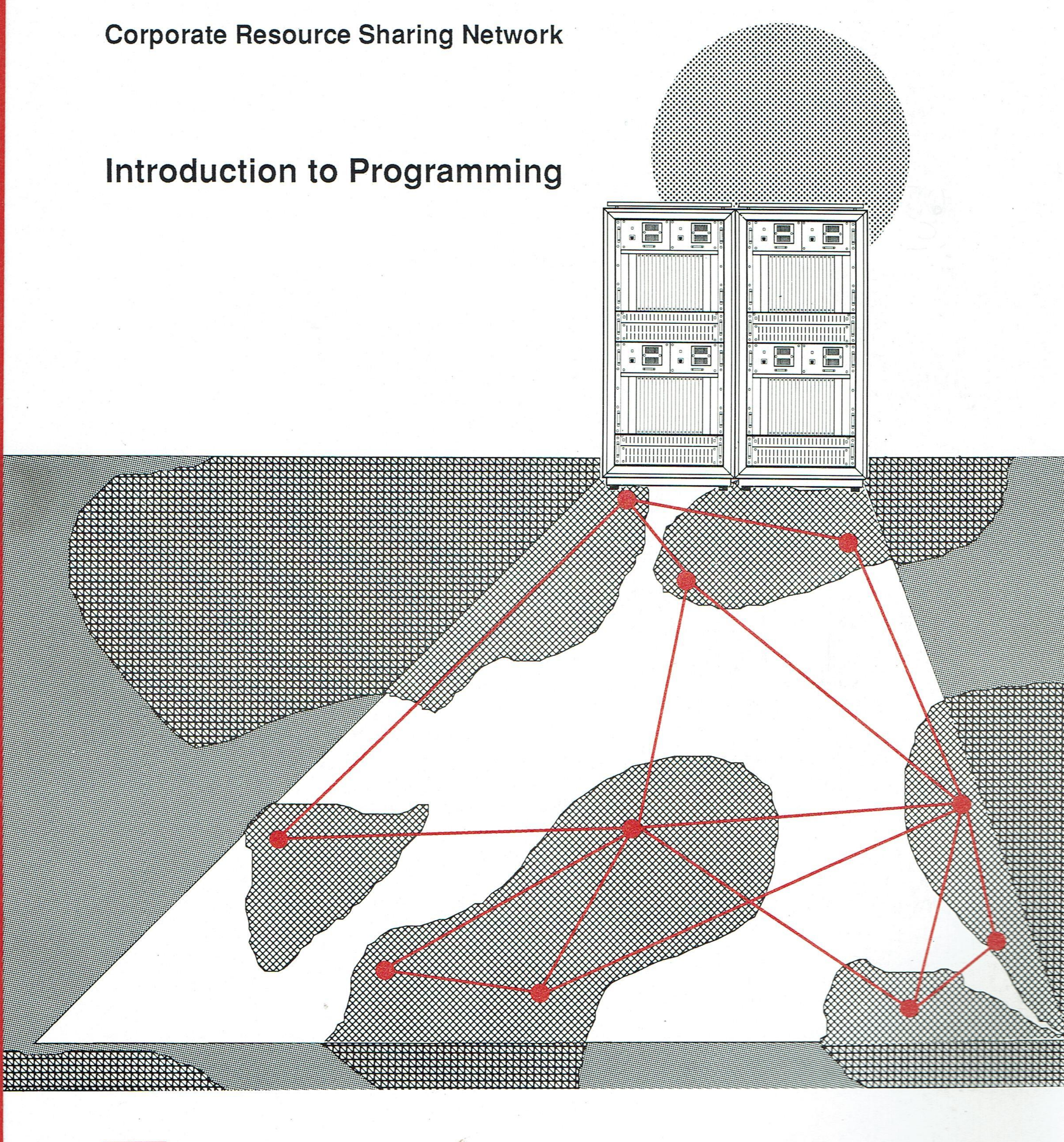
Corporate Resource Sharing Network

Introduction to Programming



CR Systems

Corporate Resource Sharing Network

Introduction to Programming

ISSUE: 1.0 BY/DATE : HRH/870301 LAST PAGE: 80

PREFACE

This document is intended for customers interested in supplementing their CRSN software with new applications. When the CRSN system is delivered, all necessary software elements are proven and reliable, suited to your current needs.

However, all companies and organizations go through development phases and what is well-suited today may be unsuitable tomorrow. Therefore, companies implementing the CRSN will sooner or later be interested in augmenting their network applications. For that purpose, a large amount of system software and utilities such as programming languages and test tools are available and delivered together with the system.

OVERVIEW

Chapter 1 describes the CRSN architecture, the hardware and software concepts and the architectural specifications.

Chapter 2 deals with the development environment, the programming languages and the test tools.

Chapter 3 treats the Resource Handler concept and the functions it performs on different levels of the architectures.

Chapter 4 describes the procedures and practices for software development.

SUMMARY

This document gives the reader an introduction to the software development on the CR80 computer used in CRSN.

CONTENTS PAGE

1 THE CRSN ARCHITECTURE

This chapter contains a description of the Architectural rules which is the foundation on which Corporate Resource Sharing Network (CRSN) is built. After an introduction, descriptions of the CR80 computer, the Software concepts and the design architecture will follow. Finally, the Data Interface Reference Manual (DIRM) is introduced together with the Prefixes.

CHAPTER CONTENTS **PAGE**

1 THE CRSN ARCHITECTURE

What do architectural rules mean when programming software?
Architectural rules mean the same as in the context of building houses.

When building houses, it is the architecture that ensures the construction is safe and which provides the desired functionality. Additionally, it may make the house appear cozy and beautiful.

It is obvious that the bigger the house, the higher need for architectural rules since otherwise the risk of mistakes is bigger.

Not everything will of course be covered by the architect. The workers building the house will make a lot of small decisions based upon their education and experience, guided by their knowledge of the architectural rules.

A lot of analogies exist between building houses and programming computer systems.

Both users and experts may have wishes as to how the building shall function and appear, wishes which may often contradict each other. This calls for an architect to design a building to serve the different purposes in an optimal way.

The same applies when creating big computer systems. There will be many requirements and wishes for the system, and they may contradict each other. Therefore, it is necessary to have an experienced architect to deliver an optimal architecture.

Tomorrow may bring new opportunities with which the architecture you choose today should be able to cope. This calls for an architect with vision and fantasy.

The next sections describe how the CRSN architecture is developed to fulfil what we believe will be the optimal solution in big data networks.

But before going into detail, we have to define some terms.

CRSN General Concepts and Terminology

The CRSN is a system solution for data networking. Meaning that it is a combination of software and hardware. Therefore, it is important to be familiar with some of the basic concepts and terms regarding:

- o The CR80 computer
- o The CRSN software
- o The CRSN architecture

1.1 The CR80 Computer

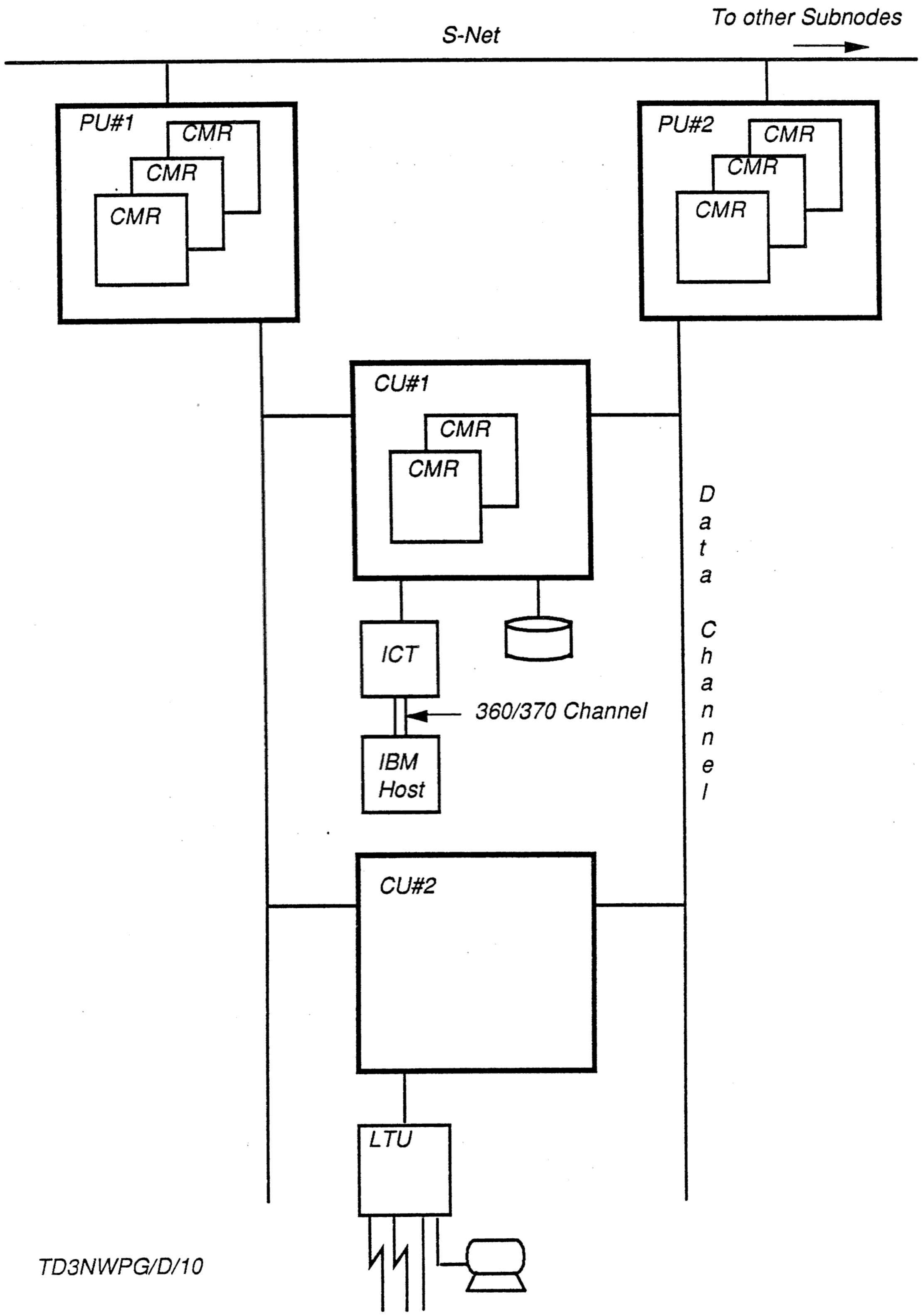
The CRSN is implemented on CR80 computers that are developed especially to cope with the demands arising from transporting lots of data in a secure way in big networks.

Fig. 1.1-1 shows the basic hardware elements of the CR80 computer. The figure highlights the items that are of interest to understand the philosophy of the CR80 computer.

There is a Supra-Net (S-net) which provides the possibility to connect a number of CR80's together, with a high speed serial connection.
When the CR80 computer is loaded with CRSN software and is part of a CRSN, then it is referred to as a **Subnode.**

There can be two **Processor Units** (PU) in the CRSN Subnode. One is sufficient, but when extreme availability is needed then two are recommended. They will both be loaded with identical code, at boot time, but only one will be active and the other standby. During normal operation, the processes in the active PU will copy vital control information to the equivalent standby processes in the other PU. This is called **checkpointing.** If the active PU fails, the standby will automatically go active, and the Subnode can continue to function.

The basic processing unit is called **Computer-Map-Ram (CMR).** There is one or more CMR-boards in the PU.



TD3NWPG/D/10

Fig. 1.1-1 The CR80 Computer

There are a number of **Channel Units** (CU). The CUs take care of the connections to external equipment, such as host-channels, host-links, printers, discs, terminal access lines, trunk lines, etc. The different types of external equipment are accessed via input/output units, of which some are dedicated and some are all-purpose.

If certain external equipment processes vast amounts of data, the CU will often be equipped with CMR(s) of the same kind used in the PU. In this way, the CU is able to execute some of the processing, thus relieving the PU of part of the burden.

The most common input/output unit is the **Line Termination Unit** (LTU). It is equipped with a Z80 processor, and it can handle up to 4 external lines. By exchanging the software, it is possible to service different types of lines, both synchronously and asynchronously. Today, most of the widely used protocols are implemented. The software in the LTU is loaded at boot time, but can be reloaded at any time as a network operation. And as a normal network operation, it is possible to change speed and other parameters just by altering some data in the LTU software.

Special hardware units, called ICT boards exist for interfacing to IBM 360/370 channels. When loaded with the proper software, the CMR is able to emulate Physical Unit Type 4 and physical Unit Type 5 in the IBM-SNA concept.

The functions assigned in the different layers of the Open System Interconnection (OSI) protocol are implemented as follows:

Level 1	Physical level is hardware-implemented.
Level 2	Link level is performed by software in the Z80 processor in the LTU.
Level 3	Higher level functionality is performed in CMRs - either in the CU and/or in the PU.

1.2 Software Concepts

Below there are some terms which are often used in computer programming.

Module	Some lines of sources code. The name indicates that programming takes place in a modularized way, with typically 500 - 1500 lines of code. A module is, of course compilable, but unable to run alone. When compiled it is called a Link Module.
Process	Process means an executable code with its associated data upon which it operates. It will consist of a number of link-modules. In the CR80 computer it is possible to let the same executable code operate upon different sets of data. In this way it is possible to have multiple incarnations of one process
Subsystem	One or more processes which together provide some well-defined functionality.

Operating System

The operating system is called MXAMOS which is an acronym for Mapped eXtended Advanced Multiprocessor Operating System.

It will always be a necessary part of the CR80 software, both when the CR80 computer is used as a development computer and when it is used as a CRSN Subnode.

Process Communication

In the CR80 computer the need for process to process communication is solved by the **Basic Communication Service** (BCS).

Basically BCS offers a messagegram service. However, messagegrams are delivered in the sequence sent.

A message is the data entity passed between any two communicating processes and a message is to be considered a sequential stream of bytes. BCS performs the following important tasks:

- Administration of pools of buffers.
- Administration of queues for messages
- Administration of messages.

The process that has a need for sending messages, asks BCS to allocate a memory area for pools of buffers. Then the process can write a message into the buffer.

If a process has a need for receiving messages, it must have a queue to receive it upon. The queue is memory area within the process data area that contains references of where messages sent to the process are placed.

So data is transported from a sender process to a receiver process in the following way:

- The sender places a message within its own buffer pool.
- The sender informs BCS that there is a message that is ready to be sent to a queue in the receiver.
- BCS places a reference of where to find the message in the queue of the receiver.
- The receiver can then read the message.
- When the receiver has read the message from the buffer, it has to dispose the buffer, so it can be used again by the sender.

Beside the trivial read and write of complete messages it is possible to read/write headers or trailers to the message, and it is even possible to read/write a message at a specific position.

Queues as well as buffers can be created/removed at any time during processing. This is essential when dealing with dynamic processing as in the CR80 computer.

1.3 CRSN Architecture

When developing the architecture for large data network some of the most important requirements are:

- o It shall be possible to interconnect to different types of external equipment.
- o It shall be possible to transport data from one point in the network to another in an easy, secure and fast and secure way.
- o It shall be possible to monitor and control the network from centralized location, including the possibility for operator interventions.
- o The software shall be developed in such a way that it shall be easy to shift to other types of hardware, when the technological development calls for that.

In CRSN these functionalities are implemented by software in the Subnodes. Such software is then referred to as **network elements.** From the above list of requirements the following list of applicable network elements can be identified:

- o Access Resource Handler(s) (ARH(s))
- o Transport Network (TN)
- o Network Management (NM)
- o System Control (SC)

In the following subsections a detailed description of the different network elements will be presented. However, to ease explanation they will be presented in reverse order of the above developed.

System Control Functionality

System Control (SC) implements a high level interface towards the operating system MXAMOS. Thus SC creates the environment in which all processes execute, and can therefore be considered as the operating system. But in the CRSN is has a broader scope, since SC will interact with the processes in the initialization.

During the initialization SC will provide the processes with queue names for the control interface to NM, TN and the SC network elements process. Additionally, it informs the process whether it shall be 'active' or 'standby'.

In the latter case the process will become an idle process just awaiting vital data updates from the active process. This is among other things the configuration and updates to the configuration. The data updates, also called checkpoints, allow the standby process to take over after a fatal error condition in the active process with a minimum loss of data and availability.

Network Management Functionality

Some of the major requirements towards the NM-functionality can be stated as follows:

- There must be a common way to describe all the different objects, that shall be monitored and controlled.

- It shall be possible to make changes in the network configuration. Small changes should be possible on-line, while it is acceptable that major changes must be performed off-line. However, changes in network configuration should not require recoding, but should only require updates of some data structures.

- It shall be possible to inform the network management if there is equipment that malfunctions.
- It shall be possible to monitor actual state of external equipment.
- It shall be possible to monitor data traffic leaving and entering the CRSN.
- It shall be possible to collect statistical information for the purpose of long term network planning.
- Some of the above functionalities will be performed automatically, while others require operator intervention. Therefore there shall be an advanced operator interface.

In the following it will be elaborated, how the different requirements above are satisfied, within the Network Management network element:

Common Description of Network Objects

An important concept is the **Resource.** Below find a list of different kinds of objects the network should be able to control and monitor:

- CR80 Hardware components:
  - CU-crates
  - PU-crates
  - CMR-boards
  - LTU-boards
  - ICT-boards

- CRSN software components:
  - Processes
  - Subsystems

- Externally connected equipments:
  - Host computer channels
  - Terminals
  - Printers
  - Terminal access lines
  - Trunk lines

In the CRSN context all these are called resources. To address them uniquely a **Resource Unit Identifier** (RUID) will be associated to them.

Resource Handler. A collection of one or more processes running in a Subnode. It will often be the same as a subsystem. The name refers to the fact that a resource handler will administrate a number of resources.

The necessary information that must be known about each resource is called the **resource record** of the resource. When talking about the collection of resource records that together holds information about all resources controlled by one resource handler, the term **configuration** is used.

The configuration will be present in the **network database,** and a subset of the configuration will be a part of the data area of the resource handler. It will be present both in the active and the standby process.

Network Configuration Changes

When a resource handler is initialized it will obtain its configuration from the Network Database in NM. During normal operation of the network a network operator can insert/delete resource records for resources that shall be made known/unknown to the resource handler.

An operator has also the opportunity to include/exclude resources in the network. This will only have the effect that the resource handler will start servicing/stop servicing the resource.

Malfunction of Equipment

The resource handlers should be programmed in such a way that they should detect if resources malfunction. The resource handler will send an **event** to NM. NM will maintain a list of all events, and if it is severe the network operator will be informed for further action.

Actual State of Equipment

A network operator can at any time request a description of what state a resource is in. Some standard states that must be maintained by all resource handlers are defined. The most important ones are:

- Off-line/On-line. The resource is known by the resource handler, but it is not served/served by the resource handler.

- Failing/not failing. Substate of On-line. The resource is serviced by the resource handler, but it has detected errors/not detected errors. Will often indicate degraded performance.

Monitoring of Data Traffic

In resource handlers connected to external equipment it is possible to **trace** the date leaving and entering the CRSN.

Collection of Statistical Information

Statistical information will cover items such as numbers of messages sent/received, number of transmission errors. Two types of statistic information can be defined. Permanent information that will be counted during normal operation, and temporary information where the counting can be invocated at any time by the network operator.

Advanced Operator Interface

To ease the operating of the network an advanced operator interface is provided by means of a Personal Computer which allows the operator to use menus or command as convenient. However, commands have been defined, so they can also be issued from an ordinary terminal in the network.

Other NM Aspects

It has been mentioned previously that the configurations are kept in the network data base. The network data base will also contain the following two items:

- o **Software database** which is a collection of all boot-modules that shall be loaded into the different Subnodes to perform the CRSN.
- o **Access control database** containing information to validate network users capabilities of using applications in the network. Thus the host computers can be relieved from access control.

Transport Net Functionality

The purpose of the TN is to transport data from one resource handler to another resource handler, regardless of where in the network the resource handlers are placed. If the resource handlers are placed in different Nodes, the data transport will be carried out by means of telecommunication lines and public data network. If the resource handlers are placed in different Subnodes in the same Node the transport will be carried out by means of the S-net, connecting the PUs. If the resource handlers are placed within the same Subnode then TN will use BCS.

But it is the purpose of TN to let all this be invisible to the two resource handlers communicating.

TN offers two different types of services. These are:

- o Connection oriented service
- o Messagegram service

The connection oriented service allows the user to obtain a fixed message flow between two partners, in popular terms a telephone service. When data are sent via a connection TN provides a protection of the messages which secures delivery to the other end. If this is not possible the two end-points will be notified. TN connection service secures that messages are delivered in the sequence which they were sent.

The messagegram service is in popular terms a mail box. TN will not secure that messages sent via the messagegram service are delivered to the user. If data is lost the users will not be notified. Furthermore messages can be received in another sequence as they were sent.

To use the TN services a user e.g. an ARH must make itself known to TN and obtain a TN address. This is done by enrolling to TN. After the enrolment to TN the messagegram interface can be used immediately. If a connection is needed a connection set-up procedure must be done. This involves the other connection end-point.

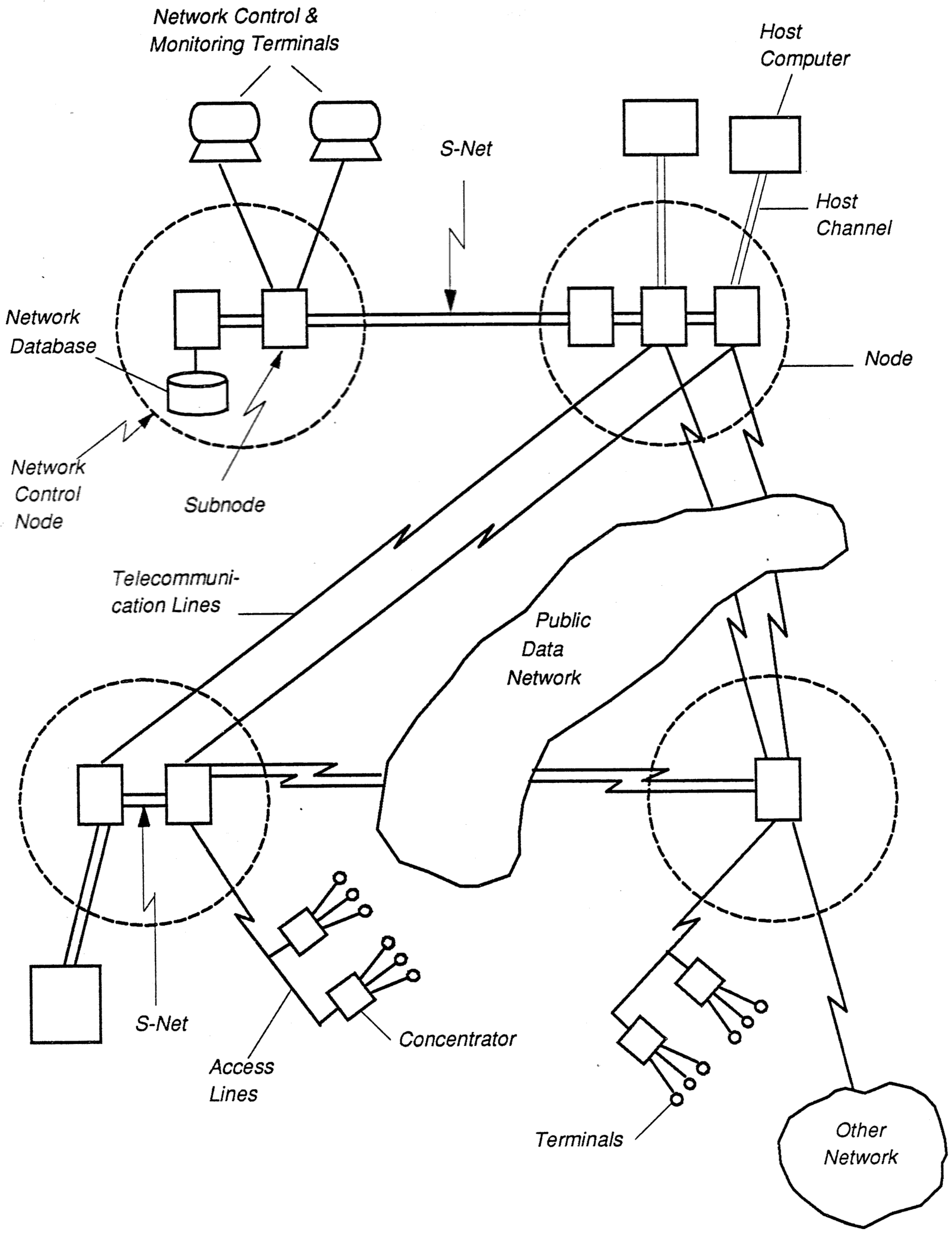
Access Resource Handler Functionality

Access Resource Handler is used to designate a resource handler which performs some functionality in connection with an external device.

So when creating a CRSN there will be NM, SC, TN and a various set of ARHs. And the differences between different CRSNs will be visible by the choice of ARHs. There will exist a number of the most commonly used standard interfaces and protocols, but it is also possible that new customers will have a need for a specially developed ARH.

It is possible for customers to program applications. It is then necessary to get familiar with the CRSN architectural rules, as the CR programmers must.

Fig. 1.3-1 shows how the network will be implemented by means of some CRSN Subnodes in geographically separated places, and by means of existing telecommunication services, such as Public Data Network and Telecommunication lines.



TD3NWPG/D/11

Fig. 1.3-1 The CRSN Network

1.4 Architectural Specifications

The above description of the CRSN architecture shows that NM, TN and SC will be common in all networks. There will be a different set of ARHs in different networks.

The NM, SC and TN will be developed and maintained by CR as well as there will exist a number of ARHs implementing commonly used interfaces.

Level 1 Architectural Specifications

Level 1 compatibility requires that a resource handler be able to work together with NM, SC, and TN. It may not be necessary to use all the provided services, but it is a must that the services needed are used in the predescribed way.

Level 1 Architecture is basically the split of functionality described in the former section. The level 1 architectural specifications are the documents describing the function of the different network elements: NM, SC, TN, and ARHs. It is important to specify the interfaces, clearly between the different elements. In addition to these, a special document exists which describes how different ARHs shall use the other Networks Elements in a standardized way.

This has the effect that many functions are decided beforehand which eases implementation of ARHs.

The rest of the implementation depends upon the requirements of the external equipment.

Level 2 Architectural Specifications

Level 2 Architectural requirements can be different, because they arise from the fact, that there often are resource handlers that shall solve tasks in cooperation. This will be the case when you consider the situations where a host computer shall be able to communicate with different types of terminals. Then the level 2 specifications will clarify how the protocols between the different access resource handlers shall work.

Level 3 Architectural Specifications

The fact that a level 1 architecture exists, and in some cases also a level 2 architecture, allows the development of software modules which can be used in different resource handlers with little or no modification.

Description of such standardized functionality is called level 3 architectural specification.

1.5 Data Interface Reference Manual (DIRM)

The CRSN product can logically be split into different systems that can be split into subsystems that can be split into processes which can be split into modules, etc.

To be able to control development on all levels, it is important to describe clearly the different groups of functions as well as the interactions between different systems, resource handlers, processes, etc.

This is done by means of the Data Interface Reference manuals (DIRM).

In the DIRM, the following items are covered:

- A syntactic description of the data
- A semantic description of the data
- This may not be enough, so there is also a description of the sequence of the commands, description of protocols, if retry must be used, if timers shall be used to wait for answers.

1.6 The Prefixes

There will be many situations, where data are common to more than one subsystem, process, module, etc.

This is to an example valid concerning basic data-structures used by the compilers. Also when two subsystems communicate together via a specific interface they use the same type of data structure.

It may cause errors if it is left to the subsystems alone to ensure that data structures are consistent. Therefore, this is done by use of prefixes.

All definitions of constants and data-structure used by more than one subsystem, module, etc., are defined in a Prefix. Then in the source code, there is an Include statement that ensures that the definitions from appropriate Prefixes are used when compiling the source code.

2 THE DEVELOPMENT ENVIRONMENT

In this chapter the programming languages used in the development of the Corporate Resource Sharing Network (CRSN) software are described. Also in this chapter there is a description of the common processing environment. A necessary element in the software development process is testing and debugging which is therefore also described.

CHAPTER CONTENTS **PAGE**

2 THE DEVELOPMENT ENVIRONMENT

The software development environment consists of the programming languages used to develop CRSN software and some basic development system software tools necessary to perform such necessary functions as entering the source code, compiling and linking the program modules. In Chapter 2.2, we look at the basic development system software, in Chapter 2.3, we discuss the various programming languages, and in 2.4, we view the Linker program. Chapter 2.5 describes the various test tools available to the software developer.

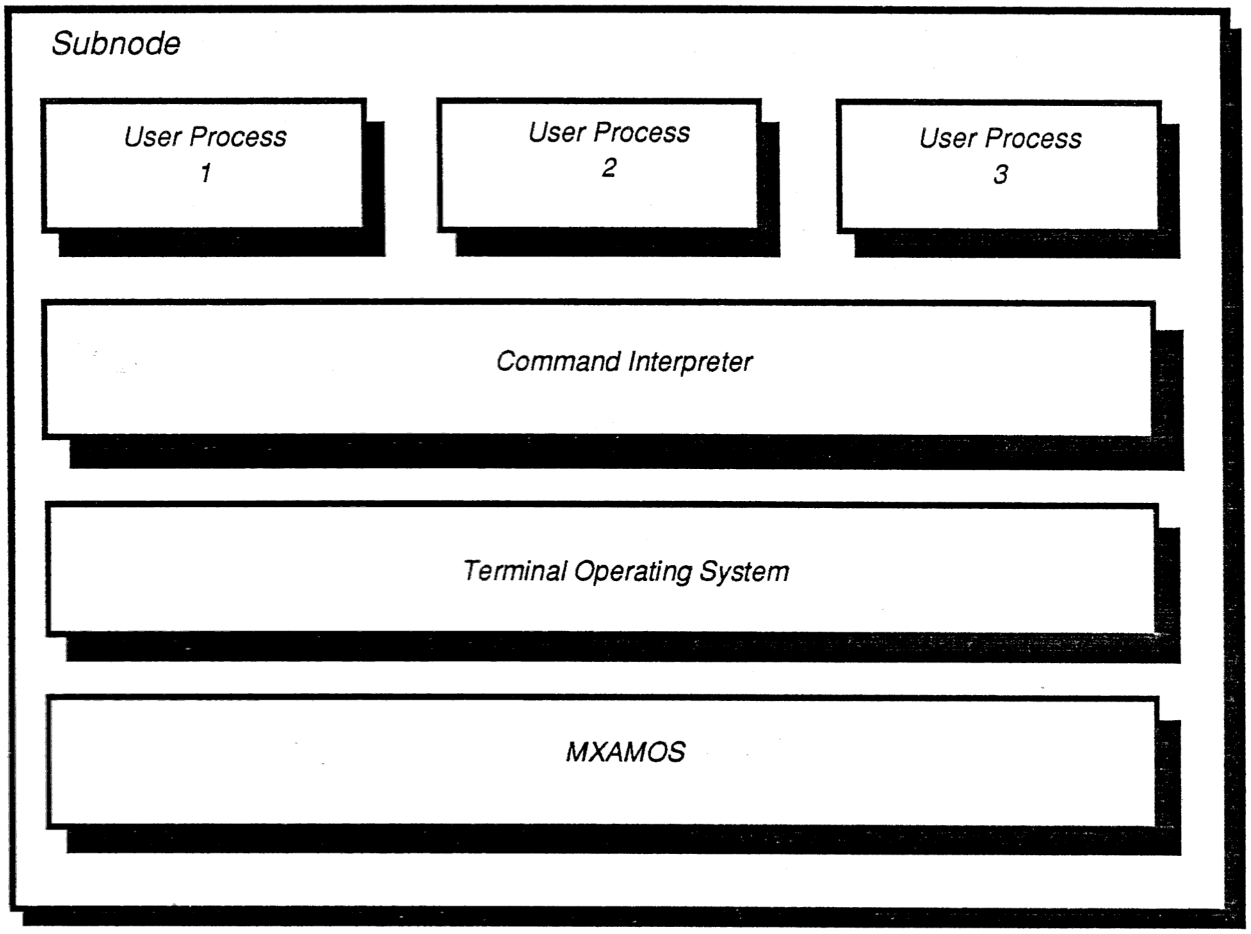
2.1 The Basic Development System Software

The CR80 Development System Software is designed for an operating system called MXAMOS: Mapped eXtended Advanced Multiprocessor Operating System.

In this document we look at some utilities controlled by MXAMOS, namely:

- The Terminal Operating System
- The Command Interpreter
- The CR80 Editor

The following drawing shows the process hierarchy on a CR80 development system:



TD4NWPG/D/25

Terminal Operating System (TOS)

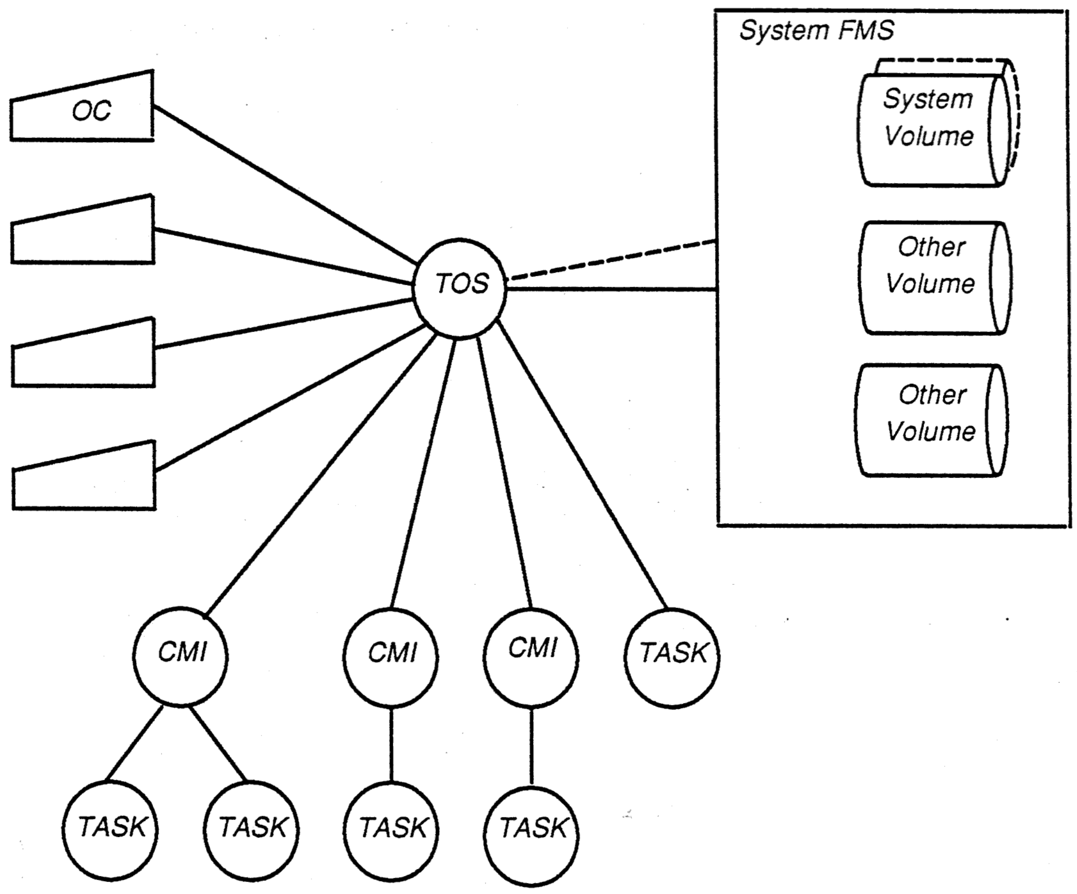
TOS is an operating system supporting terminal users. It is intended for interactive software development. The functions performed by TOS are invoked by two types of requests:

Operator command:

- Logon to the system
- Assigning peripheral devices of various kinds
- Execute programs
- Process monitoring and control
- File Maintenance
- Present status information

Programmed requests which makes it possible to carry out the same kinds of operation by taking the commands from a command file written by the user.

TOS now enters the work phase in which the environment can be visualized like this:



TD4NWPG/D/24

When a user logs on to TOS, a Command Interpreter (CMI) process is created. The CMI receives the user's command, e.g. requests to load and run programs like the Editor, a compiler, or a linker.

Command Interpreter (CMI)

The CMI is an interactive program which enables a user to execute and control CR80 programs from a terminal. The CMI can also process command input from a disk file. A long list of commands are available to the CMI user.

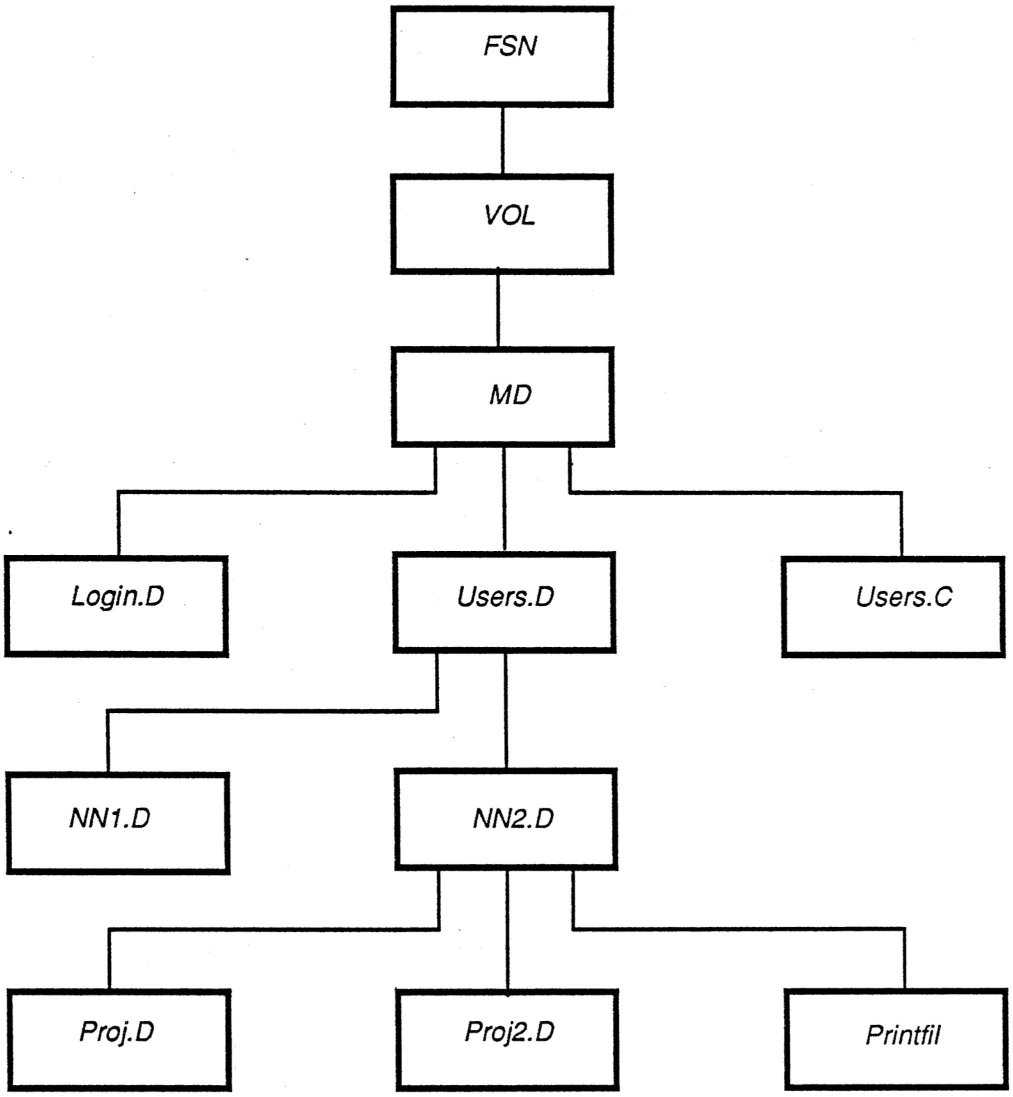
Entries not recognized as commands are assumed to be requests for execution of programs and will cause the CMI to look for a program with that name, and if found, to request the program to be loaded and started by TOS.

The CMI will recognize the operator keyboard as the standard input device and the terminal as the standard output device, but these can be redefined by CMI commands.

Initially after logon, the user is placed in the system directory. However, by using the USE and DEUSE commands the user can descend or ascend in the directory hierarchy. The WHAT command shows the user in which directory he is currently working.

The system volume will always contain a directory LOGIN.D and if the operator places a file with CMI commands in this directory, naming the file as his login ID, this command file will be executed when CMI is initially invoked. This facility is often used for setting up a standard user environment.

From the following figure you will get an impression of the file system organization on the CR80:



TD3/D/NWPG/D/1

CR80 On-line Editor

The editor operates on three files:

The command file contains the editor commands entered by the user. Normally, it will be from a terminal.

The display file is usually shown on a terminal. All output from the editor is directed to this file.

The editing file is the file on which the operator is working. The editing file is divided into lines. A line is defined as a string of maximum 132 ASCII characters.

The editor can be invoked to handle small files of maximum 1400 lines or big files of maximum 2791 lines.

2.2 Programming Languages

In developing the CRSN software, several programming languages have been used, namely Pascal, C and SWELL. Of these, SWELL is a low-level language, developed by CR.

In the following sections, we will describe the programming languages and their environments, showing some program examples to give an impression of the syntax.

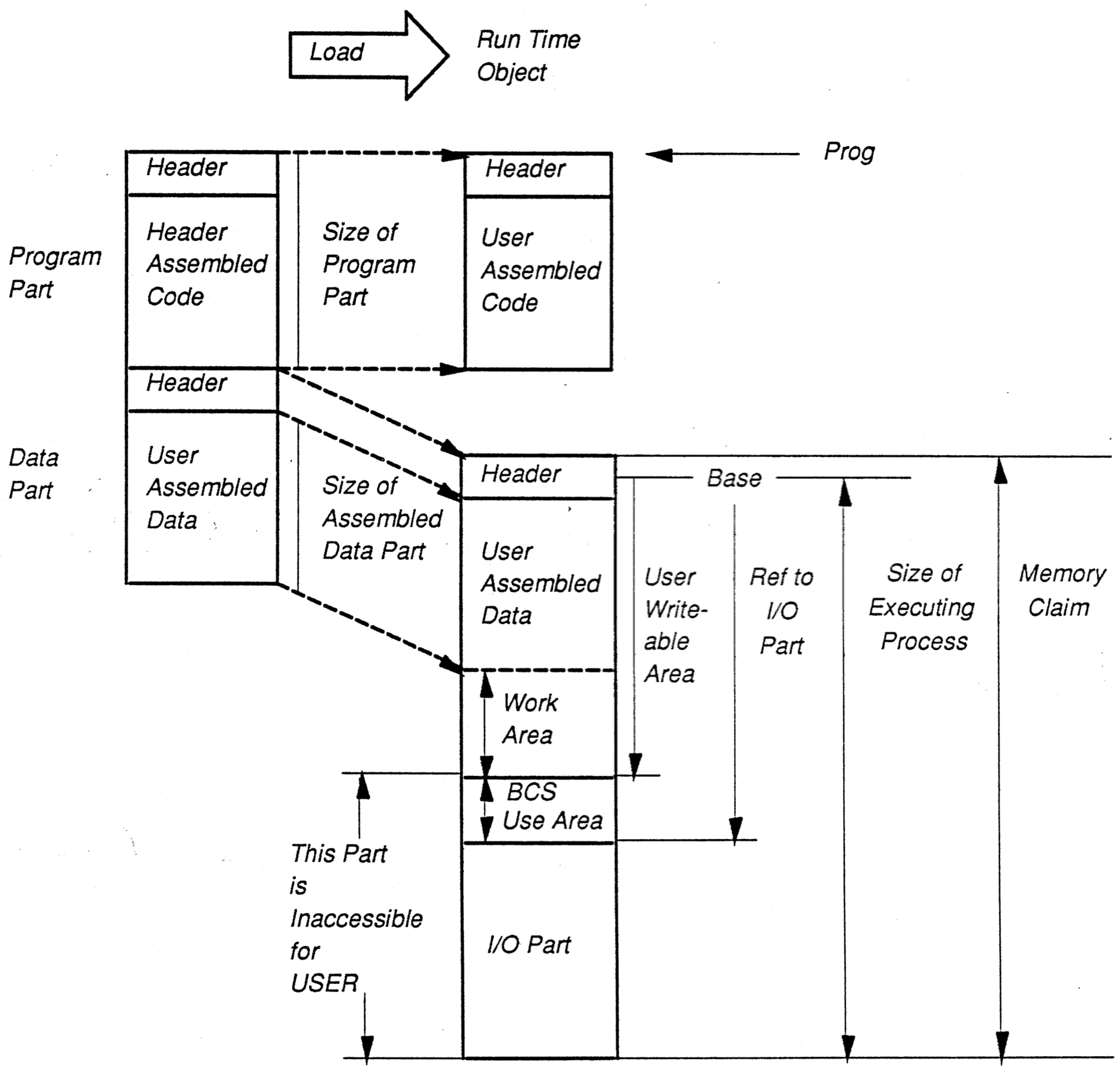
A significant advantage is that modules of all three programming languages can be linked together without problems.

CMR Processing Environment

The code and data parts of a program are separated, each part being referenced relative to its own base register. Addresses are primarily word addresses, thus giving a program direct access to 64K words of data and 64K words of code.

An important feature of the CMR processors environment is that the same program can be executed several times concurrently, using the same code, only allocating memory for the data part of each process.

The following figure shows the memory lay-out of the object code at assembly- and run-time:



TD3/NWPG/D/9

The CMR contains 8 general, 16-bit registers, used either individually for word operations, or concatenated for double word operations. The CR80 supports the following six primary modes:

- Register direct addressing
- Immediate addressing
- Code base addressing
- Database relative addressing
- Program counter relative addressing
- Register indirect addressing (indexing)

All instructions of the CR80 computer are single word instructions, using the primary addressing modes which may be extended by adding the contents of a 'modifier' register thereby extending the addressing range.

The following four data types are directly handled by the CR80 instruction set:

- Bits (single bit or sequence of bits)
- Bytes (8 bits)
- Words (16 bits)
- Double words (32 bits)

and the instructions include:

- Arithmetic and logical operations on words
- 'Move' operations on bits, bytes, words, and double words
- 'Skip' and 'jump' instructions, performing program sequence control

A CR80 instruction has as arguments zero, one or two operands. In case of logic and arithmetic operations, one of the source operands is used as destination operand. A number of instructions support memory to memory operations.

The Pascal Language

Pascal is a well-known programming language since it is used for educational purposes in many universities and technical academies. This means that it is easy to get well-qualified programmers accustomed to the language. When deciding on using the language in CR it was recognized that a standard Pascal package would not suit the needs of the company. Therefore an enhanced version was developed using the standard data and control structures, but omitting the I/O system since very little of our programming has to do with file access and manipulation. Instead a set of dedicated I/O routines compatible with the CR80 I/O system is supplied and described in the Pascal Prefix. The other differences between standard Pascal and CR80 Pascal are:

- A 'SIZE' operator, giving the size in words of the specified type of variable, is supporter.
- The underscore character is a letter.
- The 'File' type is not supported.
- The character " is used to begin and end comments.
- Labels cannot be declared or used.
- GOTO statements are not allowed.
- Nesting of Procedures and Functions not allowed.
- Procedures and Functions as parameters not allowed.
- UNIV type specifier added for relaxed type checking.
- 'Power of' operator introduced (**).
- Hexadecimal notation allowed.
- Comparison for equality between records allowed.
- Comparison for equality between arrays of all types allowed.
- Support the types LONG_INTEGER and LONG_REAL

Most programmers will not find it difficult to convert to CR80 Pascal if they have already programmed in Pascal or another high-level language.

Environment

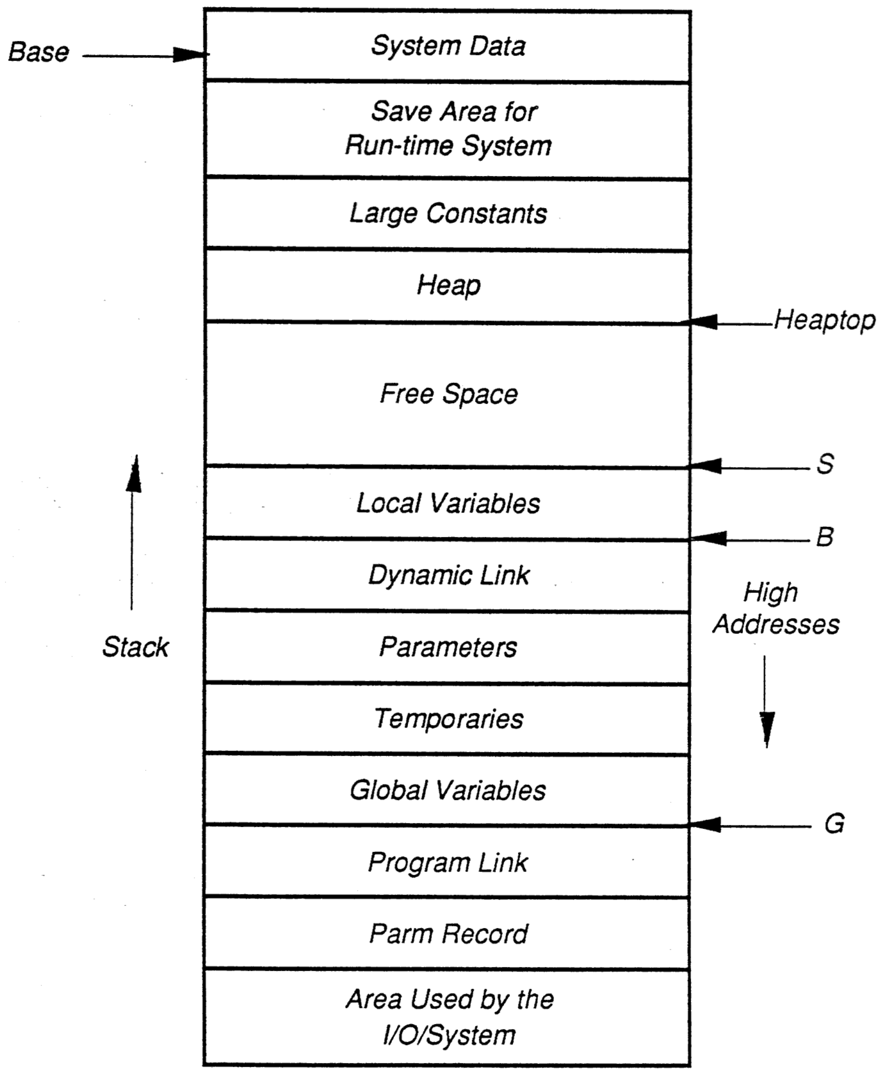
When a CR80 Pascal program is invoked from a terminal, the Command Interpreter (CMI) loads and starts the program and prepares a file with program information for the run-time system which has three functions:

- o Interface to MXAMOS Kernel procedures.
- o Supply procedures for special operations, e.i. Floating point operations.
- o Control stack and heap operations.

The first thing the run-time system does is to initialize a STACK used for variables, temporary results, parameters, and return information for procedures and functions, and a HEAP used for dynamically created variables.

Then the input and output files are opened, and finally the execution of program statements begins.

If we look at an executing CR80 Pascal program where a procedure has just been called from the main program, the data layout would look like this:



TD4/NWPG/D/5

Language An important element in all CR80 Pascal programs is the PREFIX which contains type and constant definitions, plus a list of more than 130 assembly-coded procedures and functions directly available to the programmer.

A CR80 Pascal program consists of two essential parts, a description of the actions to be performed, expressed in statements, and a description of the data manipulated by these actions, expressed in declarations and definitions.

With the differences described previously CR80 Pascal supports the standard Pascal set of types, operators and control statements.

With the restrictions mentioned previously CR80 Pascal supports Procedure and Function declarations known from standard Pascal.

A CR80 Pascal program consists of a Pascal standard prefix followed by the program block written by the programmer. The prefix supplies the compiler with the necessary information about the environment. It is therefore important to specify the correct prefix.

The following pages show a program example with basic elements of CR80 Pascal:

```
%SOURCE@**PREFIXES.D*MXAMOS.D*PASCAL.D*PREFIX.S
```

Comment This line includes the file containing the Pascal prefix in the source code of the program.

```
CONST
        PI = 3.14;
TYPE
        RANGE = 1..10;
        PTR = @PERSON;
```

Comment '@' is the pointer operator in CR80 Pascal.

```
        STR20 = ARRAY [1..20] OF CHAR;
        PERSON = RECORD
                            NAME, ADR : STR20;
                            SEX : (MM, FF);
                        END;
        TABLE = ARRAY[RANGE] OF INTEGER;
VAR
        LIST : TABLE;
        ROLL : ARRAY [1..100] OF PERSON;
        STUDENT : PERSON;
        CC, IS, OS, A, B, C, I, CTR : INTEGER;
        TAX : REAL;
        FLAG : BOOLEAN;
PROCEDURE SUMLIST (LIST : TABLE; VAR SUM: INTEGER);
        VAR
                I : INTEGER;
        BEGIN
                SUM := 0;
                FOR I := 1 TO 10 DO
                SUM := SUM + LIST[I];
        END;
FUNCTION EQUALITY (A, B, C,: INTEGER): BOOLEAN;
        BEGIN
                EQUALITY := (A = B) AND (A = C);
        END;
BEGIN
        CONNECT (PARAM.OFILE, OUTPUT_MODE, OS, CC);
        CONNECT (PARAM.IFILE, INOUT_MODE, IS, CC);
```

Comment

In the two statements above the variables OS and IS are assigned as file identifiers for standard output and input (screen and keyboard).

```
WHILE (CTR >= 100) AND (I <= 5) DO
BEGIN
        SUMLIST (LIST, A);
        CTR := CTR - 1;
        I := I + 1;
END;
IF NOT EQUALITY (A, B, C) OR FLAG THEN
      FLAG := FLAG AND (A <> B);
REPEAT
        OUTINTEGER (OS, A, #0320, CC);
        OUTTEXT (OS, 'IS SMALLEST', CC);
        FLUSH (OS, CC);
```

Comment

CR80 Pascal uses dedicated procedures for outputting variables. In OUTINTEGER a hexadecimal number is used to specify output format. The variable 'CC' is used to check the result of the procedure execution. The 'FLUSH' operation empties the output buffer on the specified file.

```
        A := A + 1;
UNTIL (A >= B) AND (A >= C);
END.
```

Compiler

The Pascal compiler is capable of translating Pascal program modules from source text into link modules. Time of compilation, number of compile errors, etc, is handed to the linker.

In addition, the compiler can generate intermediate code dumps which makes it possible to check the compiler operation after each pass.

A number of compiler directives may be included in the source code to control the compiler functions. All these directives may be controlled by conditions. In case the conditions are false, the directives are ignored.

The Pascal compilation consists of a pass driver (pass 0), 7 main passes, and a service pass (pass 8) producing debug information. The passes are executed in the same program area in memory, so the compiler is not re-entrant.

The C Language

The CR80 C compiler and standard library is based on a system developed by Whitesmith Ltd.

The CR80 C compiler and the associated standard library has tried to remain as faithful as possible to Whitesmith's language definitions, so that the CR80 C source code is portable to other computer systems running Whitesmith's compiler and visa versa.

C is a general-purpose programming language which features economy of expression, modern control flow and data structures (resembling Pascal), and a large set of operators. C is often described as a 'medium level' language; it is not a 'big' one, nor is it specialized to any particular area of application. Its absence of restrictions and its generality make it convenient and effective for a large range of tasks.

It is our experience that C code takes up less space and runs faster than the same programs written in Pascal. It does, however, require a rather high degree of experience before a programmer can write efficient C programs.

The definitive standard for C is appendix A in Kerninghan & Ritchie ' The C Programming Language'. The CR80 implementation is close to that standard, save for minor changes in emphasis. Here is a list of differences in the CR version:

- The major deviation is that the compiler requires each external declaration to be explicitly initialized exactly one among all the files that comprise a C program; the standard permits external declarations to remain uninitialized.
- The types unsigned [char, short, long] are allowed.
- Backslash is used to continue strings in the standard; here it is also used on command lines.
- Character constants with more than one character are defined here, but not in the standard.
- All struct and union tags share the same name space of all members of the struct or union in the standard; each kind of tag has its own name space here.
- Our implementation permits, as an option, separate name space for each struct or union and more rigorous checking of . and -> operators.
- A union may be initialized.
- A preprocessor macro invocation, e.g. swap (a, b), must be written all on one line.
- The size operator is explicitly disallowed in #if expression.
- Floating point operations are currently not supported.

Since our implementation produces assembler code for the target system, there is some variation in naming caused by assembler limitations. There may be as few as six, but never more than eight significant characters in external identifiers.

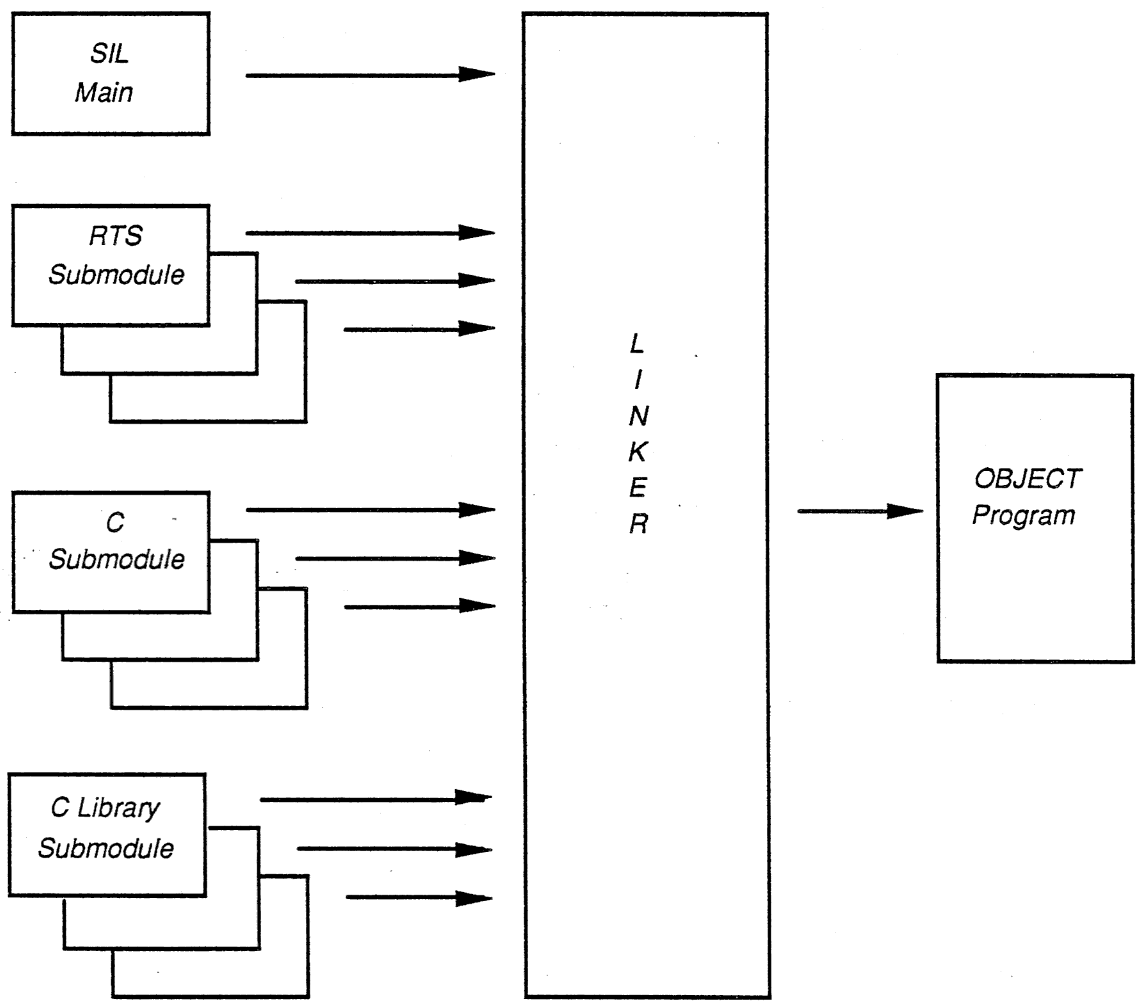
The CR80 C language implementation makes it possible for the experienced programmer to write fast, efficient code for the CRSN, with the added advantage that the programs will be portable to other computers, should the need arise, due to the fact that CR80 C is based on the Whitesmith Ltd. C system implemented on may computers.

Environment To enhance source code portability, the basic C system has access to a number of libraries and interfaces which shall be briefly described here.

- The CR80 C standard library: The standard functions available in the C language, for input and output operations, compare operations, type conversion, dynamic memory allocation, etc.
- The CR80 C run-time system: C run-time support functions and system interface functions.
- The CR80 C I/O interface prefix: An I/O prefix and interface functions for different kinds of I/O.
- The CR80 C kernel interface: A kernel prefix and a set of kernel interface functions.
- The CR80 C util library: Some utility functions for packing and unpacking data strings and buffers.

A C program consists of one or several C submodules, written by the programmer, a set of run-time system support submodules and a system interface main module called SIL.L. In addition to the C library, submodules may be included, but they are not mandatory.

This drawing shows the composition of a 'C' program:



Language

This section shows an example of a C program utilizing many of the syntactical elements and operators. The program is as close as possible to the example shown in the section describing the Pascal language.

```
#include <std.h>
#define range 10
typedef char str20[20]
typedef int table[range]
typedef struct
            {
            str20 name, adr;
            bool mm, ff;
            } person;
sumlist (list, sum)
table list;
int sum;
{
      int i;
      for (i=0; i<10; sum += list [i++]);
      return (sum);
}
bool equality (a, b, c)
int a, b, c;
{
      return ((a==b) && (a==c));
}
main()
{
      int a, b, c, i, ctr;
      person student, roll[100];
      table list;
      while ((ctr >=100) && (i <= 5))
      {
            b = sumlist (list, a);
            ctr--;
            i++;
      }
      if ! (equality(a,b,c,)) || (flag)
      flag = flag && (a != b);
      do
      {
            putfmt ("%i is smallest /n", a++);
      }
      while ((a < b) && (a < c))
}
```

Compiler

The CR80 C compiler is a set of several programs:

Input to the first program is one or more files of C source code. Output from the last of the programs is assembler code which performs the semantic intent of the source code. Source files may be separately compiled, then combined at link time to form an executable program, or C subroutines can be compiled for later inclusion with other programs.

When invoking the compiler, it is possible to specify certain options in the command line. At the moment the following options exist:

[D]	Produce line debug information
[S]	Produce symbolic debug information
[X]	Invoke cross-reference program
[CT]	Invoke the CTRACE test tool
[NR]	Do not remove temporary files

The C compiler consists of 3 main passes and assembly process.

Each of the compiler passes can be executed separately.

The SWELL Language

The Software Engineering Low Level Language (SWELL) is specially designed by CR for programming of high-performance modules on the CR80 computer. Although the language in its direct use of registers resembles assembly language, it also incorporates control and data structures only found in high-level programming languages, such as Pascal. Performance and resource requirements of programs written in SWELL are comparable with those for assembly language programs.

SWELL allows the programmer to program the CR80 computer in an efficient and well-structured manner. SWELL has adopted most of the constructs for structuring data and algorithms from Pascal. (See the previous chapter for a description of CR80 Pascal).

However, SWELL differs from Pascal on some essential points:

- o In SWELL the programmer must control register usage, addressing techniques, etc.

- o SWELL does not include any kind of run-time system. There are no run-time facilities for maintaining a variable stack or for parameter transfer. No general purpose registers or variables are allocated or used behind the programmer's back.

- o SWELL allows direct access to system software modules.

It should be noted here that SWELL is not altogether an easy programming language to learn and to use efficiently. However, programmers used to assembler programming will find it easy to convert to SWELL.

Environment As already mentioned in the previous section, SWELL does not include any kind of run-time facilities. This means that the necessary memory locations for process workspace and intermediate storing of variables and results must be declared as variables by the programmer, as part of the program. It also means that dynamic memory allocation, as known from Pascal or C is not possible in SWELL.

Language This section presents the language elements of SWELL. In the design of SWELL, the programming language Pascal was used as a model, due to the facilities Pascal offers for systematic and structured programming. Wherever practical, the language constructs are described using examples. The intention is to give an idea of the facilities of SWELL.

Constants and type definitions have been almost directly adopted from Pascal. However, in SWELL an arithmetic expression using constant values can be used to define constants.

The standard data types represents the units accessible in one instruction on the CR80:

o	BYTE	unsigned one byte integer.
o	CHAR	equivalent to byte.
o	INTEGER	depending on the operation performed either unsigned one word (16 bits) integer or signed integer using 2's complement arithmetic.
o	LONG	two word (32bits) integer.

Scalar types can be declared as in Pascal.

Array and record types may also be defined in SWELL.

Variables may be declared and initialized in SWELL. Space for all declared variables is allocated at compile time.

A distinct pointer type is not included in SWELL. Integers are used for this purpose, but to support the use of pointers, SWELL includes an ADDRESS operator returning the start address of a variable, and a SIZE operator giving the size of a data type or variable. From the information obtained using these functions it is possible to carry out pointer operations as known from Pascal.

The eight general purpose registers of the CR80 are pre-declared as integers with the name R0, R1 ... R7. They can also be used as double registers, capable of containing a LONG. The names are then R01, R12, ... R70.

Nearly all operations in SWELL require the use of one or more register variables.

In SWELL, no part of the data space is hidden from the programmer. Since arithmetic and logic instruction on a CR80 change one of the source operands, an expression involving variables is also an implicit assignment.

The expression:

```
R0 + 7
```

takes the value in R0, adds 7 and stores the result in R0. It should be noted that only operations directly supported by the computer's instruction set may be carried out, therefore adding two integer variables, I and J, looks like this:

```
I => R0 + J => I
```

Here RO is used as intermediate storage.

The corresponding statement in Pascal is

```
I: = I + J;
```

To show that all operations are carried out from left to right '=>' is used as assignment operator.

SWELL supports the following relational operators

=	is equal to
<>	is not equal to
>=	is greater than or equal to, signed
<	is less than, signed
>>=	is greater than or equal to, unsigned
<<	is less than, unsigned

SWELL also supports the logical operators:

NOT
LOGOR
LOGAND

In SWELL it is possible to interface directly to the operating system with the help of a monitor procedure called MON. The MON procedure is special in that it can take a varying number of parameters.

One of the parameters that must always be present in a procedure definition is the link register, specifying the return address after execution of the procedure.

Using the link register in a procedure is done through the EXIT statement as demonstrated in the program example later in this section. Function statements are not supported by SWELL.

In addition to the expressions and the procedure statements, the following kinds of statements exist in SWELL:

1) Compound statements

   A number of simple statements beginning with BEGIN and ending with END;

2) Conditional statements

   IF - THEN - ELSE and

   CASE - OF

   As in Pascal, the 'CASE' statement consists of an expression (the selector) and a list of statements, each being labelled by a constant.

   Example:

   ```
   CASE R0 OF
   sunday : GOTO church;
   saturday : GOTO shops;
   OTHERWISE GOTO work;
   ```

3) Iteration statements

   WHILE - DO

   REPEAT - UNTIL

4) GOTO statement performs an unconditional jump to a label.

5) Dedicated statement. Executes one specific CR80 machine instruction. Similar to a procedure statement, but without the link register. Examples on dedicated statements are the 'STC' and 'UNS' statements in the following program.

On the following pages an example on a SWELL program is shown:

```
MAINMODULE EXERCISE;
%SOURCE @**GENERALPARAMS.S
%SOURCE @**MONITORNAMES.S
%SOURCE @**IOSPARMS.S
%SOURCE @**UTH.I
CONST
      TEXT = 'IS SMALLEST (:0:)';

TYPE
      RANGE = 1..10;
      PERSON_PTR = INTEGER;
      STR20 = ARRAY[1..20] OF CHAR;
      PERSON = RECORD
                      NAME, ADR : STR20;
                      SEX : (MM, FF)
                      END;
      BOOLEAN = (FALSE, TRUE);
      TABLE = ARRAY[RANGE] OF INTEGER;
VAR
      LIST : TABLE
      WORKAREA : ARRAY[1..6] OF INTEGER;
```

Comment Since there is no stack operation, workarea for procedures must be reserved by the programmer.

```
      ROLL : ARRAY[1..100] OF PERSON;
      SUM, A, B, C, I, CTR ; INTEGER;
      FLAG, FLAG2 : BOOLEAN;
INIT
```

Comment In SWELL it is possible to initialize variables to an initial value.

```
      CTR = 105;
      I = 0;
PROCEDURE SUMLIST (R6);
VAR
      REG_STACK : ARRAY[0..7] OF INTEGER;
BEGIN
      R7 => REG_STACK[7];
      STC(6, ADDRESS(REG_STACK[7]) => R7);
      0 => R1 => R2 => SUM;
```

A 'SIZE' operator, giving the size in words of the specified type or variable, is supporter.

```
    WHILE R2 <= R5 DO
        LIST[R2+1] => R1 + SUM => R4 => SUM;
    UNS (7, ADDRESS(REG_STACK) => R7;
    EXIT (R6);
END;
PROCEDURE EQUALITY (R6);
VAR
    REG_STACK : ARRAY[0..7] OF INTEGER;
BEGIN
    R7 => REG_STACK[7];
    STC(6, ADDRESS(REG_STACK[7]) => R7);
    ((A=>R1 = B=>R2) LOGAND (R1 = C=>R3)) => R5 =>FLAG;
    UNS(7, ADDRESS(REG_STACK) => R7);
    EXIT (R6);
```

Comment The 'procedures' STC and UNS are in reality instructions from the CR80 instruction set. They save and restore the contents of the 8 register variables.

```
END;
BEGIN
    ACCEPTFILES(R6);
    READSYSPARAMS(R6);
```

Comment The two procedure calls above ensure that current output file (the screen) and the parameter file (with user input) is opened.

```
    DISMANTLEFILE(R4, R6);
    OPENSTREAM(ADDRESS(CINFILETYPE) => R4, INPUT_MODE => R3, R6);
```

Comment The two procedure calls above close the parameter file and reuse the file description to open the current input file (the keyboard).

```
    WHILE (CTR=>R0 >= 100=>R1) LOGAND (I=>R2 < 6=>R3) DO
    BEGIN
        SUMLIST(R6);
        CTR => R0 - 1 => CTR;
        I => R2+1 => I;
    END;
    IF NOT (FLAG=>R4 = TRUE=>R6) LOGOR (FLAG2=>R5 = R6)
        THEN ((FLAG2=>R5) LOGAND
REPEAT
      ADDRESS(WORKAREA) => R0;
      A => R2;
      #0720 => R3;
      COUTFILETYPE.S=> R4
      MON(STEAM, OUTINTEGER, R0, R2, R3, R4, R7):
      ERROR_DONE;
      ADDRESS(TEXT)=> R6;
      MON(STEAM, OUTTEXTB, R6,R4,R7): ERROR_DONE;
      A=>R2+1=>A;
UNTIL (R2>=B=>R3)LOGAND(R2>=C=>R3);
            CLOSESTREAM(ADDRESS(CINFILETYPE)=>R4, R6);
            CLOSESTREAM(ADDRESS(COUTFILETYPE)=>R4,R6);
            DISMANTLEFILE(R4, R6);
            MON(TERMINATE,0=>R0=>R1,R7);
END
ENDMODULE
```

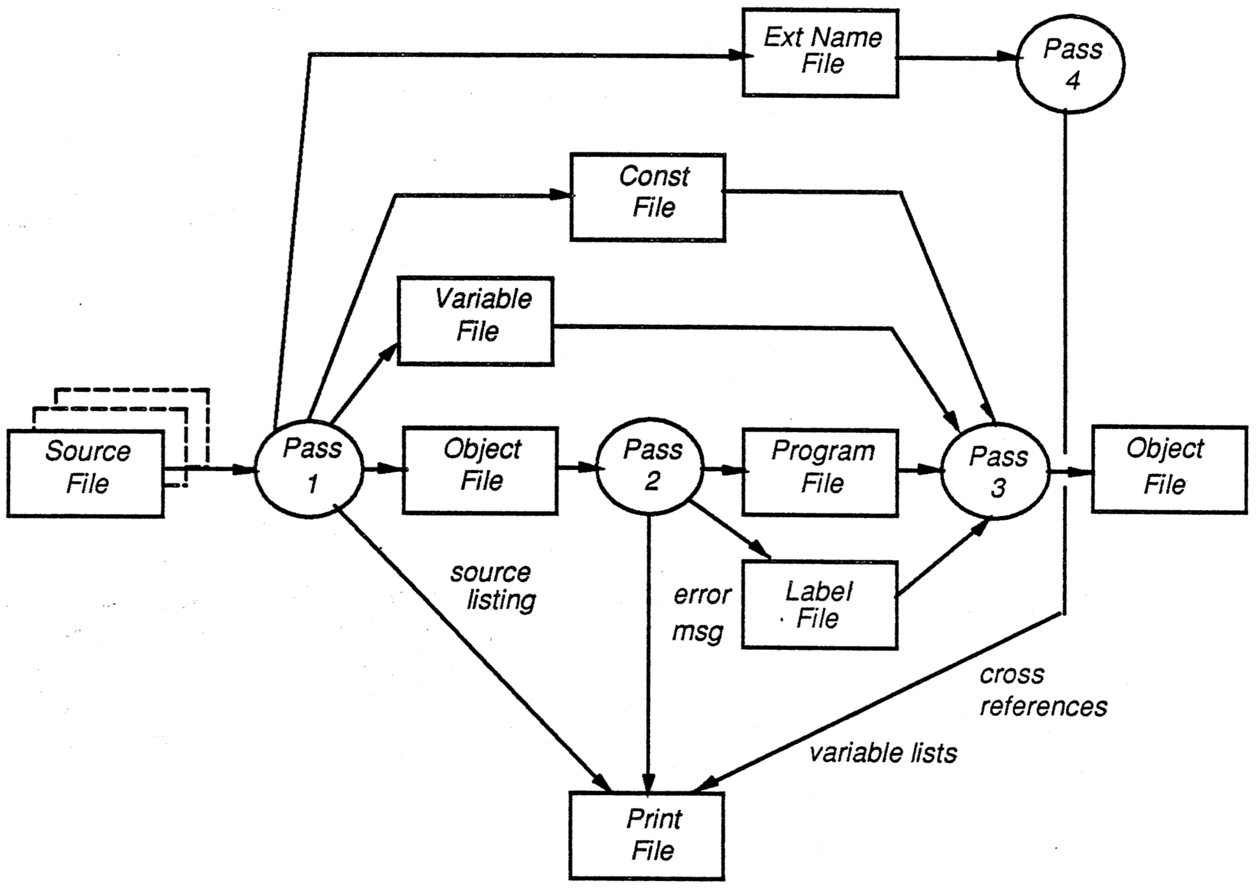
Compiler

The SWELL compilation process consists of five passes, executed in the same program area in memory. This has the advantage that the compiler process takes up less space, thereby allowing several compilation processes at the same time. It also means that the compiler is not reentrant.

The user can choose from a set of options, among which are:

P:	Generates a print file for the source listing.
L:	Determines the amount of information in the source listing.
X:	Controls cross-reference generation.
D:	Controls inclusion of symbolic debug information.
PL:	Sets the number of lines per page in the print file.

The compilation process is shown in the following figure:



NWPG/D/8

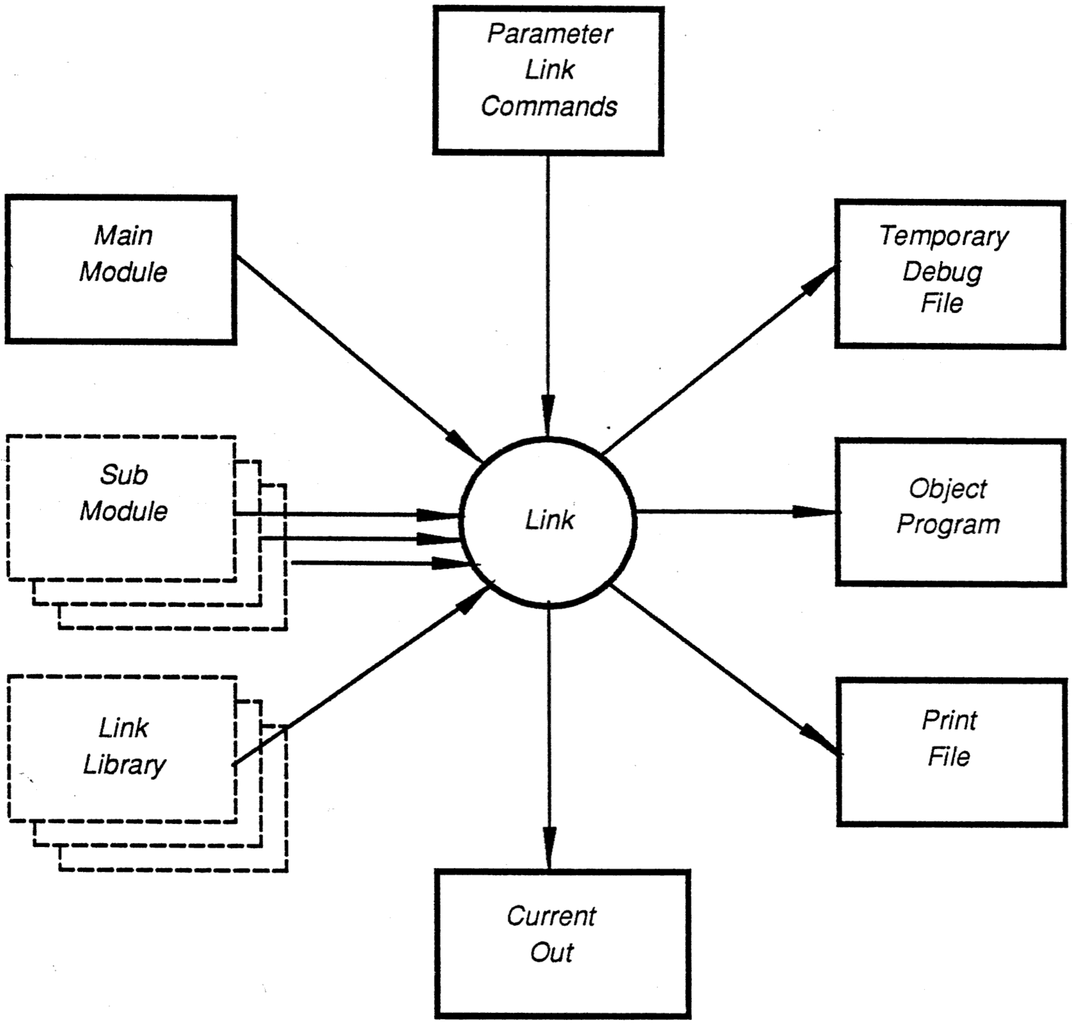
2.3 CR80 Linker

The linker is capable of transforming a number of link modules, one which is defined as the main module, to a single executable program.

The linking process includes module linking, final address resolution within a module and code assembly. The linker may optionally print debug information like assembled code, initialized variables, and program and process headers. The contents of the headers are composed of information from the link modules and linker activation parameters (program options).

The linker is common for all CR80 programming languages, and links together program modules written in one or several languages.

The linking process may be visualized like this:



TD3/NWPG/D/7

The linking is performed by concatenating the link modules and by exchanging addresses according to directive in the program modules.

Link libraries consist of precompiled functions and procedures, i.e. for handling arithmetic operators.

First, all submodules are handled. The linker then checks that all addresses referring to other modules are correct. During the linking, all addresses are checked and final object code is assembled. The linker also generates program and process headers, composed of information from the link modules and program options.

Optionally, the linker may generate object code listings.

2.4 Test Tools

All programs require testing, and the more complex the program the more advanced the test tools must be. When developing software for a very fault tolerant environment such as the CRSN, advanced test tools are crucial.

Three different debug tools are available; the first two apply only during the development of the code. The third is part of the maintenance and diagnostic system in the CRSN and is used during development as well:

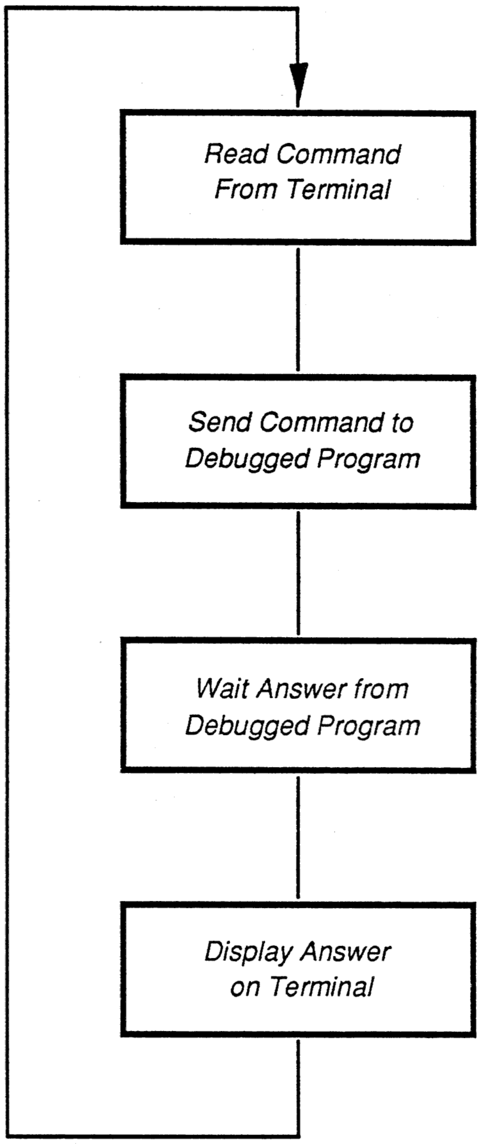
- MX Debugger
  Interactive test tool - used during development for small programs or modules

- Test Command Interpreter (TCI)
  Interactive test tool - used during development for processes

- Cyclic Debugger
  Test tool integrated within your program - used both during development and during maintenance

MX Debugger

This test tool is especially designed for test of small programs or modules from a big subsystem. Any module can be tested separately. The MX Debugger is a test tool which allows the developer to debug a program, provided the program has been compiled and linked with the debug option. MX Debugger is an interactive program which alternates between command reading and command execution as shown in fig. 2.5-1:


Read Command From Terminal
Send Command to Debugged Program
Wait Answer from Debugged Program
Display Answer on Terminal

TD3NWPG/D/13

Fig. 2.5-1 MX Debugger Control Flow.

The most important commands used to control the debugging of the program are:

- o Single step the program
- o Dump variables (symbolic)
- o Alter variables
- o Display registers.

When using this test tool, additional debug information is incorporated into the code by the compiler, allowing the MX Debugger to control the execution of the program under test.

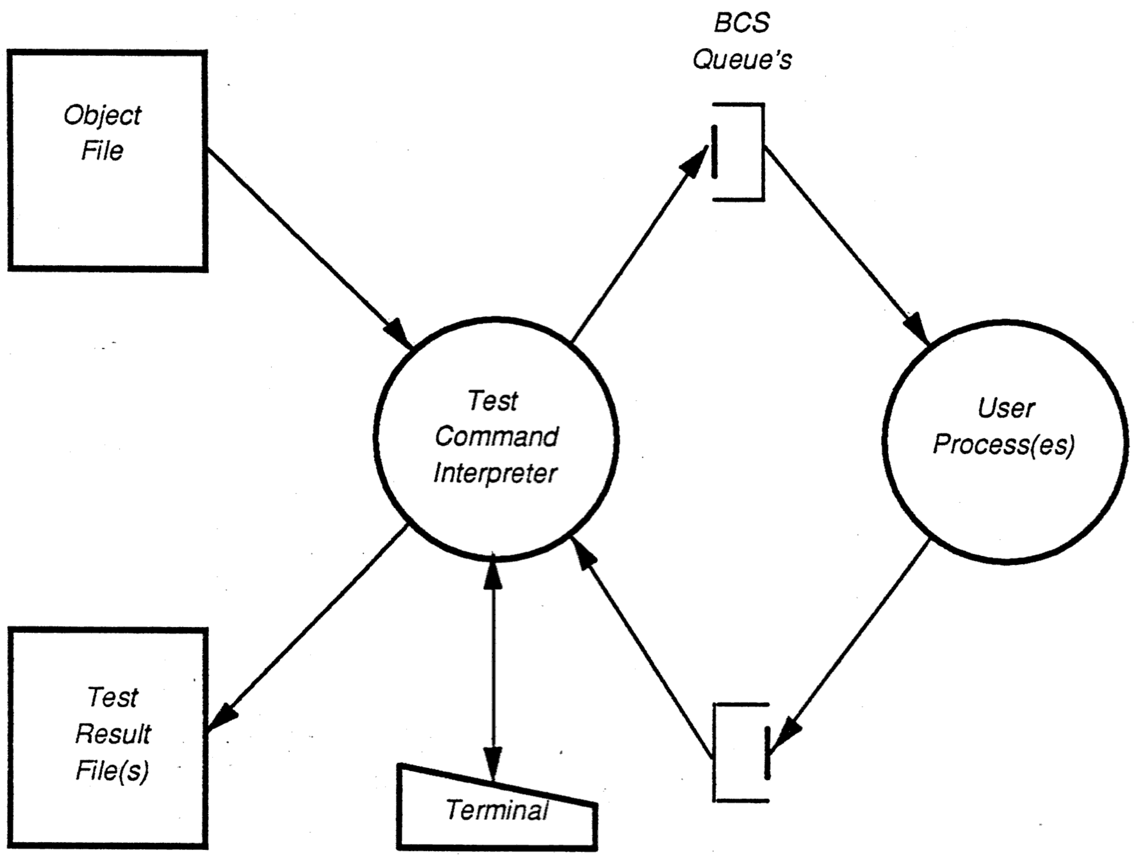
Test Command Interpreter (TCI)

The main purpose of this facility is to simulate other external environment with which the program under test is interfacing. The processes in a CRSN environment communicating by means of the basic entity - the message - are of special interest. Any program that must communicate uses the Basic Communication Service (BCS).

The CR80 Test Command Interpreter interprets a language 'Test Command Lanugage (TCL)', which is a high level language having a syntax very much like the Pascal Language.

When using TCI the test sequence can be written in a high level language and the result of the test is formatted printouts of how the program responds to messages in all situations. The TCI normally executes the test in sequence , but another advantage of the TCI is that it supports interactive test of a program. That is; any procedure written in Test Command Language can be invoked from a terminal at any time while testing.

The Test Command Interpreter is a two pass compiler where pass one delivers an object file as input to pass two. During pass two the actual test of the user process is performed as shown in fig. 2.5-2. The object file from pass one of the TCI can be used to repeat a certain test a number of times.



TD3NWPG/D/14

Fig. 2.5-2 Test Command Interpreter (pass two)

An especially powerful capability of this tool is to drive the process under test into extreme situations where error handling routines in the developed program must interact and recover. Certain error situations seldom occur in a live network, but when they do, some implemented code that has never been tested starts to execute and results in fatal error. Therefore, when using the TCI to simulate an external environment one of its most important tasks is to test if the program can recover from various error situations.

Any number of test programs written in TCI can be activated simultaneously and thereby simulate several processes to which the process under test has interface. Usually the TCI would be used as simultators for SC, NM and TN subsystems.

Cyclic Debugger

The purpose of the Cyclic Debugger is twofold. Originally it was intended to serve as a part of the maintenance and diagnostic tool integrated within the CRSN, but considering the nature of this test tool it is clear that it has a major function during the coding phase of any CRSN product.

The Cyclic Debugger is a utility which can dump trace information and internal variables from a program. As additional coding the programmer must include a number of procedure calls within the program to perform the actual dump of the variables that need to be verified. The name 'Cyclic Debugger' refers to the fact that the buffer that holds the latest debug information operates in a cyclic manner. The program code must internally allocate memory for the cyclic buffer.

The amount of information to be logged depends on the debugging mask which is handed to the process as a part of the process parameters received in the initialization phase. The mask contains the following bits which can be set independently:

o	Trace bad completion codes. Whenever a result of an external or internal operation is not ok this should be logged.
1	Trace input/output from the process. All BCS messages received or sent by the process must be logged.
2	Trace all procedure calls. All procedures within the process must generate a tracepoint when called.
3	Trace all operation within procedures. All operations which may result in internal error handling and therefore the result must be logged.
4-13	Module trace bits (module 1/10). All modules within the process must be assigned to one of 10 classes (levels). Only the procedure belonging to a class which is selected in the module trace mask should produce tracepoints.

14 Timestamp flag. When set, each tracepoint will be timestamped be the Cyclic Debugger interface procedure (CDEBUG).

15 Log trace on disc file. The trace procedure should log all tracepoints on a disc file.

The actual logging is performed by calling the procedure CDEBUG.

It is obvious that this debug interface will slow down any system, but the masking information used ensures that only the modules which are under test need be logged. Generally, the mask will be set so that no modules in a program will be logged and only during a maintenance phase or when coding the mask is set to log large amount of information to a file.

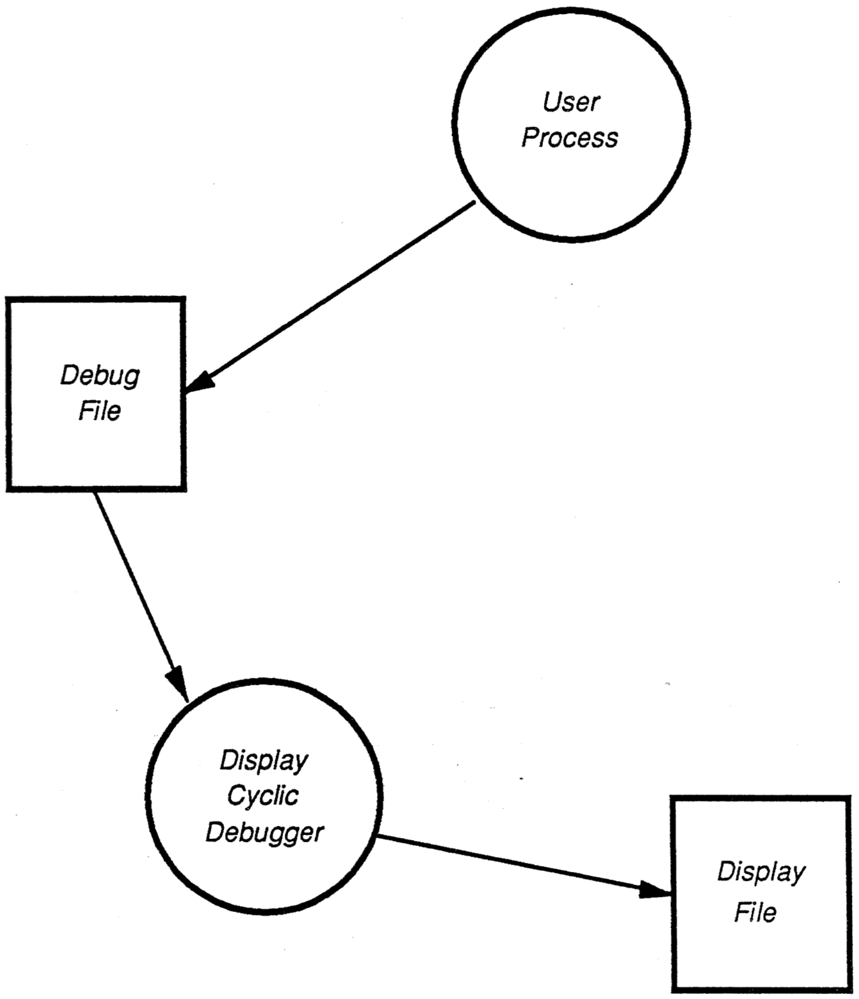
In CRSN

When programs are running in a CRSN environment this cyclic buffer can at any time be dumped to a disk file in order to be inspected during maintenance of the program. As a Normal Operator interface it is possible to change the debug mask so that different modules and levels can be logged.

During Development

During the Coding Phase of the program the Cyclic Debugger is used with the file option so that all debug information goes directly to a file in order to get a very good history log of how the program has performed while running.

A utility to display the Cyclic debug buffer is available and is called the 'Display Cyclic Debug Buffer' (DCDB), see fig. 2.5-3. The DCDB takes the debug file from the program under test as input, and when displaying this file it is a possible to select which debug information to display. This allows the programmer to select the module or procedure that has errors.



TD3NWPG/D/15

Fig. 2.5-3 Debug Information Flow.

3 THE RH TEMPLATE

CHAPTER CONTENTS PAGE

3 THE RH TEMPLATE

When a user wants to develop new application software, e.g. an Access Resource Handler (ARH) for the Corporate Resource Sharing Network (CRSN), the user must secure that the new ARH fulfils the requirements stated by the CRSN architecture.

To ease development of new ARHs, which fulfils a number of the CRSN requirements, a software packet has been developed which can be reused from one ARH to another. This software packet is called the RH Template and it will implement many of the functions common to most ARHs in CRSN.

This chapter will introduce the RH Template and outline the functionality available. But before describing the functionality the most common level 1 requirements stated against an ARH and an abstract of the processing phases, which a typical ARH will go through, will be given, too.

Template

An ARH is a network element connecting external equipment to CRSN. As an ARH is an integrated part of the CRSN it must fulfil some requirements given by the architectural rules outlined in the first chapter of this document.

The architectural rules state three levels of requirements. The minimum set of requirements an ARH must fulfil is the level 1. Level 1 concerns the interfaces to:

- Network Management
- System Control
- Transport Network

The level 1 requirements means that all ARHs in the CRSN will contain almost identical software; in addition, each ARH has special modules to implement its specific interface towards the external equipment.

By level 2 requirements means the requirements that ARHs must fulfil in a peer to peer protocol.

The level 3 requirements state the requirements for standardized components of an ARH. The fact that all ARHs contain similar software because they interface to the same network elements, creates a basis for development of standardized components and thereby define a number of level 3 requirements.

A number of standardized components have been developed, and together they constitute the RH Template. The components or modules within the RH Template implement a number of level 1 interfaces, hence hiding them for an application ARH. The term application ARH will be used when referring to the part of an ARH which is not implemented by the RH Template.

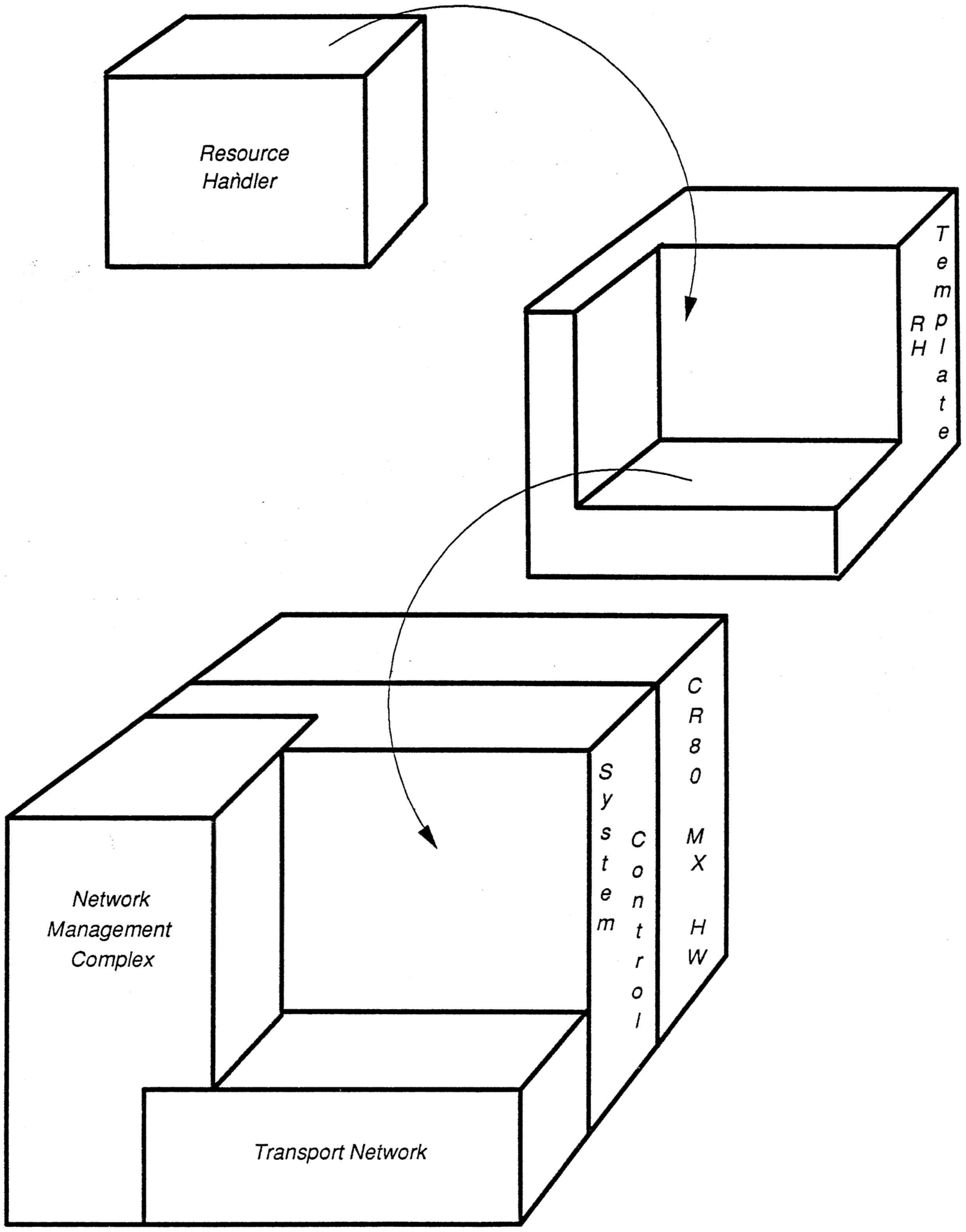
The RH Template is intended for reuse when designing new ARHs and by reusing the RH Template a number of benefits are gained:

- o Shorter development time of ARHs fulfilling the CRSN requirements.
- o Effort can be concentrated on designing the application part of an ARH.
- o Higher reliability of the ARH from the start because reuse of well-tested and documented software.

Basically the RH Template is a collection of software modules which can be reused as building blocks to form a level 1 compatible application ARH. The RH Template cannot be used as a stand alone ARH - the application ARH must implement some functionalities in addition to the ones given in the RH Template to form a complete ARH. This is especially the software which implements the interface towards the external equipment and/or ancillary processes.

The modules of the RH Template will implement many of the level 1 interfaces towards NM, SC and TN, demanding a minimum of interaction from the using application ARH. Furthermore, a number of modules will implement some utilities which makes life easier for an application ARH developer.

Fig. 3-1 gives an overview of an application ARH using the RH Template to fulfil the level 1 interfaces:



TD4/NWPG/D/22

Fig. 3-1 CRSN building blocks using RH Template.

Before the functionality offered by the RH Template is described a short list of the most common level 1 interfaces will be given, and a short abstract of the typical processing phases an ARH will go through will be given, too, to ease understanding of the RH Template functions.

3.1 Abstract of Common Level 1 Interfaces

This section will give a short description of the most common level 1 interface and the functionality which can be reached via the interfaces.

Network Management

In the CRSN a network operator can monitor and control all ARHs including their resources. To do this NM requires a number of interfaces to be serviced by the ARHs. Furthermore, NM offers a number of services from which the ARH can obtain vital information. The most important ones are:

- o Configuration

  The configuration is a description of ARHs resources and their interrelation. This is done by means of a number of resource records. The configuration can be updated with new resource records and records can be deleted.

- o Resource Control

  Resource control allows an operator to include/exclude resources in the on-line network. Excluded means that the resource is not served by the network. When the resource is included data can be sent to or from it.

- o Status

  All resource records contain a status field where status of the resource is maintained by means of a number of standard states. An operator can monitor the status of the resources via the status interface.

- o Events

  Events are used to notify the network operator about the current status of resources and detection of severe failures in an ARH. Events are sent unsolicited from the ARH.

- Statistics

  Statistics are used to count the number of different occurrences, e.g. number of messages received by a resource. This will give an impression of the network in the past. Two types of statistic are defined: permanent and temporary.

- Trace

  The trace interface allows the network to monitor data traffic from/to a resource.

Transport Network

TN is used to get data transported between the Nodes in CRSN. To do this TN offers 2 services: the messagegram service and the connection service.

- Connection service

  The connection service allows a user to obtain a fixed message flow between two partners. The service is pro tected meaning that messages sent via a connection is secured to be delivered to the other end in the sequence they were sent. The service is in popular terms a telephone call.

- Messagegram service

  This service is an unprotected service which means that TN does not secure that a message sent is delivered to the receiver or that messages are delivered in the sequence sent. The service is in popular terms a mail box.

Before one of the services can be used the user must obtain an address from TN via an enrol interface.

System Control

System Control will interact with the ARH in the initialization phase and it will handle the switch between the active ARH and the standby ARH. Furthermore, SC can change the cyclic debugging mask in an ARH at any time.

3.2 Access Resource Handler Processing Phases

During the lifetime of an ARH it will be in one of a number of processing phases where some specific actions have to take place. A typical ARH will go through some of the following processing phases:

- o Process Initialization Phase
- o ARH Configuration Phase
- o Standby Processing Phase
- o Active Processing Phase

Fig. 3.2-2 shows the different phases and their logical interrelation:

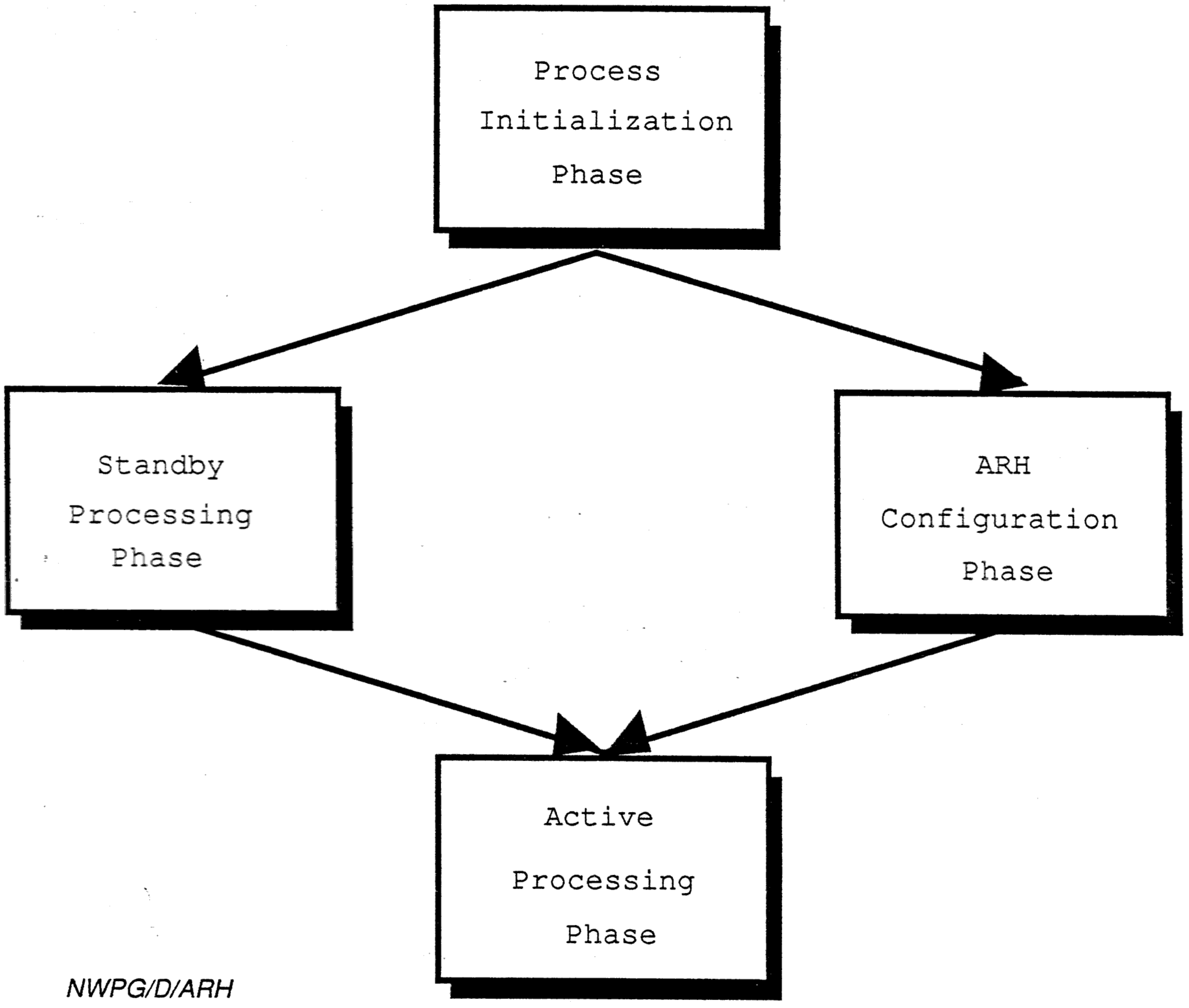


Fig. 3.2-2 Major ARH processing phases.

1. Process state equal to 'active' received.
2. ARH included in network by NM
3. Process state equal to 'standby' received.
4. Request to go active received from SC.

Process Initialization Phase

The first phase all ARHs must go through is the process initialization. In this phase the process exchanges system parameters with SC and this is, among other things, the queue names of the control queues owned by NM, SC and TN. This enables the ARH to get in touch with NM, SC and TN. As the last thing in this phase the ARH will receive a process state from SC. The process state can either be 'active' or 'standby'.

ARH Configuration Phase

If the process state received was 'active' the ARH must enter the ARH configuration phase. In this phase the ARH must make itself known to NM and request and receive its configuration. The ARH must be in this processing phase until NM requests the ARH to enter the next processing phase, the active processing phase.

Active Processing Phase

The ARH must be in this phase the rest of its lifetime. Here the actual processing of the ARH is performed - data is switched from the external equipment to another ARH and the level 1 and level 2 interfaces (see chapter 1) are serviced.

Standby Processing Phase

If the process state received in the process initialization was 'standby' the ARH must enter the standby processing phase. This ARH is now a standby process receiving vital data from the active ARH which is identical to the standby ARH it has just received a process state equal to 'active' instead. The vital data received are the configuration and checkpoints.

Checkpoint is e.g. updates to the configuration (new resources inserted or a resource is deleted) and states changes in a resource (e.g. from on-line to off-line). The checkpoint enables the standby ARH to take over in case a fatal error is discovered in the active ARH. If such an error is discovered SC will terminate the active ARH and request the standby process to go to the active processing phase.

3.3 Functions in the RH Template

This section will outline some of the functions available in the RH Template.

The RH Template will form a complete ARH shell, meaning that it will implement the main waiting point and the dispatching of control in addition to the level 1 functionalities. The main waiting point is where all external events are received by the ARH. External events can be that a message is received or a BCS queue has been defined, etc. If the event is a message received event, dispatching of control to the prober function within the RH Template or the application will be done.

The following functionalities are available in the RH Template:

- o Process Initialization
- o Handling of SC interface
- o Handling of TN interface
- o Handling of NM interfaces
  - - ARH initial configuration reception
  - - Standardized handling of configuration
  - - Event generation
  - - Trace facility
  - - Status and statistic reporting
- o Handling of standby processing

Furthermore, a number of utilities are implemented, e.g. timer services, monitoring of queues, etc.

Process Initialization

The RH Template will handle the entire process initialization phase; no interaction from the application part is needed. The RH Template will create the input queues for NM and SC, where messages are received from the two network elements throughout the life time of an ARH. The RH Template will exchange system information with SC in order to make itself known to SC and BCS and to obtain a process state. Depending on the process state the RH Template will enter the active or standby processing phase.

Handling of SC Interface

All interfaces towards SC are implemented entirely within the RH Template. The user ARH does not have to worry about this. When e.g. an indication for initialization of the cyclic debug buffer is received the RH Template will handle this without any interaction from the application ARH.

Handling of TN Interfaces

The RH Template has implemented two state event machines to handle the enrol and connection interface towards TN. The enrol interface allows an ARH to obtain an address in TN and to use the messagegram service. When an application ARH wants e.g. to establish a connection it can be initiated by a single function call. The connection state event machine will take care of the rest of the connection set up procedure transparently and concurrently with other tasks in the application ARH.

The same is true for the enrol state event machine.

A state event machine is a software submodule which takes actions upon occurrences of events. A state event machine is built upon a state event table which tells how the state event machine must act upon occurence of an event. An event can be e.g. that a message has been received or a a timer has expired. The action taken depends on the state where the state event machine resides at the time where the event occurs. With other words, a state event table defines how the state event machine must act in all situations which have been foreseen.

The application ARH does not have to interact with the TN interfaces. The RH Template will take care of this.

Handling of NM Interfaces

If the process state received in process initalization was 'active' the ARH will enter the ARH configuration phase. Most of the functionality needed in this phase is implemented by the RH Template. The RH Template will request and receive its configuration from NM. The configuration is stored in a configuration tree which allows dynamically insertion or deletion of resource records in the on-line network. A number of utility functions are available for configuration updates and searching in the configuration tree. The RH Template will await further updates to the configuration and NM indication which informs the ARH that it can enter the active processing phase.

The ARH configuration phase will entirely be handled by the RH Template except for the unpacking of the configuration records. The outlook of the individual configuration records differs from ARH to ARH so no general functionality is implemented.

After the ARH configuration phase the ARH will enter the active processing phase where the actual processing of the ARH is performed; the ARH is now considered active and on-line in the network and it must start serving all its resources. A number of state event machines are implemented by the RH Template in order to support the active processing required by NM of the ARH. The state event machines are:

- Status Reporting state event machine
- Statistic Collection state event machine
- Event Generation state event machine
- Trace state event machine

Common to all state event machines is that they fulfil the level 1 interfaces in a standardized way outlined in the Level 1 Architectural Specification. The RH Template will control the state event machines in a way which demands a minimum of interaction from the application part of the ARH. Only in the case where the RH Template does not have the needed information an application interface function is invoked. This is the case when e.g. statistic for a resource is maintained in an ancillary process. The application interface function must fetch the statistic and pass it on to the statistic collection state event machine. The evaluation of the request and generation of the response is handled entirely within the individual state event machines.

Before using one of the state event machines some application interface functions must be edited to fulfil the need of information for the state event machine. In the above example concerning statistic collection the interface function must implement the interface to the ancillary process.

When an ARH communicates with NM a standard request/response protocol must be followed. The protocol requires that a request to NM is issued a number of times with a time delay in between if NM does not confirm the request within the time period. Furthermore, an ARH must recognize duplicate requests sent from NM. This is handled transparently to the application ARH by the RH Template.

Handling of Standby Processing

If the ARH is standby a number of specific actions must take place, e.g. the configuration must be transferred from the active ARH.

The RH Template has implemented an active/standby interface with associated functionality which will handle most of the standby processing transparently to the user. This is, among other things, copying of the configuration to the standby ARH handling of the standby queues, updates to the configuration already received and reception of the SC command, indicating that the standby ARH must go active.

In the active ARH a checkpoint state event machine is implemented which will handle transference of checkpoints to the standby ARH.

Utilities

A number of utilities which implement a high level interface towards the Basic Communication Service (BCS) is available, too. These utilities allow, among other things, to create, write and send a message to a certain destination with only one function call, to start a timer in order to prevent a deadlock for an outstanding request, to handle loss of communication queues transparently, etc.

4 PROCEDURES AND PRACTICES

This chapter describes the development procedures and practices on a basis so that a developer could start his writing of an application and be guided through the process of creating software for the CR80 computer and especially for the Corporate Resource Sharing Network (CRSN) Subnode or the CRSN network application.

CHAPTER CONTENTS **PAGE**

4 PROCEDURES AND PRACTICES

In this chapter the procedures and practices for designing and developing of software to the CRSN are summarized. The description follows the practices as used by the CR developers in terms of:

- o Design
- o Development
- o Testing and Network Integration

4.1 Design

This section outlines the practices used by the CR developers when designing software for the CR80 computer or the CRSN.

When a programmer has made an overall design of any application there are some vital things to consider when programming for the CRSN. These things are listed below:

- o **RH Template** - Which Level 1 functionalities should be supported? Can the RH Template be re-used?
- o **Database Specification** - What should be specified as database entries for a particular subsystem?
- o **Program Language** - Which language is suitable?
- o **Testing and Network Integration** - Which test tools should be used for this specific application?

The design phase of any CR product provides guide lines from a major design overview down to detailed design; the latter may even provide some pseudo coding as input to the actual coding phase.

The design incorporates a modular application structure (and the actual CR80 code reflects this) as modularization is supported both by compilers and the linker. The modularization is often such that each application function is handled totally or partly by a distinct module (e.g. status collection handling).

To use these procedures is not mandatory when programming for the CR80 computer, but as the development tools of the CR80 support modular design it is useful to use a design method that implements modular design.

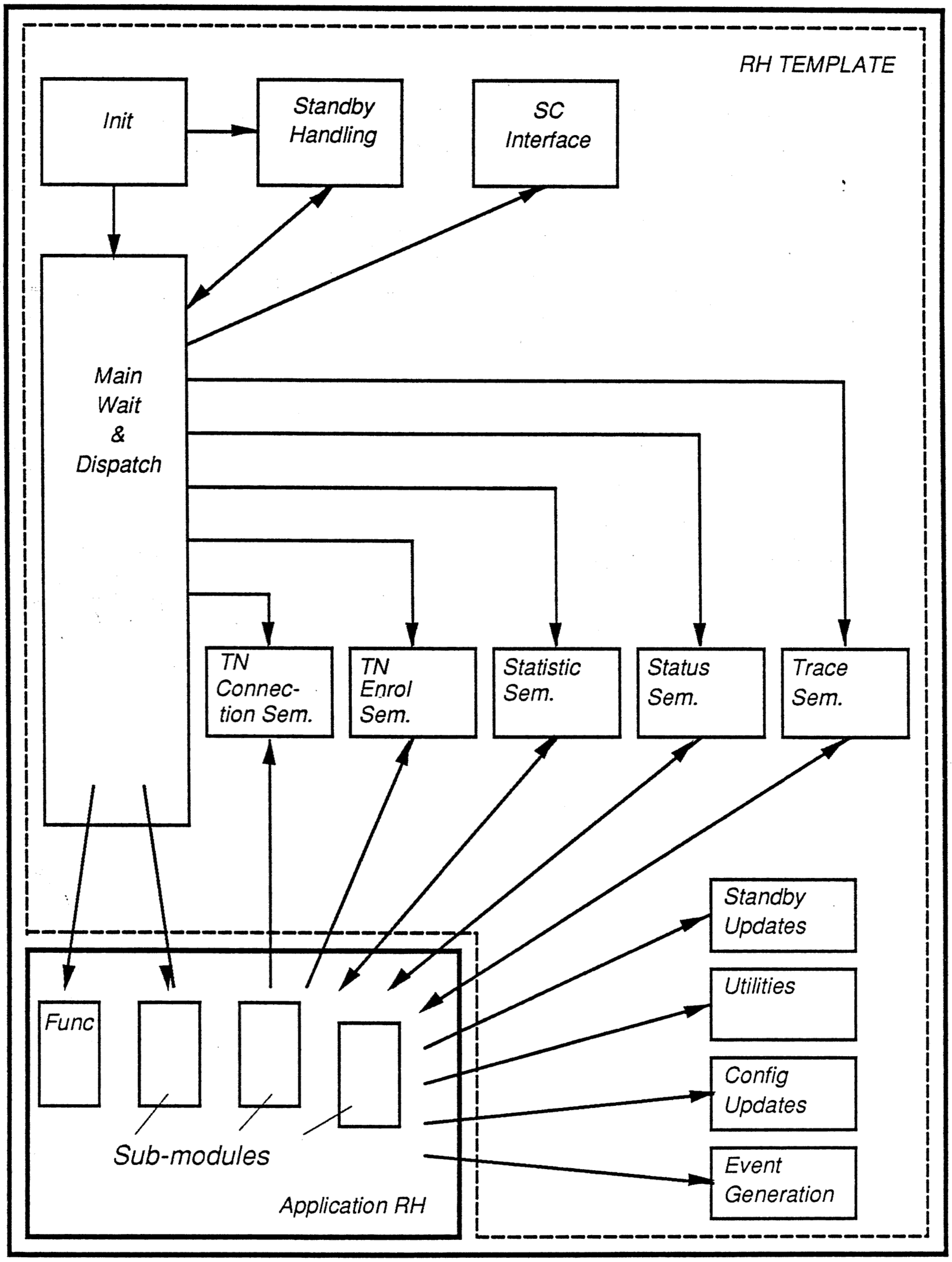
4.2 Development

This section describes the design and development procedures and practices as used by CR engineers.

RH Template

The following paragraphs briefly explain how a developer can use the RH Template to construct a real CRSN Access Resource Handler (ARH). But first is outlined how the RH Template sees the application software.

As the RH Template contains the main waiting point and the ability to dispatch control to both the RH Template and the application ARH, the application software is seen as a number of functions the RH Template can call. In return, the RH Template offers a number of functions to implement the desired service. Fig. 4.2.1 shows the internal build-up of a complete ARH, including both RH Template and application ARH software.



TD3NWPG/D/23

Fig. 4.2-1 Internal build-up of ARH using RHT.

When a developer wants to build a new application ARH which fits into the CRSN architecture a number of questions must be answered first:

- Has this ARH statistics to be collected? If so, what kind should it be - temporary or permanent or both?
- Is the ARH going to transfer data within the network - via Transport Network (TN)? If so, is messagegram service sufficient or is protection of the data needed so that a connection is necessary?
- What configuration shall the ARH have? How many resources does it have and what configurable parameters do they contain?
- Has the ARH interface to the firmware in the Line Termination Unit (LTU) (if any)? Does some software exist which implements the interface?
- Which resource should be traced?

When all the questions have been answered, the ARH developer can take the modules from the RH Template which provides the desired services and implement the few interfaces to the RH Template modules; e.g. fetching status from an external process. The developer must enter the configuration for the application ARH in a prefix for the RH Template - as this is the key data structure in the ARH. Then he compiles the selected RH Template modules and links the result to a link library. This module is now ready to be linked to the application ARH link modules. This can be done after the developer has designed and coded the application specific software.

The application ARH is now ready for test and subsequently integration. Some specification of configuration, status and statistic need be specified for the network manager (NM) as database records.

Database Specification

This section describes the entities that every developer must introduce to the on-line database before the program can be loaded into the network.

When new application software is going to be inserted in the network the developer has to specify some specific database information:

- **The boot-file** is the object code from the compiled and linked program. This file is to be specified as input for the system generation tools.
- **The database configuration** is the configuration which the individual developer specifies to the CRSN configuration tool. The configuration can obtain both dynamic and static items; static configuration is a configuration that cannot be changed after initialization of an ARH - the dynamic configuration specifies on-line configuration updates which the ARH must be able to receive at any time during processing.
- **The status and statistic records** that can be collected from the ARH must be specified. There are two statistic types: temporary and permanent statistics. The temporary statistic is subject to collection on operator initiative, the permanent statistic is subject to collection on regular time intervals.

A REFERENCED DOCUMENTS

	Architecture
SE/SDS/0001	AXDN CONCEPTS AND TERMINOLOGY
SE/SPC/0003	AXDN LEVEL 1 RESOURCE HANDLER ARCHITECTURAL SPECIFICATION
	Basic Software
CSS/7380/USM/0116	CR80 MXAMOS TOS USER'S MANUAL
CSS/381/USM/0037	MX COMMAND INTERPRETER USER'S MANUAL
CSS/102/USM/0021	CR80 ON-LINE EDITOR USER'S MANUAL
	Programming Languages
CSS/449/RFM/0004	CR80 PASCAL REFERENCE MANUAL
CSS/427/RFM/0021	CR80 C REFERENCE MANUAL
CSS/415/RFM/0002	SWELL 80 REFERENCE MANUAL
	Linker
CSS/416/USM/0048	CR80 AMOS, LINKER USER'S MANUAL
	Test Tools
CSS/8210/USM/0125	CYCLIC DEBUGGER USER'S MANUAL
CSS/216/USM/0124	DISPLAY CYCLIC DEBUGGER USER'S MANUAL
CSS/7201/USM/0107	MX DEBUGGER USER'S MANUAL
CSS/212/USM/0131	TEST COMMAND INTERPRETER
	Resource Handler Template
CPA/PSP/001	PRODUCT SPECIFICATION FOR THE RESOURCE HANDLER TEMPLATE

CR Systems