Flows with non-destructive read-out are always present and cannot be a data-condition.

Some processes are not triggered by data, but by control, which means that its activation is described in the CSPEC of the parent-process. Processes which are not mentioned in this CSPEC are considered to be data triggered.
4.1.4 The architecture model

Only a slight notion will be given here of the architecture model, because it is beyond the subject discussed in this report.

Purpose of the architecture model is to
- show the physical entities that make up the system
- define the information flow between these physical entities
- specify the channels on which this information flows

As in the requirements model these purposes are fulfilled using diagrams, supported by textual specifications and a dictionary. These components are shown in figure 4.3 and described below.

![Figure 4.3 The structure of the architecture model](image)

The architecture flow diagram (AFD) shows the physical configuration of the system, in terms of its architecture modules and all the information flow (data and control).
The architecture interconnect diagram (AID) shows the physical interconnect of the system components, in terms of the channels by which information flows between the architecture modules.

The architecture module specification (AMS) captures the allocation of requirements from the requirements model to specific architecture modules.

The architecture interconnect specification (AIS) captures the characteristics of the channels by which information flows between the modules.

The architecture dictionary is an enhancement of the requirements dictionary. It captures the allocation of all the data and control flows to architecture modules and the channels on which they flow.

Like in the requirements model the system architecture model consists of a layered set of AFDs and AIDs and their associated AIMs and AISs.

As admitted by the authors of [Hatley] this model is not suited for mapping software requirements into software modules. Their advice is to use structure charts as described in a number books of Yourdon-literature (see §4.3). For the hw/sw-interfaces and the hardware requirements itself, however, being essential parts of a real-time system, the development of the architecture model should not be overlooked.

### 4.1.5 Transformation from requirements to architecture model

Though a detailed description of the transformation process is beyond the scope of this report, some words will be spent to it, as it is a part of the method.

The transformation from requirements to architecture model is made through an intermediate stage called the "enhanced requirements model". This model is formed by inserting a buffer between the essential requirements model core and the environment as shown in figure 4.4. This adds some nonfunctional requirements to the
model. The enhanced requirements model has the same general structure as the architecture model (the so-called "architecture template"), and therefore can be translated to the architecture model by assigning processes to modules.

Figure 4.4 Enhancing the requirements model

4.2 ProMod

ProMod is a CASE (Computer Aided Software Engineering) tool which supports the Hatley & Pirbhai method. Flow diagrams can be drawn, control and process specifications and the data dictionary can be typed in. ProMod checks the consistency of the model given by the user and produces a report. For the development of the
requirements model, as described in chapter 5, I made use of version 1.6_2 of ProMod on an MS-Dos machine. At the end of the modelling process version 1.7a became available and therefore the existing model was translated to the newer version.

Major advantage of ProMod is not the drawing and text editing facility, but the flexibility to make changes to a model already (partly) typed in. An important option of the tool is the possibility to write parts of the model to disk. Each element of the model can be saved separately, or a group of elements can be saved. This gives the possibility to create some alternatives, from which a choice can be made later. These alternatives need not all be typed in from the start, but can be made out of little changes to an existing element.

Among the disadvantages of the tool is the fact that it is slow. Furthermore, I strongly recommend to make use of an external professional text editor (e.g. Word Perfect). DOS-text files generated by this text editor can then be read in by ProMod. Another disadvantage is that it is not entirely compatible to the Hatley & Pirbhai method. Some of the most important differences are listed below:

- ProMod demands a specification of all process, not only for the primitive ones. They are called "mini specs" (MSPs).

- When merging and splitting flows in ProMod, one name must be given to the entire flow. For instance a flow which splits flow A+B into flow A + flow B cannot exist. The problem can be resolved by splitting flow A+B into two flows A+B. Of each of the resulting flows a part will not be used by the receiving process. The fact that a part is not used cannot be seen from the diagrams, but it has to be looked up in the MSPs.

- Flows to or from the parent process may not directly connect to a store in ProMod. This problem can be resolved by showing the store in the parent process, aswel as in the child process, where it is used.
Another feature of ProMod is that it automatically makes the translation from requirements to architecture model. For ProMod this means designing a structure chart, but as indicated in section 4.1.4, [Hatley] considers this as only a part of the architecture model, namely the software part. Structure charts do not take interfaces with hardware or hardware requirements into account.

### 4.3 Some other aspects of the method

In this section two important aspects of the method will be discussed, that did not get attention in [ProMod] or in [Hatley]: The concepts 'coupling' and 'cohesion' give insight in how to make the decomposition of processes or modules.

#### 4.3.1 Coupling

In a good designed system the modules have a high degree of independence. In other words: They have low coupling.

[Page-Jones] gives three reasons why low coupling is important:

- The fewer connections there are between modules, the less chance there is for a bug in one module to appear as a symptom in another.
- Changes in one module must have minimum risk of having to change another module, and each user change must affect as few modules as possible.
- While maintaining a module, there must be no need to worry about the internal details of any other module. The system must be as simple to understand as possible.

Though modules are parts of the architecture model, low coupling is important between the processes of the requirements model too. Not only are the above reasons valid for both processes and modules; by transforming the requirements model into the architecture model low coupling between processes will result in more independent modules.
[Page-Jones, pp.103-104] describes five different types of coupling, from good to bad and thus indicates how to get good coupling.

### 4.3.2 Cohesion

Another way to measure the quality of the design of a system is the so-called cohesion of its modules: Modules must be composed of strongly associated items. This makes it possible to see the modules as a "black box" and decreases the coupling between modules.

[Page-Jones pp.119-132] describes seven different types of cohesion. In figure 4.5 a decision tree is shown which helps determining the type of cohesion of a module.

![Decision tree to determine the level of cohesion](image)

Figure 4.5 Decision tree to determine the level of cohesion

As for coupling, cohesion is not only important for modules, but also for the processes in the requirements model.
# 5 The requirements model of the control system

## 5.1 Orientation

### 5.1.1 A model for the entire exchange

In order to get a good understanding of the communication of the control system with its environment, in a first step the entire exchange was considered as the system to be modelled.

In annex VI.1 and annex VI.2 respectively the context diagrams and the first level flow diagrams are shown for the entire system. Further decomposition is not done, neither are other parts of the requirements model shown. The only goal for these diagrams is to show the outside world of the control system, not to re-design the entire system.

In this model the exchange system is shown in an operative configuration, which is an extension of 'Version 0'. This last one is only a test configuration, which does not, for instance, include a deconcentrator and does not communicate with a remote exchange.

The redundancy system is not a requirement for the exchange, but a design decision and is therefore not shown in the flow diagrams. It is, however, a part of the environment of the control system. For this reason the enhanced requirements model (see §4.1.5) is shown in annex VI.3. It includes coding, decoding, redundancy checking and interfacing for communication with the outside world of the exchange.

### 5.1.2 Call set up procedure

In figure 5.1 the connections of a call are depicted, to show where which information is needed in the network. The call is assumed not to be local to a concentrator-deconcentrator stage nor to one exchange to make it generally valid. In annex VII a
procedure is given in pseudo-Pascal describing the set up of the connections for a call. Each procedure-call describes the communication between two parts of the system.

![Diagram of connections for a call](image)

Figure 5.1 Connections for a call

### 5.2 Structural description

In appendix I the rapport produced by ProMod is printed. In this section the partitioning of the processes into subprocesses will be discussed. Section 5.3 will describe the different elements of the model.

#### 5.2.1 Functions of the control system

The control system is to perform the following functions:
- Call processing
  - Functions as searching, selecting, marking 'occupied' or 'free' of paths through the switching network.

- Management and maintenance
  - Management of hardware and software of the exchange; For instance changing of data relating to the subscribers, changing of translation tables, put into use of new subscribers and interlocal lines or changing the configuration of the switching network.
  - Supervision on the functioning of the exchange and doing measurements with regard to the traffic.
  - Detection and localisation of errors, doing tests either periodically or on request.

These functions were stated in [Aggenbach]. However, some of the functions mentioned in [Aggenbach] are not repeated here, because they belong to local processors, or will not yet be implemented in Version 0.

### 5.2.2 First level partitioning

First level partitioning of the control system has as a result DFD 1 and CFD 1, shown in [Verhoof p.10]: Starting from the requested functions, described in section 5.2.1, the processes are split up. During the split up process factors as coupling, cohesion, the need for a clear picture, and the need to show only the important things were taken into account. Because call processing was judged important, in this first level it is already split up into three subprocesses. Management and maintenance was considered less important, so it has only two processes in this first level, though three subfunctions were stated in section 5.2.1.

The error detection function is hidden in the processes 'break_down_link' and 'change_link', because it has sequential cohesion with these processes. At first it seems strange to have this function twice in the model. A further consideration, however, shows that having only one error detection process is a result of 'flow chart
thinking' and causes many problems; The goal of the requirements model is to state
the system requirements as clear as possible. Making efficient use of the resources is
an architecture problem, so there is no reason not to have two identical processes in
the requirements model.
Because the error detection function needs to send information to its activating
process, the situation of having only one of such function is undesirable: It needs an
extra information flow indicating its activator. This solution, however, introduces
some design decisions, which should not be a part of a requirements model.

In the same way the process 'change_link' might be seen as superfluous, because it
does nothing more than performing 'set_up_link' followed by 'break_down_link'. This
solution too, introduces design decisions, (because of the needed sequential
performance) and makes the model unnecessarily difficult. For 'change_link' a
separate process is introduced, which makes use of 'set_up_link', but performs its own
breaking down.

In some cases it is harmless to have more activators of one process: When a process
needs not know its activator, it makes the model only clearer if it shows the process
only in one place. For instance the process 'make_changes', which is a subprocess of
'set_up_link': It can be activated by either 'do_single_routing', 'do_double_routing' or
by 'find_intermediate_slot'. Though having only one process 'make_changes' might be
seen as an architecture decision, it is hardly thinkable that architecture would indeed
come up with three such processes. And therefore it is shown as one process,
stressing the fact that only one such a process is activated at a time.

5.2.3 Lower level partitioning

The splitting of lower level processes is in most cases very straight-forward and needs
not be discussed. For some of them, however, a short comment is given here.

Descending in the tree of flow diagrams (see [Verhoof]) it can be seen that the error
detection function appears in two processes: 'break_down_link' and 'change_link'.

The error detection function in 'break_down_link' is similar, but not identical to the one in 'change_link'.

The process 'set_up_link' is control coupled with its activating processes: In the implementation probably a 'what-to-do-flag' is needed to indicate whether a DOUBLE_NUMBERED_CONNECTION, an ADDRESSED_CONNECTION, a CHECKED_ADDRESSED_CONNECTION or a ROUTED_CONNECTION needs to be set up. Though this is a rather bad form of coupling, it can hardly be avoided here in the requirements model. The alternative would be to place those subprocesses which translate the desired connection to a ROUTED_CONNECTION outside 'set_up_link'. This would mean that the subprocesses with activators from outside the system would have to be placed in the first level diagram, for which they are far too detailed.

For this reason the situation is left as it is. The coupling problem may later be resolved during development of the architecture model. The option of joining the three translating subprocesses into one, called 'do_routing', was abandoned, because it takes the coupling problem too far down in the hierarchy.

Another problem exists for the processes performing routing: A kind of mutual exclusion is needed to ensure that a switch cannot be changed from 'free' to 'occupied' by another process during routing. This must not occur, because in that case the sequence below may cause two different routing processes to employing the same switch in a different connection:

1. Routing process A reads data about switch X. (Switch I is marked 'free'.)
2. Routing process B reads data about switch X. (Switch I is still marked 'free'.)
3. Routing process A concludes that switch X can be used for its connection. It marks switch X 'occupied'.
4. Routing process B also concludes that switch X can be used for its connection. (According to the data process B has, switch X is still free.) It marks switch X 'occupied'.

5. Both process A and B include switch X in their connection.

This can only be resolved by introducing an extensive control structure or by making big changes in the structure of the model. Because the processes in the requirements model are assumed to be infinitely fast, the problem is however seen as not-existing for the requirements model. A remark is made in the mini-specs of the processes concerned, relating to the need for mutual exclusion in the architecture model.

5.3 Functional description

In this section the elements of the developed requirements model will be described, as far as the ProMod report (see [Verhoof]) needs further explication.

5.3.1 The terminators

ProMod does not leave much space to the description of the outside world, though an individual description of each terminator does add to the clearness of the model.

At first sight a description of the terminators is not needed, because the environment has already been described in chapter 2 and in earlier reports of the project. However, the requirements model does not include the interfaces with the outside world, which are only included in the enhanced requirements model (see §4.1.5). This means that the communication with the terminators will be somewhat different from the earlier environment descriptions.

Below all four terminators of the model will be treated individually.

5.3.1.1 The TST_switch

The terminator 'TST_switch' gives the 'control_exchange' process access to the CONNECTION_STATUS, containing information about how the switches are set. This data is regarded as being always present, though in the actual situation (see annex II) a read command must be performed before the data can be offered. This
terminator thus is regarded in the same way as a data store, which does not needs to be addressed before it can be read, in this model. Though the actual situation is in essence the same, for the CODECs terminator a different construction is used to read data from it (see next section). The reason for the unnatural (but not faulty) construction to read from the TST_switch and having another, more natural construction, to read from the CODECs is a simple one: At first all read constructions in the requirements model were like the one given here, to read from the TST_switch. Reading from this terminator was simply overlooked when a better construction was introduced.

Writing to the TST_switch terminator can also be seen as writing to a store: the switches can be set by writing NEW_CONNECTION to the TST_switch. Writing is always performed to all four (see §2.2.1) slices at the same time. NEW_CONNECTION contains the information for all four switches that need to be changed to make the connection, whereas CONNECTION_STATUS gives only the data from one switch in one slice.

In section 2.1.2 it was already remarked that the physical data inside the connection memory of a switch is different from the slot number it represents. The transformation from and to physical data is done inside the TST_switch terminator. In the enhanced requirements model this calculation may be done in the input and output processing interfaces.

5.3.1.2 The CODECs

The terminator 'CODECs' offers on demand of 'control_exchange' four STATUS_WORDS. The demand consists of offering the CODEC_ADDRESS (indicating a CODEC and a slot) to the terminator. The response consists of the STATUS_

* A T1-switch, a T2-switch and, in order to avoid conversation interference (see §2.1.3), TWO S-switches.
WORDS of all four slices, because they are always needed at the same time. The STATUS_WORDS are in fact the same as the error/modus vector described in section 2.2.3.1. This vector is changed by writing NEW_CODEC_STATUS to the terminator, which will change the indicated vector in all four slices.

Another data flow from CODECs to 'control_exchange' is CONNECTION_TO_BE_CHANGED. It contains the address (physical CODEC number and slot number) of a CODEC which has generated an interrupt. The interface between CODEC and control system must trace the origin of the interrupts, because the CODEC itself does not give an address when generating an interrupt signal (see [Aggenbach p.22]).

5.3.1.3 The concentrator

As indicated in section 2.3.2 the communication between the control system and the concentrator/deconcentrator has not been defined very well. The requirements model therefore may be incomplete in this description. Only the signals needed for call processing are shown. Because a deconcentrator will not be a part of 'Version 0' it is not yet included in the context.

The concentrator terminator can give request to 'control_exchange' to set up a connection. In Version 0 this will always be a 'double_connection', being a two-way conversation between two subscribers. The concentrator determines the incoming and outgoing slots for the caller and passes these slot number together with the telephone number of the receiver through to 'control_exchange'. If it is impossible to make this connection, for whatever reason, control_exchange issues CONCENTRATOR_NOT_POSSIBLE. This is a sign for the concentrator to give the occupation tone to the caller.

This is however one probable solution. Another way is to let the concentrator send both the telephone number of the caller aswell as the number of the receiver to the control system. When introducing the account system into the control, it will be necessary to know the number of the caller, so it might aswell be given at this
moment. The control system then determines the slots to be used for the conversation and passes them to the concentrator.

When a connection must be broken down, the concentrator sends both the input as the output address of the connection to 'control_exchange'.

The name 'concentrator' for this terminator is maybe not such a lucky one. In later versions this may be changed into a combination of concentrators and deconcentrators, possibly supplemented by other exchanges in the network. As shown in section 2.3.1 a concentrator and a deconcentrator together form a local exchange, which represent the same function as all exchanges in the network.

5.3.1.4 The supervisor

The terminator 'supervisor' is a man-operated machine, which includes a system for man-machine communication. This system presents the STATUS_INFORMATION in a readable format to the person, and translates his (typed in) commands into the proper signals to the control system. Some of the signals can be directly translated into a single signal, but a command as 'Reset' causes a sequence of signals to be send to 'control_exchange':

- Give an initial value to the PARAMETERS and STATUS by issuing some STATUS_COMMANDs accompanied by the proper NEW_STATUS_DATA.
- Let all the contents of the connection memories in the TST-switch be defined by performing a series of set up requests.

The supervisor can set up a connection by issuing an ADDRESSSED_CONNECTION to 'control_exchange'. This need not be a double connection, as for the concentrator, but broadcast connections are not allowed, so each input (or output) address can only be part of one connection.
5.3.2 The error localisation strategy

The 'localise_errors' tries to determine regions in the exchange system which are responsible for repeated errors, so that paths can be chosen through the TST-switch avoiding these bad regions and the CODECs can reject data which has passed through these regions.

Finding bad regions is restricted to finding combinations of lines and slots which are bad. For instance, it can be determined that the combination of line number 0 and intermediate slot number 20 is bad. Whether this error is caused by a T-switch or an S-switch is not certain. As can be seen in figure 5.2 (same as figure 2.3) the reason for this error could as well be a broken memory cell in the connection memory of T-switch number 0, as a broken connection memory cell in S-switch 0. It might as well have a different cause.

As can be seen in [Verhoof], the principle of the error localisation is the following: After finding an error in a connection, the path of the connection through the TST_switch is looked up in SWITCH_STATUS, and all connection memories which were part of it are checked. If no error can be found all regions of that path will have their TOTAL_WEIGHT_OF_ERRORS increased by the same WEIGHT. This value depends on the kind of error which was found. If the TOTAL_WEIGHT_OF_ERRORS of a region rises above LIMIT1, a test connection will be set up through this region. If the region is indeed bad, its TOTAL_WEIGHT_OF_ERRORS will very soon rise above LIMIT2 and it will be marked 'broken'. This will be reported to the supervisor, by means of DEFECT.

A disadvantage of the method is the fact that in the long run all regions will be marked 'broken'. It is up to the supervisor to regularly reset the value of TOTAL_WEIGHT_OF_ERRORS for all regions which are not broken.

The contents of TOTAL_WEIGHT_OF_ERRORS is influenced by errors in other regions; if an error is found, but the broken region cannot immediately be found, all regions of the path will be blamed. It is even possible that the broken region is not a
part of the path; A broken memory cell in an S-switch will cause two lines being switched to the same line at the same time. For this reason special care should be given to errors found in the connection memory of the S-switch: It can cause errors in any line and therefore the slot has to be marked 'broken' for all lines.

By means of CHECK_METHOD the supervisor can choose if the entire error localisation function must be performed, only parts of it, or not at all. This option is included to be able to make processing time available for call processing if required.

### 5.3.3 The data stores

The structure of some of the stores of the model (see [Verhoof]) is not very obvious at first sight and will be explained in this section.

The store SWITCH_STATUS describes how the switches in the TST-switch are set and it contains information about which elements are free and which ones are broken. It registers also the broken parts of the TST-switch, which cannot be confined to a switch.

The structure of SWITCH_STATUS was chosen in such way, that only switch positions that are possible in the TST-switch can be registered. For instance: If a switch is set to couple \{incoming line 0, incoming slot 10\} to \{incoming line 0, intermediate slot 20\}, it is impossible to couple at the same moment \{incoming line 0, incoming slot 11\} to \{incoming line 0, intermediate slot 20\}, because the memory cell for intermediate slot 20 can only contain one slot number at a time. To illustrate this, figure 5.2 repeats the structure of the switches as it was shown in figure 2.3.
Figure 5.2 The internal structure of the switches

A path through the TST-switch passes through four regions, indicated by \{IS,IL\}, \{IL,INT\}, \{INT,OL\}, \{OL,OS\}. Each region \{IL,INT\} must always be coupled to exactly one region \{INT,OL\}; If more than one region \{IL,INT\} is coupled to a region \{INT,OL\} two conversations will interfere. A region \{INT,OL\} being free implies another region \{INT,OL\} being double occupied. Because of the hardware structure (see figure 5.2) a region \{IL,INT\} is always coupled to a single region \{INT,OL\}.

* The same abbreviations are used here as in appendix I:
  IS = INCOMING SLOT, IL = INCOMING LINE,
  INT = INTERMEDIATE SLOT, OL = OUTGOING LINE,
  OS = OUTGOING SLOT
Because of this one-on-one relationship, the heart of a path through the TST-switch is formed by a CONNECTION_ITEM. It is indicated by \{IL,INT,OL\} and contains the couplings from \{IL,IS\} to \{IL,INT,OL\} and the couplings from \{IL,INT,OL\} to \{OL,OS\}. From the fact that a memory cell always contains exactly one data word follows that an item \{IL,INT,OL\} always is coupled to exactly one region \{IL,IS\}. Further, a CONNECTION_ITEM shows if either one of its regions contains broken AREAs (an AREA indicates a slice) and if the CONNECTION_ITEM is free or in use.

The broken areas, and whether or not the region is free, is registered for the regions \{IL,IS\} and \{OL,OS\} in respectively INCOMING_SLOT_ITEMS and OUTGOING_SLOT_ITEMS.

The store STATISTICS collects system information to be used for the design of later versions of the software. It is composed of ERROR_STATISTICS and TIMING_STATISTICS. The contents of the first has not yet been determined, the second contains the duration of the calls. In later versions this will make charging possible, but in Version 0 the subscriber number of the caller is not available to the control system. Instead, an internal identifier is used to make a global distinction between the duration of different calls.
6. Conclusion and recommendations

With the aid of ProMod a requirements model of the control system has been developed. The tool does not find any errors (see [Verhoof p.168]), so the model satisfies ProMod's demands. Furthermore it is as good as possible in agreement with the method of Hatley & Pirbhai and some concepts described in [Page-Jones]. But most important, it describes the requirements for the control system of the telephone exchange being developed at the Digital Systems section. It is, however, not up-to-date, because of some changes which were made after the start of the model building process: The model still counts on 64 incoming and outgoing lines, instead of only 4 in Version 0, and it supposes the CODECs to be present.

During the development of the requirements model some errors in the hardware design were revealed and ProMod could be tested. Though a number of bugs were found in ProMod (part of which is corrected in ProMod version V1.7a) it appeared to be a helpful in the development of requirement models for large systems.

Before implementing the control system the requirements model must be transformed into an architecture model, the model must be adapted to the changes in the hardware, and a suitable programming environment must be chosen. When building the architecture model attention must be given to the fact that ProMod can only generate a model for the software, and the Hatley & Pirbhai method should be used to developed the hardware architecture.
7. List of references

7.1 Reports of the telephony project

Aggenbach, M.H.J.M.
ONTWERP VAN EEN (4,2)-DECODER/CODER VOOR EEN DIGITALE
TELEFOONCENTRALE VOLGENS HET '(4,2)-CONCEPT'.
Eindhoven: Department of Electrical Engineering, Eindhoven University of
Graduation report. TUE-EB112

Duijkers, L.J.M.H.H.
IMPLEMENTATIE VAN EEN MULTITASKING OPERATING SYSTEM (DE
LEX).
Eindhoven: Department of Electrical Engineering, Eindhoven University of
Technology, October 1986.
Graduation report. THE-EB023

Hense, M.L.G.E.
ONTWERP VAN EEN TST-BUSTRANCEIVER VOOR EEN DIGITALE
TELEFOONCENTRALE VOLGENS HET '(4,2)-CONCEPT'.
Eindhoven: Department of Electrical Engineering, Eindhoven University of
Graduation report. TUE-EB190

Hoorens, P.V.W.
ONTWERP VAN EEN TIME SWITCH ELEMENT VOOR EEN DIGITALE
TELEFOONCENTRALE VOLGENS HET (4,2)-CONCEPT.
Eindhoven: Department of Electrical Engineering, Eindhoven University of
Graduation report. TUE-EB092
Van Engelen, W.J.
ONTWERP VAN EEN CONCENTRATOR VOOR EEN DIGITALE TELEFOONCENTRALE VOLGENS HET '(4,2)-CONCEPT'.
Graduation report. TUE-EB127

Van Etten, M.C.M.J.
EEN LOCAL EXECUTIVE DRAAIEND OP EEN IBM PC ONDER MSDOS.
Hobby report.

Van Lier, A.
DESIGN OF A TIME/SPACE SWITCH ELEMENT FOR A DIGITAL TELEPHONE EXCHANGE.
Graduation report. TUE-EB202

Verhoof, P.H.W.
REQUIREMENTS MODEL FOR THE CONTROL SYSTEM OF A DIGITAL TELEPHONE EXCHANGE.
PART 2: PROMOD SA-REPORT.
Witkam, L.F.
DIGITALE TELEFOON-CENTRALE.
ONTWERP EN CONFIGURATIE VAN HET CONCEPT VOOR EEN (4,2)-
CENTRALE.
Eindhoven: Department of Electrical Engineering, Eindhoven University of
Intermediate report.

7.2 Yourdon literature

DeMarco, T.
STRUCTURED ANALYSIS AND SYSTEM SPECIFICATION.

Hatley, D.J. and I.A. Pirbhai
STRATEGIES FOR REAL-TIME SYSTEM SPECIFICATION.
ISBN 0-932633-04-8

Myers, G.J.
RELIABLE SOFTWARE THROUGH COMPOSITE DESIGN.
(Also: New York: Von Nostrand Reinhold, 1975.)

Myers, G.J.
COMPOSITE/STRUCTURED DESIGN.
Page-Jones, M.
THE PRACTICAL GUIDE TO STRUCTURED SYSTEMS DESIGN.
ISBN 0-917072-17-0
bsk DFX 80 PAG

PROMOD COMPUTER AIDED SOFTWARE ENGINEERING USERS
MANUAL.
Version V1.6_2,

PROMOD COMPUTER AIDED SOFTWARE ENGINEERING USERS
MANUAL.
Version V1.7a,

Ward, P.T. and S.J. Mellor
STRUCTURED DEVELOPMENT FOR REAL-TIME SYSTEMS.
ISBN 0-917072-51-0

Yourdon, E. and L. Constantine
STRUCTURED DESIGN: Fundamentals of a discipline of computer program and systems design.
bsr DFN 79 YOU.
(Also: Englewood Cliffs (NJ): Prentice Hall, 1979.)
7.3 Other references

Boonen, P.E.J.
THE DEVELOPMENT OF BITBUS Firmware and the Accessory VME/ERM BITBUS DRIVER FOR THE dDCM804 BITBUS BOARD.
Graduation report. TUE-EB136

Heetman, A.
Department of Electrical Engineering, Eindhoven University of Technology, December 1979.
Lecture notes. nr. 5,564

Krol, Th.
HET '(4,2)-CONCEPT' VOOR HET MAKEN VAN FOUTEN tolerancende COMPUTERS.

Krol, Th.
(N,K) CONCEPT FAULT TOLERANCE.

Parnas, D.L. and P.C. Clements
A RATIONAL DESIGN PROCESS: How and why to fake it.
Ronayne, J.P.
INTRODUCTION TO DIGITAL COMMUNICATIONS SWITCHING.
ISBN 0-273-02178-8
bse LTD 86 RON

Vijay Ahuja.
DESIGN AND ANALYSIS OF COMPUTER COMMUNICATION NETWORKS.

Wallace, R.H. et. al.
A UNIFIED METHODOLOGY FOR DEVELOPING SYSTEMS.

ISBN 7 8529 6325 4, ISSN 0537-9989
bse LTC 73 SOF

SEVENTH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING FOR TELECOMMUNICATION SWITCHING SYSTEMS. Bournemouth (UK), 3-6 July 1989.
ISBN 0 8529 6381 5, ISSN 0537-9989
bse LTC 73 SOF

For a list of more works about the control system of a digital telephone exchange see:
Glossary of terms

(4,2)-concept Concept for a redundancy system developed by ir. Th. Krol. See [Krol 1983] and [Krol 1986].

Architecture model Technology dependent part of the system description according to [Hatley] and others. See §4.1.2, and §4.1.4 for description.

Architecture template Structure as in figure 4.4, having a functional and control processing core, surrounded by four processing units, which establish the connection to the physical outside world.

CASE = Computer Aided Software Engineering

CFD = Control Flow Diagram. (See §4.1.3.)

Channel associated signalling Signalling method which uses PCM-slots for the transport of control information to and from other exchanges. See [Ronayne pp.140-145].

Child process Process one level lower in the hierarchy of flow diagrams than the concerned (parent) process.

CODEC Integration of a decoder and a coder (in this order). Its task is to correct errors and to supply the control system with information about error occurrence.

Coder Encoding device of the (4,2)-system. For coding principle see [Aggenbach], [Krol 1983] or [Krol 1986]. Does not communicate with the control system.

Cohesion A measure of the strength of functional association of processing activities. (See [Page-Jones chap.7].)
Concentrator  Device connecting only part of its input lines to its output lines. Because telephone lines have a low occupation rate, a concentration factor of 11 is allowed for this exchange, without resulting in too much blocked calls.

Connection  Throughput from a slot on a certain incoming line to a slot on a certain outgoing line. For the TST switch a connection consists of a path, but when the word 'connection' is used, no attention is payed to the position of the switches.

Connection memory  RAM memory in a T- or S-switch containing for each slot the current position of the switch. (see figure 2.3)

Control flow  Flow containing control information. Shown as a dashed directed arc in a flow diagram. The difference between data and control is difficult to describe and depends on the interpretation of the model maker.

Coupling  The degree of dependence of one module upon another; specifically, a measure of the chance that a defect in one module will appear as a defect in the other, or the chance that a change to one module will necessitate a change to the other. (See [Page-Jones chap.6].)

CSPEC  Control Specification. (See §4.1.3)

CSS = Common Channel Signalling: Signalling method which has, unlike channel associated signalling, a separate network for the transport of control information between exchanges. See [Ronayne pp. 140-145].

Data flow  Flow containing information. Shown as a directed arc in a flow diagram. The difference between data and control is difficult to describe and depends on the interpretation of the model maker.

Data memory (Speech memory)  RAM memory in a T-switch used to temporarily store the PCM-data. Makes it possible to change the sequence of slots.
Decoder  Decoding device of the (4,2)-system. Errors, if not too many, can be corrected, thanks to redundancy coding. For coding principle see [Aggenbach], [Krol 1983] or [Krol 1986].

Deconcentrator  Device having more output lines than input lines, which performs the reverse function of the concentrator.

DFD  = Data Flow Diagram. (See §4.1.3.)

Enhanced requirements model  Intermediate phase during transformation from requirements model to architecture model. (See §4.1.5.)

FD  = Flow Diagram. Either DFD (Data Flow Diagram) or CFD (Control Flow Diagram)

Flow  Either data flow or control flow.

Local call  Call between subscribers which are both connected to the same digital subscriber stage.

Module  Collection of program statements. One of the basic elements of a structure chart.


parent process  Process one level higher in the hierarchy of flow diagrams than the concerned (child) process.

Path  Route through T1-, S- and T2-switch, making a connection in the TST-switch.

PCM  = Pulse Code Modulation. (See [Ronayne].)
Permanent error  Error being constantly present.

Primitive process  Lowest level process: A primitive process is not divided into subprocesses.

Process  Part of the hierarchical structure described in §4.1.3.1, representing an action, which produces its outgoing flows from its incoming flows.

ProMod  A CASE-tool. (See [ProMod].)

PSPEC  Process Specification. (See §4.1.3.)

Redundancy  Deliberate abundance of data, which is used to correct errors.

Requirements model  Technology independent part of the system description according to [Hatley] and others. See §4.1.2, and §4.1.3 for description.

SDL = Software Developing Language of CCITT. (See e.g. [Telecom Sw.].)

Slice  Part containing all functional elements of a system protected by redundancy coding. According to the (4,2)-concept a system consists of four identical and independent slices. (See §2.2.1.)

Slot  Time interval in a TDM system assigned to a channel.

Slot number  Number indicating a slot in a PCM-TDM signal. A slot number represents a channel.

Space-switch (S-switch)  Switch in a PCM-TDM system, which allows each incoming slot to be directed to an individually chosen outgoing line, but does not permit a change in the sequence of slots.
Structure chart  A graphic method for depicting the partitioning of a system into modules, the hierarchy and organization of those modules, and the communication interfaces between the modules. (See [Page-Jones].)

switch  Either a T1-, S-, or T2-switch. The position of a switch is a rather figurative expression, meaning the current contents of the connection memory of a switch.

T1-switch  Time switch connected to an incoming line of the TST-switch.

T2-switch  Time switch connected to an outgoing line of the TST-switch.

TDM  = Time Division Multiplexing. (See [Ronayne].)

Terminator  Parts of a context flow diagram representing connections to the outside world, i.e. to objects outside of the system. (See §4.1.3.1.)

Time-switch (T-switch)  Switch in a PCM-TDM system, which allows each incoming slot to be directed to an individually chosen outgoing slot, and thus a change in the sequence of slots.

Transient errors  Errors occurring randomly. They cannot be traced any more after their occurrence.

TST-switch  Switch configuration, used for the central switch part of the considered exchange, composed of respectively (from input to output) a T-switch, an S-switch and a T-switch.

Version 0  First prototype version of the considered telephone exchange, to be built in the near future. Version 0 will not include all essential parts of a telephone exchange and will be used to test the present design.
annex I  Version 0 of the TST-switch
annex II  Read and write protocols for the TST-switch

Read protocol

Write protocol
annex III Lay-out of a CODEC

**Inputs**

PCMIN0..3  
Incoming PCM-lines from the four different slices.

FRMSYNC  
Frame synchronisation pulse indicating the start of a frame.

BITCLK  
4096 KHz clock. The bitrate is 2048 kbit/s.

NRESET  
Reset; Active low.

NDS  
Data strobe; Active low. Indicates that the data on the databus is valid.

RD_NWR  
Determines the direction of data transport. In a write cycle RD_NWR is low, in a read cycle RD_NWR is high.

NSEL  
Selection of CODEC; Active low. Indicates that the data is valid.

A<0:6>  
Addressbus. Memory is selected with addresses 0 to 63, the interrupt register with address 64.
### Outputs

<table>
<thead>
<tr>
<th>PCMOUT0..3</th>
<th>Outgoing PCM-lines; Only one of them is used, depending on the error area.</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>Interrupt.</td>
</tr>
<tr>
<td>NDTACK</td>
<td>Indicates the end on the bus cycle; Active low. When the processor detects NDTACK low on a write cycle, the bus cycle is ended. When the processor detects NDTACK high on a read cycle, data is clocked in. (open collector output.)</td>
</tr>
</tbody>
</table>

### Input/Output

| D<0:4>     | Bidirectional databus.                                                   |
annex IV

Estimation of the calculation time on a PC

The goal of this estimation is to give a general idea whether or not a PC is fast enough to handle the traffic of the telephone exchange. This estimation was made in an early stage of the project, so some parameters may have been changed.

Presumed characteristics PC:
Clock frequency: 10 MHz.
Average duration machine command: 10 clock pulses = 1\(\mu\)s.

Characteristics TST-switch, CODEC:
(Maximum) clock frequency CMDCLK: 800 Khz. (At least 8-10 times smaller than BITCLK)
Bit-time: 1/800 KHz = 1.25 \(\mu\)s.

Processing times:
select a switch: ca. 40 \(\mu\)s.
select 3 switches: \(3 \times 40 \mu s = 120 \mu s\)
write to CODEC:
ca. 20 machine commands: 20 \(\mu s\)
acknowledge time: \(2.5 \mu s\)
ca. 25 \(\mu s\)
read from CODECs (4 slices):
ca. 22 machine commands: 22 \(\mu s\)
acknowledge time: \(2.5 \mu s\)
\(4 \times \text{ca. } 25 \mu s = 100 \mu s\)
set up a connection: \(120 \mu s + 25 \mu s = 145 \mu s\)
break down a connection: \(100 \mu s + 25 \mu s = 125 \mu s\)
total processing time for a conversation (2 connections):
\(2 \times (145 \mu s + 125 \mu s) = \text{ca} \ 550 \mu s \approx 0.5 \text{ ms}\)

routin

Calculation time for routing is dependent on the data structure used to store the connections. Two different structures were designed and their performance was measured in terms of processing time needed for routing and memory occupation. The structures are not shown here because this would be far too detailed for the rough guess that is made here for the calculation time. The results of the comparison are shown below:

- data structure 1 (as few memory as possible: bitmap)
  memory occupation: \(\text{ca } 6 \text{ Kbyte}\)
  duration one look-up (estimation): \(\text{ca } 260 \mu s\)
data structure 2 (as fast as possible: pointer structure)
memory occupation: ca 200 Kbyte
duration one look-up (estimation): ca 115 \mu s

One look up in structure 2 immediately finds a free slot (switch), whereas the slot (switch) in structure 1 may be occupied, so that another search is necessary. In structure 1 the duration of the routing is dependent on the percentage of slots (switches) that is occupied. For an occupation percentage of 50% the following estimated total calculation times per call are obtained:

- data structure 1: \(2 \times 1.8 \text{ ms} = 3.6 \text{ ms}\)
- data structure 2: \(2 \times 0.6 \text{ ms} = 1.2 \text{ ms}\)

(Though the actual look-up time is about nine times higher for structure 1, the difference in the final result is less, because the times for communication with TST-switch and CODEC, as well as the times for administration are essentially the same for both cases.)

Presumed call characteristics

number of physical lines per PCM-line: 750
subscriber busy hour calling rate (BHCR): 0.04 E
Average duration of call: 3 min.
Calls and call durations are Markov distributed.

- PCM-slot BHCR: \(750/64 \times 0.04 \text{ E} = 0.47 \text{ E}\)
- exchange BHCR: \(64 \times 750 \times 0.04 \text{ E} = 1925 \text{ E}\)

Demand on waiting time

In only 1% of the cases the control system may take more than 0.5 seconds after receiving a call to start setting up the connection:

\[ P[ T \leq 0.5 \text{ s}] \geq 0.99 \]

As there is one server (the PC), call digestion is performed in a M/M/1-queue. Therefore, according to [Heetman p.47]

\[ P[ T \leq t ] = 1 - e^{-\lambda t} \]

with \( \lambda = \frac{\lambda}{\mu} \);
\( \lambda = \text{average arrival rate} \), \( \mu = \text{average service rate} \)
\( T = \text{waiting time} \), \( \tau = \text{service time} = 1/\mu \).

\[ \lambda = 1925 \text{ E} / 3 \text{ min} = 642 \text{ min}^{-1} \]

\[ \mu \geq 1127 \text{ min}^{-1} \text{ and } \tau \leq 53 \text{ ms} \]
Here $\tau_c$ is a first estimate for the maximum total calculation time for one call. Because the system is not an exact $M/M/1$-system the result must be seen as an estimate.

Some considerations for a more exact determination are:
- Call set up requests and call break down requests should both be seen as arrivals. Therefore \( \lambda \) should be doubled. This will make $\tau_c$ about half as large, so that the sum of both service times will again be about 50 ms.
- Service time is not Markov distributed for data structure 2: In this case a $M/D/1$ queue should be taken as a model. According to [Heetman pp.112-113] this will half $E[n]$, so that $\tau_c$ will increase.

**Conclusion**

A reasonable estimation is that the set up and break down times must both be smaller than 25 ms.

The estimated calculation time (set up and break down) was found to be between 1.7 ms (1.2 ms + 0.5 ms) and 4.1 ms (3.6 ms + 0.5 ms). These numbers are estimates and do not take redundancy checking and error localisation into account, so in reality they might be 5 to 10 times higher.

Still the conclusion is that this realization is possible, though it might not be wise to spill too much calculation time.
annex V Interfaces for the communication with TST-switch and CODEC

1. Interface between PC and TST-switch
8-bit address-decoder: 8-bit latched demultiplexer (output changes only if Clock_in is low). Line 192 to 255 are not used.

2-bit address-decoder: 2-bit latched demultiplexer (output changes only if Clock_in is low).

16-bits P.S.T.: 16-bits parallel-serial-transformer. Begins on the rising edge of Start transmitting the 16 bits on the parallel input serially with a bit time equal to the cycle time of CLK (ca. 1 s). Stops after 16 bits until the next rising edge on Start.

7-bits S.P.T.: 7-bits serial-parallel-transformer. Reads in a serial signal. This signal has a startbit 0 and a bit time equal to the cycle time of CLK. When the 7 bits are offered in parallel at the output, Data_Valid is made high. With a low pulse on Reset a new read process is started.

AND: 4 x 2-input AND. One of the inputs is connected to the lowest input and the other one with one of the lines van of the 4-bits-bus.

CLK: Clock signal. the frequency is BITCLK/8 maximum (see [Aggenbach]).
Procedure to write data to the TST-switches

1. Put the 8-bits TST-address and the 2-bits chip-number on the Data_Out-bus.

2. Make the Select_all_Chips-line low, if all four slices must be written.
together. (In rest this line is high.) This makes all \( \overline{CS} \)-lines low.

3. Clock the TST-address and the chip-number into the address-decoder, by making \( \overline{SEL} \) low for a short moment. This will cause the \( \overline{SEL} \) and the \( \overline{CS} \)-line belonging to the addressed TST to become high, while the other \( \overline{SEL} \) and \( \overline{CS} \)-lines stay low. The address-decoder must latch the clocked-in address, so that the \( \overline{SEL} \) and \( \overline{CS} \)-lines stay active during the entire period.

4. Make \( R/W \) low.

5. Put the address and the data-word on the Data_Out-bus. Depending on the switch being controlled, this are 7+6 (TS1), 7+7 (SS) or 6+7 (TS2) bits. If \( n \) is the number of address-bits and \( m \) is the number of data-bits, then the address is sent over lines 2 to \((n+1)\) and the data over the lines \((n+2)\) to \((n+m+1)\).

6. Make Data_Set_Ready high for a short moment and then low again. The write-action takes place now: The P.S.O. starts sending.

7. Wait until the write-process is ready. (Implementation to be determined: This can happen by having the PST send a signal to the PC, but also by making the program in such a way, that always enough time will be waited.)

8. Make Select_all_Chips high.

9. Make all SEL-lines 0 and all \( \overline{CS} \)-lines high-impedance. (Implementation still to be determined.)
1. Put the 8-bits TST-address and the 2-bits chip-number on the Data_Out-bus.
(Select_all_Chips-line must stay high, because it is not possible to read data from all four CODECs at the same time. Probably a security must be made in hardware, so that the input of the AND is always high, if R/W is high.)

3. Clock the TST-address and the chip-number into the address-decoder, by making SEL low for a short moment. This will cause the SEL- and the CS-line belonging to the addressed TST to become high, while the other SEL- and CS-lines stay low. The address-decoder must latch the clocked in address, so that the SEL- and CS-lines stay active during the entire period.

4. Make R/W high.

5. Put the address on the Data_Out-bus.

6. Make Data_Set_Ready high for a short moment and then low again.

7. Wait until Data_Valid is high.

8. Read Data_In.

9. Make all SEL-lines 0 and all CS-lines on high-impedance.
(Implementation still to be determined.)
2. Interface between PC and CODEC

- 2-bit address-decoder: 2-bit latched demultiplexer (output changes only if Clock_in is low).
**AND:** 4 x 2-input AND. One of the inputs is connected to the lowest input and the other one with one of the lines van of the 4-bits-bus.

**3-state-buffer:** Buffer for the prevention of bus conflicts on the bidirectional D-bus. When the state-input is high, then the outputs are high-impedance, otherwise the input signal is passed to the output.
1. Put the 6-bits CODEC-address and the 2-bits chip-number on the Data_Out-bus.
2. Make the Select_all_Chips-line low, if all four slices must be written together. (In rest this line is high.) This makes all CS-lines low.

3. Clock the TST-address and the chip-number into the address-decoder, by making SEL low for a short moment. (See also item 3 of the TST-switch.)

4. Make R/W low. The buffer lets the data coming in on the Data_Out-bus pass to D.

5. Put the 7-bits address and the 5-bits data-word on the Data_Out-bus.

6. Make NDS low. The write-action takes place now. (In rest NDS must be high.)

7. Wait until NDTACK is low, as a sign that the CODEC has copied the data.

9. Make NDS high.

10. Make Select_all_Chips high.

11. Make all NSEL-lines 1 and all CS-lines on high-impedance. (Implementation still to be determined.)
Procedure to read data from a CODEC

1. Put the 6-bits CODEC-address and the 2-bits chip-number on the Data_Out-bus.
(Select_all Chips-line must stay high, because it is not possible to read data from all four CODECs at the same time. Probably a security must be made in hardware, so that the input of the AND is always high, if R/W is high.)

3. Clock the TST-address and the chip-number into the address-decoder, by making SEL low for a short moment. (See also item 3 of the TST-switch.)


5. Put the 7-bits address on the Data_Out-bus.


7. Wait until NDTACK is low, as a sign that data is ready.

8. Read Data_In

9. Make NDS high.

11. Make all NSEL-lines 1 and all CS-lines on high-impedance. (Implementation still to be determined.)
annex VI  Flow diagrams of the entire system

1. Context flow diagrams
2. First level flow diagrams

```
<table>
<thead>
<tr>
<th>Command</th>
<th>Remote Request</th>
<th>Local Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor Information</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control Switching</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concentrate Deconcentrate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switch Conversations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DFD 1 exchange conversations

```

```
<table>
<thead>
<tr>
<th>Command</th>
<th>Remote Request</th>
<th>Local Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor Information</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control Switching</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concentrate Deconcentrate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switch Conversations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CFD 1 exchange conversations

```
3. Enhanced requirements model

Only the data flow diagram is shown:

- Control switching
- Concentrate deconcentrate
- Convert A/D D/A
- Monitor information
- Encode decode
- Statistics
- Switch conversations
- Perform redundancy check
- Read keyboard
- Display on screen
- Send remote request
- Send to remote exchange
- Receive from remote exchange
- Get remote request
- Connections
annex VII Set up procedure in pseudo-Pascal

{All lines are virtual lines: They are in fact channels (slots in a PCM-system).

For configuration see figure 5.1. Switch 1 corresponds to "switch" and switch 2 corresponds to "remote switch".}

constant sub_max = 45000; {subscribers are numbered 0..sub_max}
loc_max = 4031; {local lines are numbered 0..loc_max}
max = 4095; {lines (local + remote) are numbered 0..max}
rem_base = 4032; {remote lines are numbered rem_base..max}
base = N*64; {lines between local exchange of caller and switch are numbered base..base+63 (N<63)}

type subscriber_line = record
  in : line; {from subscriber to local exchange}
  out : line; {from local exchange to subscriber}
end;

local line = record
  in : line; {from local exchange to switch}
  out : line; {from switch to local exchange}
end;

remote line = record
A to B : line;
B to A : line;
end;

local_set = record
  in : 0..loc_max;
  out : 0..loc_max;
end;

remote_set = record
  in : rem_base..max;
  out : rem_base..max;
end;
mixed_set = record
  in : 0..max;
  out : 0..max;
end;

acktype = (ACK, NACK);
subscriber_number = 0..sub_max;

var subscriber = array(0..sub_max) of subscriber_line;
  {lines from subscriber to their local exchange}
lines_in = array(0..max) of line;
  {input lines of a switch.
   line numbers 0..locmax from a local exchange,
   line numbers rem_base..max from a remote exchange.}
lines_out = array(0..max) of line;
  {output lines of a switch.
   line numbers 0..locmax to a local exchange,
   line numbers rem_base..max to a remote exchange.}
caller, receiver : subscriber_number;
ack = acktype

process local_exchange_of_receiver(loc2 : local_set,
  sub2 : subscriber_number,
  var acknowledge : acktype);
  {respond to call set-up request of switch or remote switch.}
begin
  if blocked
    then acknowledge := NACK
  else
    begin
      connect(subscriber(sub2).in,
        lines_in(loc2.in));
      connect(lines_out(loc2.out),
        subscriber(sub2).out);
      acknowledge := ACK;
    end;
end; {local_exchange_of_receiver}

process remote_switch(rem : remote_set, sub2 : subscriber_number;
  var acknowledge : acktype);
  {respond to call set-up request of switch.}
begin
  remote : remote_set;
remote.in := translate(rem.out);  {line numbering may be different in switch and remote switch.}
remote.out := translate(rem.in);
switch(remote,sub2,acknowledge); {a remote_switch is a switch}
end; {remote_switch}

process switch(loc1 : mixed_set, sub2 : subscriber_number;
  var acknowledge : acktype);
  {respond to call set-up request of local exchange.}
begin
  var
  loc2 : local_set;
  rem : remote_set;
  blocked : boolean;
begin
  if in_this_switch(sub2) {look if the call is local to this switch.}
    then
      begin
        assign(loc2, sub2, blocked);
        {find a free line to the local exchange of sub2. if not found then blocked:=True}
        if blocked
          then acknowledge := NACK
          else begin
            connect(lines_in(loc1.in),
              lines_out(loc2.out));
            connect(lines_in(loc2.in),
              lines_out(loc1.out));
            local_exchange_of_receiver(loc2,
              sub2,acknowledge);
          end;
      end
    else
      begin
        assign(rem, sub2, blocked);
        {find a free line to the remote exchange of sub2. if not found then blocked:=True}
        if blocked
          then acknowledge := NACK
          else begin
            connect(lines_in(loc1.in),
              lines_out(rem.out));
            connect(lines_in(rem.in),
              lines_out(loc1.out));
            remote_switch(rem,sub2,acknowledge);
          end;
      end;
  end; {switch}
process local_exchange_of_caller(sub1, sub2 : subscriber_number;
    var acknowledge : acktype);
{respond to call set-up request of subscriber.}
var blocked : boolean;
loc1 : record
    in : base..base+63;
    out : base..base+63;
end;

begin
    if in_this_conc(sub2) {look if the call is local to this local exchange}
        then connect(subscriber(sub1), subscriber(sub2))
        else begin
            assign(loc1, blocked);
            {find a free line to the switch. if not found then blocked:=True}
            if blocked
                then acknowledge=NACK
                else begin
                    connect(subscriber(sub1).in,
                        lines_in(loc1.in));
                    connect(lines_out(loc1.out),
                        subscriber(sub1).out);
                    switch(loc1,sub2,acknowledge);
                end;
    end; {local_exchange_of_caller}
end {main}

begin
    input(caller,receiver); {read who wants to call with whom.}
    local_exchange_of_caller(caller, receiver, ack);
    {pass set-up call to local exchange of caller}
    if ack = ACK
        then start_conversation
        else beeps;
end.