M A N - M A C H I N E I N T E R F A C E

T O A N E L E C T R O N I C

T E L E P H O N E E X C H A N G E

U S I N G A M I C R O C O M P U T E R

Ian Geoffrey Kennedy

A Dissertation Submitted to the Faculty of Engineering,
University of the Witwatersrand, Johannesburg,
in fulfilment of the requirements for the
Degree of Master of Science (Engineering)

Johannesburg, February 1985

Man-Machine Interface
to an Electronic Telephone Exchange
using a Microcomputer

ABSTRACT

This dissertation describes a project that created a user-friendly Man-Machine Interface to the Operations and Maintenance Centre (OMC) computer. (The OMC computer controls the SA128E version of the E10B Electronic Telephone Exchange used in the South African public network.)

The approach taken involved no changes to the OMC computer, but incorporated the necessary logic in a microcomputer that replaces the existing Visual Display Terminal.

In anticipation of the 1984 International Telegraph and Telephone Consultative Committee (CCITT) Man-Machine Language (MML) recommendations, a form-based interface was created for the microcomputer. The existing command language was converted to a menu-based language in order to place the minimum demands upon the operators' memory. On-line help screens were provided to give further assistance to the operator.

KEYWORDS

advanced terminal;
communication program;
communications computing;
communications line;
E10 exchange;
man-machine interface;
man-machine language translator;
microcomputer;
on-line operation;
on-line systems;
operation and maintenance;
personal computer;
protective ware;
program interpreter;
user-friendly;
visual display terminal.

DECLARATION

I declare that this dissertation is my own work. It is being submitted for the degree of Master of Science (Engineering) to the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination to any other University.

J. G. Kennedy

J. G. Kennedy

14th day of February, 1985.

DEDICATION

TO ROSEMARY

DEDICATION

TO ROSEMARY

THE PREFACE

The preface to this dissertation identifies the subject, the purpose and the audience of this dissertation. It outlines the organization of the dissertation, and concludes with the acknowledgements.

This dissertation describes a research project to improve the man-machine interface of one type of Electronic Telephone Exchange (ETE) used in the South African public network. The project took the form of a practical plan to improve the existing interface by creating a Man-Machine Interface (MMI) using microcomputers in place of Visual Display Terminals (VDT's). It was discovered that such an plan was reasonable.

The project relates specifically to system level 7.4 of the SA128E version of the French E10B ETE and associated Operations and Maintenance Centre (OMC) computer. The MMI runs on the Apple IIe microcomputer, using the Apple Pascal operating system version 1.1.

This dissertation is directed to those who seek to improve an existing man-machine interface. It indicates which techniques of Software Engineering can be applied to such a project. It is assumed that the reader is familiar with the principles of Telecommunication and Software Engineering [1 p. 32].

Chapter One defines the problem of the existing man-machine interface and its root causes. A broad overview of existing knowledge of man-machine interfaces is given. Subsequent chapters explain the MMI project according to the phases of the Software Development Cycle [1 p. 31]. Chapter Two analyses the problem, and delineates the scope of the project. Chapter Three deals with the functional

and interface requirements, and Chapter Four the design phase. The implementation phase is treated in Chapter Five, the creation of the program in Chapter Six, and the division of the program into modules in Chapter Seven. Chapter 8 explains the system test plan. Finally, conclusions are drawn in Chapter Nine.

The author wishes to thank the staff of the South African Post Office (SAPO) for their support, discussions, and interest. The author would also like to express special thanks to his supervisor, Professor H. E. Hanrahan of the University of the Witwatersrand for his constructive advice and guidance during the project.

LIST OF ABBREVIATIONS USED

The following is a list of the abbreviations used in this dissertation. The list does not include mnemonics for characters, instructions, procedures or units:

ASCII = American National Standard Code for Information Interchange
AC = Administrative Centre
CCITT = International Telegraph and Telephone Consultative Committee
CHILL = CCITT High Level Language
EPROM = Eraseable Programmable Read-Only Memory
ETE = Electronic Telephone Exchange
EWSD = Siemens Digital Electronic Switching System
FDL = Formal Description Language
MMI = Man-Machine Interface (this project)
MML = Man-Machine Language (CCITT)
OMC = Operations and Maintenance Centre
PABX = Private Automatic Branch Exchange
ROM = Read-Only Memory
SAPO = South African Post Office
SSC = Subscribers' Service Centre
UCSD = University of California at San Diego
VDT = Visual Display Terminal

TABLE OF CONTENTS

CHAPTER 3 - FUNCTIONAL AND INTERFACE REQUIREMENTS

CHAPTER 4 - DESIGN PHASE

CHAPTER 8 - SYSTEM TEST PLAN

CHAPTER 9 - CONCLUSIONS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1. INTRODUCTION TO THE PROBLEM

The SA128E ETE is disappointing in its lack of consideration for the user. In particular, administrative staff of the SAPO experience difficulty in learning and remembering the mnemonic symbols [1 p. 24] of the command language [1 p. 12] used to control the SA128E ETE via the OMC computer. The cryptic error messages returned from the OMC computer also create problems. Unfortunately, the terminals connected to the OMC computer and the operating system include no flexibility to allow changes or improvements. This section gives the reason for starting the MMI project. It details the causes of the problem being experienced locally with the existing man-machine interface, and also delimits the scope of the project.

The problem is caused firstly by the existing terminals. Traditionally, telephonic functions are controlled by having operators type their commands onto simple terminals. (Intelligent terminals are starting to be used by the SAPO -- these terminals have e.g. function keys that can be loaded with frequently used character strings.) The operators have to remember the French mnemonics for the functions and the sequencing of them.

Unfortunately, the simple terminals contain no programs, so it is not possible to modify them to improve the man-machine interface.

The lack of a standard interface aggravates the problem further. Daily tasks executed at ETE's are similiar for all telecommunication administrations. However, the operator interfaces used by SAPO are not standardized. This is a partly as a result of the ETE's being designed at different points in time [2 pp. 144,145], and partly as a result of the ETE's being designed by different manufacturers.

The problem is compounded by inadequate training. The operators at the Administrative Centre (AC) and Subscribers' Service Centre (SSC) received no overseas training or training in the French language, and minimal local training. They had no manuals, simply a few handwritten notes.

Finally, attempts to solve the problem are confounded by the design authority being based overseas. Operating systems for the ETE's used in South Africa are designed and maintained overseas. Consequently even the smallest change must be implemented with the co-operation of the overseas desi n authority.

In order to provide a solution to the problem this research project creates a better MMI. The approach taken allows the root causes of the local problems to be avoided or corrected. Learning and remembering difficulties become minimal, inflexible terminals are replaced, and the problems with the operating system of the OMC computer are circumvented.

1.1 INTRODUCTION TO THE PROJECT

The remainder of this chapter gives the historical background to the present work. It reviews the available literature, laying the foundation for the MMI project.

When the first SA128E was installed, it printed a new fault message (using French mnemonics) every 5 seconds. The sheer volume of messages caused them to be ignored by the maintenance staff. To help the staff, the author created a system named TRACE FAULTS to automatically translate and analyse these messages. The system helped the staff to readily determine the pattern of faults and to take remedial action where it was most needed. The system was of great utility during the period of coming to terms with the new ETE. As a result of the success of this system, a similiar system for automatically translating, analysing and distributing alarm messages has been developed by a team at the Post Office laboratories [3].

Attention was focussed subsequently on operators other than the maintenance staff at the ETE. These operators also had to suffer an awkward interface. They received no overseas training and minimal local training. Consequently they experienced difficulty in initially learning the French mnemonics required to command the OMC computer. Operator manuals were available, but were not problem-orientated. A quick look at a log of their work revealed highly inefficient keying strategies, contributing to low productivity.

The problem of training is not unique to South Africa. Many facets of Man-Machine interactions were covered in the International seminar held in Zurich in 1982. In particular, Beredetti argued in favour of funding research in order to reduce staff costs:

> "Costs incurred by Administrations ... for the operation and maintenance of telephone systems continue to increase as a result of steadily increasing personnel costs. Such costs will not decrease with the introduction of SPC (Stored Program Control) systems; in fact, the enhanced and sophisticated features of SPC systems require more experienced personnel, which will augment costs for the personnel training and/or retraining.
>
> It would thus seem appropriate to invest in research ... to supply, whenever possible, automatic computer supported tools that can assist the operator in the operation and maintenance procedures." [4 p. 269]

In the field of improving man-machine interfaces in telecommunications, an extensive literature survey revealed one similiar project [5], [6], of a microcomputer being connected as a terminal to a public ETE (System X). This present project is original in that forms are used for the operator interface, and connection [4 p. 14] is made to the OMC computer.

1.2 THE MAN-MACHINE INTERFACE (MMI)

Before using the theoretical knowledge on man-machine interfaces, it is necessary to bear the warning of Tesler in mind:

> "Some people think they can design friendly software just by being clever, or by reading papers on principles of user interface design" [7 p. 9].

In order not to fall into this trap, the MMI project first takes a theoretical and then a practical approach to the problem. The remainder of this chapter deals with the

theory of man-machine interfaces in general, and then concentrates on man-machine interfaces in the field of telecommunication. The man-machine interface will be analysed in three parts: the man, the machine, and the interface between them.

1.3 MAN-MACHINE INTERFACE -- THE MAN

The first part of the man-machine interface is the man (or woman). In this context, some references talk about the "user", others refer to the "operator". Tesler advises enlisting the help of typical users at their workplaces, and gives four goals for the interview. The difficulties that the users experience must be observed, and explanations need to be sought about [7]:

1) "Procedures" -- the nature of the current work, how the operator obtains the data [1 p. 15] and enters it; what functions the operator performs; the interrelationships between the tasks, and the sequencing of the operator's activities.

2) "Problems encountered" -- the clarity of the displays, error-rate etc.

3) "Forms and Tools" -- the flow of paperwork and the flow of data.

4) "Terminology" -- the terms used by the operators.

To overcome the difficulties of the users, some guidelines [8] suggested by Simpson need to be borne in mind during the development of an interface:

1) be consistent,

2) minimize the demands placed on the operator's memory,

3) "keep the program simple",

4) match the program to the level of skill of the operator,

5) sustain the orientation of the operator,

6) encourage the operator to form a "conceptual model" [9] of the interface,

7) use the "what you see is what you get" principle, and

8) have universal commands.

Immediate feedback is important, and La tz advises that feedback be provided after each parameter [1 p. 26].

> "The user does not have to wait until the command is completed to discover that he mistyped the first parameter." [10 p. 96]

1.4 MAN-MACHINE INTERFACE -- THE MACHINE

The second part of the man-machine interface is the machine -- a computer terminal. Fortunately, the study of terminals is a more exact science than the study of man, but only recently are guidelines [11] becoming available. Currently no international recommendations or standards exist [12]. In the "well-established ... areas such as keyboard ... and static display design" [11 p. 21], useful guidelines for the hardware designer do exist. However, "in less established areas such as dialog(ue) types and display formatting ... guidelines ... concentrate on

hardware" [11 p. 21]. Few guidelines exist in the areas of "dynamic displays ... input devices ... and interactive dialog(ue) modes" [11 p. 21].

1.5 MAN-MACHINE INTERFACE -- THE INADEQUATE INTERFACE

The third part of the man-machine interface is the interface itself. This man-machine interface has two directions: the first is from the Machine to the Man; the second is from the Man to the Machine. In the first direction, the Machine displays information to the operator usually on a screen or printer. In the second direction, the operator usually enters data by means of typing on a keyboard.

Why is the design of most of these interfaces inadequate? Answers are found in the literature:

1) "Those who initially used computers were often the original designers of the machines and therefore had an intimate knowledge of their method of operation" [13 p. 337].

2) "The implementors were unable or unwilling to expend so much effort on the user interface" [14 p. 24]. We know that many lines of code go into making a program "user-friendly".

3) Machines themselves were "very expensive" [13 p. 337] to work, and the operator interface had to be as (machine) efficient as was possible. One way of achieving this is to skimp on the man-machine interface:

> "Cryptic prompts and legends are not only easy for the programmer, but they also save valuable time and space" [15 p. 74].

4) "The designer's background is likely to be strongly technical; it is unlikely that he will have training in ergonomics or in ... educational technology" [13 p. 338]. (Educational technology is the application of technology to education e.g. Computer Aided Instruction.) The designer (erroneously) assumes that the operator is also a technical expert.

5) The design of the teleprinter interface has had a large influence on the design of subsequent interfaces. Hayes et al. [14 p. 23] and James [13 p. 342] agree sadly that the operator and the computer are often still restricted to taking turns to type on a scroll of paper or on a scrolled screen. Hayes et al. and James are also distressed by the "oppressive demand" [13 p. 343] of the computer for complete accuracy in a specific, "highly restricted artificial language" [14 p. 19].

6) Maguire (cited by Paxton [16 p. 138]) points out that designers are often confronted with conflicting research results and information. In the design of an operator-interface there is often no "right" answer. The negative response to this is that "many ... issues are highly subjective and are therefore addressed in an 'ad hoc' fashion" [17 p. 246]. Man cannot easily be measured. As James expresses it: "There is a very complicated machine on the other side of the interface" [13 p. 339]. Unfortunately, psychologists "have rarely been invited by users or system designers to participate in the design process" [13 p. 339]. Consequently "the cognitive issues in human factors are largely unexplored" [18 p. 333].

Because of these problems, the man-machine interface became a "man-machine communication barrier" [14]. Today, the widespread use of microcomputers can help to change the

situation. We can asymptotically approach the most effective means of man-to-machine communication, viz. natural language. As James puts it:

> "People communicate most effectively when they can both use the language which has been familiar to them since birth." [13 p. 342]

1.6 PRINCIPLES FOR THE INTERFACE DESIGNER

What positive steps can the designer take to ensure a good man-machine interface?

1) One positive step the designer can take is to remember that "producing a good user interface is basically concerned with human patterns of learning, using, remembering and forgetting" [13 p. 339].

2) More rigorous approaches to operator-interface design are developing and can be taken advantage of:

> "It is now generally recognized that a formal analysis of the man-machine interface is an essential part of the design process for any good interactive system" [19].

3) An early critical step is to define the operator's background, conventions, needs and goals (task analysis [17 p. 248]). Typically:

> "users will have no technical knowledge of computing systems ... the general 'member of the public' ... will increasingly become the dominant class of computer user" [13 p. 340].

4) Next, what the operator is "trying to do with the

computing system" [13 p. 340] must be determined.

5) At all stages, the limitations of the medium must be remembered. One useful approach is to regard the man-machine interface as being a medium of communication -- a much less effective method of communication than man-to-man communication.

6) As a final step forward, the designer of an interface must remember at all times that operators are peculiar people:

> "They continually make mistakes and they change their minds ... They forget what has previously been done ... They require constant reassurance that everything is working as expected" [13 p. 348].

1.7 MAN-MACHINE INTERFACE TO ELECTRONIC TELEPHONE EXCHANGES

As in any other field, good man-machine interfaces are particularly needed in the field of telecommunication:

> "In large exchanges, where the consequences of an error are considerable and the amount of data to be handled is formidable, the user's friendly man-machine language eases the problem" [20 p. 9.7].

Unfortunately, the advanced features of ETE's require operators to have higher qualifications. One method of coping with the advanced features is to provide the best "interface between the operator and the system in order to enhance the effectiveness of the operator's work and to reduce training costs" [4 p. 270]. Currently the trend is for even PABX data (e.g. Centennial II) to be entered interactively by "non-technical persons" by means of "a low-cost commercial personal computer" [21 p. 1013].

In response to the needs of the telecommunication community, the CCITT has recommended the use of a man-machine interface termed Man-Machine Language (MML) [22]. The purpose of the Recommendations is to formally define the syntax [1 p. 34] and grammar of the man-machine dialogue. This provides clarity to the designers and testers of the ETE's, allowing communication with the ETE via a simple teleprinter-like terminal. The CCITT MML is already in use in one type of ETE in South Africa [23].

1.8 OBJECTIVES OF THE CCITT MAN-MACHINE LANGUAGE (MML)

Telecommunication Administrations have stated to the CCITT their requirements for a MML. The MML should:

1) be applicable to the "operations, maintenance, installation and testing" phases [24 p. 12];

2) fit in with the "organizational structure" of any Administration [24 p. 12];

3) be secure against "mistakes (made) by operational staff" [20 p. 9.6];

4) "be easy to learn and use" [24 p. 12], and "be friendly and forgiving" [20 p. 9.15];

5) "accommodate different national languages" [24 p. 12];

6) "allow quick and easy recovery from input errors" [24 p. 12];

7) "be extendible" [24 p. 13];

8) "be structured so that subsets" of the language may be used by different groups of staff [24 p. 13].

1.9 MML COMMUNITIES IN SOUTH AFRICA

Operators in the SAPO can divided into three communities according to the MML functions they perform:

1) the Administrative Centre (clerical staff),

2) the Subscribers' Service Centre (technical staff),

3) the ETE's and the OMC (technical staff).

The AC is composed of two sections each having the authority to suspend a subscriber's service. The two sections are the Contracts Preparation Section, and the Accounts Section. The suspensions being effected by the two sections are known as Contracts Suspension and Accounts Suspension respectively.

A list of the tasks performed at the AC follows, based on a list in the CCITT Recommendation:

"Subscriber administration

- taking subscriber's lines to/out of service (including PABX lines);
- (allocating, changing and removing) ... classes of subscriber service;
- changing of a subscriber's number;
- blocking and unblocking a subscriber's line;
- interrogation of a subscriber's classes of service;
- interrogation of blocked subscriber lines;
- reading of subscriber's charging information;
- retrieval of charging information;
- malicious call tracing;
- putting a subscriber on to 'subscriber charging observation'." [22 p. 115]

A list of the tasks performed at the SSC's has been extracted from the same Recommendation. The tasks are more closely linked with subscribers' lines and equipment:

"Maintenance functions

Maintenance of subscribers' lines

- test of one subscriber's line and associated equipment;
- test of a group of subscribers' lines and associated equipment;
- measurement of one subscriber's line and associated equipment;
- measurement of a group of subscriber's lines and associated equipment;
- blocking or unblocking a subscriber's line for maintenance purposes;
- observation or supervision of subscribers' lines and equipment." [22 p. 116]

A list of the tasks performed at the SAPO ETE's and at the OMC's has also been extracted from the Recommendation. The tasks are more specialized and less routine:

"Subscriber administration ...

- reading of subscriber's charging information;
- retrieval of charging information;
- malicious call tracing;
- putting a subscriber on to 'subscriber charging observation'

...

Routing administration ...

Traffic administration ...

Tariff and charging administration ...

System control operation ...

Maintenance of lines between exchanges ...
Switching network maintenance ...
Control system maintenance ...
Plant installation functions ...
Testing functions." [22 pp. 115-118]

1.10 ADVANTAGES AND DISADVANTAGES OF MAN-MACHINE LANGUAGE

The advantages of MML are the following:

1) It may be prompt driven. The operator need not remember all parameters, but may be "lead ... through (the) entry of instructions and parameters" [20 p. 9.16].

2) Alternative modes for the entry of instructions exist. "Parameters may be identified by names and come in any order or be identified by their position in the instruction" [20 p. 9.16].

3) MML can be "self documenting. If the full form of the dialogue is used and is recorded on a hardcopy terminal" the work of the day is documented [20 p. 9.16].

4) Three modes are available to allow staff of differing experience to enter instructions.

   (a) In the parameter-name mode the format is:-
   command code: parameter name = value, ... ;
   e.g.

   <CRSUB:DN=8036465,EQN=15-7-4,CAT=MS,SCOS=PB!
   DN=8036456,EQN=15-7-5,CAT=MS,SCOS=PB;

   Here the order of the parameters is NOT significant.

(b) In the parameter position-defined mode the format is:-

command code: parameter 1, parameter 2, ... ;

e.g.

<CRSUB:8036465,15-7-4,MS,PB;

Here the order of the parameters is significant.

(c) In the parameter prompt mode the format is:-

command code:
parameter name = value;
..... ;

e.g.

<CRSUB:
DN=8036465
EQN=15-7-4
CAT=MS
SCOS=PB;

Here the ETE supplies the prompt (e.g. DN=), and the operator supplies the rest of the line.

The disadvantages of MML are:

1) Of necessity, a great number of parameter types are available. To speed up entry for the experienced operator, mnemonics are used. However this is confusing for the inexperienced operator, who must learn the meaning of a number of command and parameter mnemonics.

2) If the parameter position-defined mode is used, the operator has to bear the burden of remembering the correct order as well as the required mnemonics. (CCITT Recommendations do not and can not go as far as specifying what the order of the parameters should be.)

3) If the operator has made a mistake in the middle of a line, the operator cannot correct just that mistake without having to type in characters that have already been entered correctly.

4) Provision is made for the operator to type spaces and Carriage Returns to improve the legibility of the record, but this provision is not used in practice as it is time consuming.

5) Attention to correct punctuation is essential. In this context, it should be remembered that the standard of education of many operators is such that they cannot even name punctuation marks correctly.

6) There are no Recommendations regarding the interface between the terminal and the computer.

7) A menu based approach is not supported.

8) No Recommendations for HELP screens exist.

9) Features of modern "intelligent terminals" [25] such as cursor control, paging and function keys are not catered for.

To overcome the disadvantages inherent in MML, the CCITT Study Group XI is studying enhancements to MML to incorporate the use of the full range of features provided by advanced terminals [2 pp. 155-160], such as the use of menus and forms. The scope of reference of the Study Group also includes improvements to the dialogue procedures and syntax. The work of the Study Group is complex because there are a variety of Human Factors to be considered, and because the Recommendations must not stifle innovation or disregard technical advances.

1.11 THE USE OF FORMS

Surprisingly, the technique of entering data by means of forms is not new. In 1973, Martin described the method as follows:

> " ... present the user with a 'form' to fill in. The form has blanks on it, and the user moves the cursor to the appropriate place and fills in the blanks." [26]

In the field of business, Shu et al. are convinced that business activities can be expressed naturally in terms of the processing of forms:

> "We have decided to take a forms oriented approach because we concur with many others that forms are the most natural interface between a user and data" [27].

Brockman cites Galitz [28] as saying that a well-designed screen format can reduce human errors.

In the field of telecommunications, the CCITT Working Party XI/3 has also noted this:

> "the use of forms offers great advantages ...
>
> - well-structured forms are easy to fill in;
> - an improved user guidance (results) ...;
> - ... data ... input at special points on the screen ensures a fast man-machine dialogue." [29 p. 131]

The Working Party feels that such forms should be designed and their elements (data fields) be defined from the viewpoints of functionality, clarity and efficiency. The Working Party posed a list of questions [29 pp. 133-

138] that need to be resolved during the design of forms for advanced terminals. "The questions ... are divided into four separate but closely related functional areas: data entry, data display (screen layout and menus), interactive control (navigation) and user guidance." [29 p. 133] A Working Team was formed [30] to study the capabilities of these advanced terminals.

1.12 SA128E MAN-MACHINE OPERATION

The man-machine syntax of the SA128E ETE does not comply with the CCITT MML Recommendations, but "employ(s) a language common to all French electronic switching systems and defined (by the French PTT in 1977)" [31]. The main characteristics of the French PTT standard are the following:

1) It is not prompt driven. The operator needs to remember a large number of instructions and formats.

2) No alternative modes exist for instructions, which are in the form of French mnemonics. Parameters must be identified by French mnemonics, but may come in any order. One of two modes may be selected: continuation mode (parameter line terminated with a colon and a carriage return) or exit-to-command-level mode (parameter line terminated with a semi-colon and a carriage return).

3) Only a parameter-name format exists:
The instructions typed by the operator follow the following format :-

```
logon
command code:
parameter name = value, parameter name = value...:
.
.
.
parameter name = value, parameter name = value...;
```

e.g.

```
SOH
ABOCR:
ND=741465,NE=15-7-4:   <--- continuation mode
ND=741466,NE=15-7-5:   <--- continuation mode
ND=741457,NE=15-7-6;   <--- exit to command level
```

1.13 SUMMARY OF INTRODUCTION

This chapter has introduced the problem being experienced with the SA128E ETE man-machine interface. The historical background to man-machine interfaces, with particular reference to applications in telecommunications has been presented. A need for work in the fields of business and telecommunications into better man-machine interfaces has been revealed in the literature. Theoretical guidelines for designers working in these fields have been gathered together in preparation for the design of a new Man-Machine Interface.

The present work seeks to apply the theoretical knowledge of man-machine interfaces to solving the practical problem created by the existing unfriendly man-machine interface to the SA128E. In particular, attention is focussed on solving the problem being experienced by the administrative staff of the SAPO, who have no technical background, no knowledge of French and little training. The method that was adopted avoids the need for experienced

staff by replacing the inflexible terminals and presenting a uniform interface through the medium of menus and forms. The need for training of the administrative staff was further eliminated by including help screens.

Besides for being unfriendly, the existing man-machine interface to the SA128E contributes to low productivity by forcing the operator to adopt inefficient keying strategies.

CHAPTER 2
REQUIREMENTS ANALYSIS

2. INTRODUCTION TO REQUIREMENTS ANALYSIS

This chapter analyses the requirements [1 p. 29]: the needs of the operator are studied in order to define the system required. The chapter outlines the MMI project as a solution to the problem that was introduced in Chapter One, and gives the method used to determine the real needs. The computing power required is estimated, and literature on the use of microcomputers in similiar applications is surveyed. The novel methods used are set out in order to concentrate attention on implementing the critical pieces first [1 p. 15].

2.1 AIM AND SCOPE OF PROJECT

An ideal solution might have been to redesign the OMC computer software, matching it to an intelligent terminal. For practical reasons, a different approach had to be adopted. This involved putting more software in the terminal, with no changes to the OMC computer software. By replacing each simple terminal previously used with a microcomputer, enhanced functionality could be provided to the overseas design at the same time.

One aim of the project was to use good design techniques. However, no attempt was made to produce the best possible design through original research into human factors. An outline of the project follows.

The project involved analysis, followed by the design and testing of a library of general-purpose and commonly needed routines for the microcomputer. A flexible approach was taken to cater for changes in operating practices in the SAPO, and different levels of ETE and OMC software. Model dialogues were stored on diskette, together with menu and help files that can be readily edited. Operations are menu-driven, so only a small User Manual was needed. Additionally, one option in each menu is the HELP option, providing on-line help. Whenever the operator requires assistance, the operator selects the help option and the microcomputer provides a concise solution to the current problem. As Magers defines it:

> "On-line help ... information, and the software for delivering that information, provided directly on the user's terminal screen ... , which provides reference and some times tutorial documentation about the use of an interactive computer system and minimizes or eliminates the need for the user to consult paper manuals." [32 p. 277]

The scope of the project was deliberately limited to fit the available resources. The project excluded the Siemens EWSD ETE as it is designed in compliance with the CCITT MML Recommendations [2 p. 144]. Instead, attention was devoted to the SA128E version of the E10B ETE from CIT Alcatel, France, which does not comply [2 p. 145]. The project included helping operators at AC's only. The tasks that other staff perform were not covered in the MMI project, but can form part of further projects.

2.2 INITIAL INVESTIGATIONS

The initial investigations were based on interviews. Operators and their immediate superiors were interviewed informally in their work environment. Sufficient information was elicited during the initial interviews that a questionnaire was rendered unnecessary. Also, it has been suggested that:

> "asking users about the value of some proposed change without giving them experience of it is an essentially useless guide to their satisfaction with it in practice." [33 p. 86]

The questions that [illegible] asked were centred around some suggested by Tesler [7]. The interviews yielded some surprising but honest opinions of the existing man-machine interface. Interviews were to be held again at the end of the project in order to gauge operator satisfaction with [illegible]

Some of the operators that were interviewed work with both the OMC computer and the PRONTO computer. (The PRONTO computer contains records of subscriber's fault reports.) Operators unfamiliar with other computers would not know that a better interface to the OMC computer was possible. Naturally the operators made comparisons. The implications of their comments are discussed later. During interviews at the AC, the staff made the following revealing comments:

1) "The layout of PRONTO is easier (to understand) than the E10."

2) "PRONTO tells you quite clearly what is wrong."

3) "The E10 only tells you that something is not right here ... if no meter reading is given we know that the E10 didn't respond to our suspension."

4) "I haven't seen anything like that before (the operators' manual [34])."

5) "Could we be given a list of all current E10 suspensions, (required for enquiries at the counter), without having to enter the first and last number of every exchange?"

6) "We would be very glad if the E10 looked like PRONTO."

7) "It talks French!"

8) "There are words there we don't understand."

9) "How to suspend, and how to restore - that's all we know." (PABX groups are a mystery.)

10) "It is slow." (Normally response is immediate, but when the OMC computer has been loaded, response times have extended to two minutes.)

11) "ALL this ... (pointing to cryptic error messages in E10 printout) ... What IS this? ... WE don't know."

12) "It (the E10) tells you the WHOLE story ..."

13) "WHY must you type in EVERYTHING?" "Why not type in 'Suspension Category 1' just once at the beginning, like PRONTO?"

At a SSC, an experienced operator was interviewed. She displayed intelligence, curiosity, a methodical nature,

and initiative. She had also received no training, and prided herself on being self-taught. Although she herself was managing, she pointed out that a system in one of the official languages would be more easily understood by new staff when she had to teach them. She herself was experiencing difficulty in remembering even the simplest commands in French. She thought that "YES" in French was "IOU" until she consulted the notebook that she had made. She felt that the terminal should use the nomenclature traditionally used in the SAPO. She made erroneous guesses about how to interpret the results of subscriber line tests as presented by the OMC computer.

2.5 SUMMARY OF REAL NEEDS OF OPERATORS

The real needs of the operators have been derived directly from the thirteen comments quoted in the previous section, and define the goals of the MMI project. The operators needed a system that:

1) presents information in a format that is easily understood;

2) tells them clearly what to do when something goes wrong;

3) tells them directly what has been done successfully ("feedback" [8 p. 112]);

4) is well documented, in a place where the information cannot be lost;

5) can be modified to suit the users requirements;

6) is menu driven;

7) is in English (or Afrikaans);

8) spells everything out in full but is still brief and to the point --"clarity first and then, if possible, brevity" [15 p. 74];

9) holds the intelligence required to make complex tasks simple;

10) is rapid;

11) is lucid;

12) is brief and to the point;

13) requires a minimum of keystrokes to achieve an end.

2.4 ESTIMATION OF MACHINE-POWER

With the real needs having been defined, it is possible to start estimating what machine power is necessary to meet these needs. As Bernstein and Kashar admit, "there is no accepted definition for an intelligent terminal" [25]. They go on to say that the "Intelligent Quotient" (IQ) of a terminal can vary over a wide range. The "IQ" of the terminal required to satisfy the requirements of the project can be estimated in three major areas: memory, peripherals and processing power. The machine-power implications for the MMI were:

1) Memory -- space must be provided to keep the text of all translations and some error messages in main memory for speedy presentation on the screen, together with space for the sequencing of operations.

2) Peripherals -- the following peripherals must be connected: printer, modem, diskette (to load the operating system and program), and monitor.

3) Processing power -- sufficient reserve processing power must be available to store the complete the MMI syntax as part of the main program.

With the price of microcomputers dropping to the level of the cost of terminals, it becomes economic to use microcomputers in place of terminals. Weiderman has this to say:

> "We must provide the incentive of offering better interactive access to maxi computers through a microcomputer. In this case, 'better' can mean faster, cheaper, and more friendly computing ... Prearranged scripts are one way to shorten interactive sessions" [35 p. 33].

Consequently, a commercially available microcomputer was selected to replace the terminal used previously. The microcomputer has:

1) A memory mapped screen;

2) A single floppy-diskette drive;

3) A higher order [1 p. 20], structured [1 p. 34] programming language;

4) One port for connection to a printer;

5) One port for connection to a modem.

2.5 THE COMPUTER AS A TERMINAL

The philosophy of using a computer as a terminal to a larger computer is not new. The approach Colin and Mc regor took in 1976 was described as follows:

> "Instead of altering the internal code of the operating system, they have chosen to add on a separate processor, programmed wholly by themselves, which is responsible for communications with the user, and which presents to him an interface which is both more useful and convenient than that provided by the operating system on the main machine." [36 p. 9]

The advantage of this approach is "rapid response" [36 p. 39]. Their system offers many services, some of which are also found in the MMI project:

> "(1) The maintenance of a local file store.
> (2) The provision of an interactive editor ...
> (the MMI project has screen-based verification.)
> (3) ...
> (4) The ability to submit jobs to the main system.
> (5) ... printing on a ... (local) printer.
> (6) Running simple BASIC programs interactively."

Their system "allows the users to apply their intellect to genuine problems instead of the artificial difficulties and frustrations produced by a poorly designed interface" [36 p. 42].

The method of the MMI project is close to the method adopted by James. Because the description applies equally well to the MMI project, it is quoted in full:

> "My method for improving the quality of the current man-machine interface is very simple in principle. It is to place between the user and the existing interface a layer of software which we may call 'protective' ware. This operates rather like a communication channel in that it has at its front end an interface with the user and at its rear an interface with the existing operating system of the machine. All messages between the user and the original operating system are intercepted, monitored, and if necessary, interpreted. For example, an obscure, jargon-ridden message from the computer can be recognized and translated into a message meaningful to the user. In the opposite direction, a message from the user expressing rather approximately what is required may be converted by the protective ware into the sequence of control commands required by the operating system. It is possible that a single request from the user could generate a series of interactions between the protective interface and the original operating system." [13 p. 349].

"Protective ware" [37] can be embodied in a microcomputer which serves to protect the operator and the host computer from the worst aspects of each other.

Jordan expresses the advantages of software solutions in these terms: "The possibility for change is the feature which makes software solutions so attractive and is one reason why more software based microprocessor systems are replacing custom designed hardware." [38 p. 15] This feature of software was the reason for implementing the MMI project almost entirely in software. Software implementations are justified by Walker as being a legitimate engineering activity using this argument:

> "It would be absurd to view the generation of electric power as being, somehow, a more legitimate activity than utilising that potential through an electric motor. So too, the concept must be regarded as absurd that building a computer system from hardware components is more legitimate as an engineering activity than designing the software to realise its potential." [39 p. 46]

2.6 NOVEL METHODS USED

Four original methods were employed that were critical to the success of the MMI project. It was necessary to:

1) establish two-way communication between a microcomputer and the OMC computer;

2) quickly transfer the contents of a file containing information to the screen of the microcomputer used;

3) speedily represent forms on the screen by means of installing firmware [1 p. 19] containing the graphics character set required to draw forms;

4) treat help-screens, forms and menu-screens in a uniform way (with slight variations on account of their differing types), using a tree [1 p. 36] to access them all.

The MMI project is not the first project where microcomputers have been used together with a structured language to communicate with larger computers. For example, McBurney describes "an intelligent terminal program written in Pascal for the Apple II Plus" to access the "General Electric Information Services Company's ... Mark III time-sharing service" [40 p. 315]. However, the

establishment of communication between the microcomputer and the OMC computer required an investigation into the particular method of communication. The protocol [1 p. 28] had to be determined at all levels -- from the type of parity used to the method of recovery from errors. A teleprinter with a paper tape perforato provided answers at the primitive levels of the protocol. Unexpected characters from the OMC computer were detected this way. During software development, the higher levels of protocol were simulated by connecting two microcomputers together, with one representing the OMC computer, and the other the terminal.

When the operator selects the help option of the menu, a form that contains helpful information is quickly read off the diskette, and presented on the screen. The current screen is restored after the help screen has served its purpose.

The frames of forms could be drawn fast using vector graphics. However, in order to allow for easy maintenance of the forms, it was decided that graphic characters would be used. An analysis of forms revealed that eleven characters were necessary to build all forms (four corners, four Tee-pieces, a vertical line, a horizontal line and a cross-piece). The existing Read Only Memory (ROM) character generator of the microcomputer was removed. The ROM was replaced by an Eraseable Programmable Read-Only Memory (EPROM) that had been programmed to include the eleven graphic characters.

2.7 THE ADVANTAGES OF MENUS

Van Den Bos et al. clearly express the application of menus:

> "Menus find wide application in screen oriented dialogues because they guide the user through an application and at the same time they significantly decrease the possibility of user errors." [41 p. 256]

A menu may be defined as "a sequence of 'frames' or 'pages,' each containing some text and a list of options". "The text offers information to the user, and the options allow the user to choose what to do or where to go next". The operator "indicates a choice by typing a single character" (in the MMI project). "The selection made determines which frame will be displayed next" [42 p. 412]. Each option of a menu either leads to a terminal node [1 p. 25], or to another menu:

> "Since each frame has several options linking it to other frames, a frame can be thought of as a node in a network or graph, and the option links then correspond to 'arcs' or 'edges.' Moreover, since the option selection represents a one-way transition from one node to the next, the menu system (the collection of frames and option links) forms a directed graph" [42 p. 412].

(A graph [1 p. 20] is a model consisting of a finite set of nodes having connections called arcs or edges.)

By following the classic finite-state machine [1 p. 19] model used by Casey and Dasarathy, "features can be extended or new features added simply by adding appropriate nodes and arcs to the model and linking them to the global states." [43 pp. 559,560]

One particular advantage of a menu is that it requires less training than a parametric system does [16 p. 143]. Another is that the operator "does not have to RECALL the

label, only to RECOGNIZE it" [18]. Brown expresses the advantage of a menu in these words:

> "Menu selection ... makes the most of the computer's ability to find and display large quantities of information rapidly ... It also recognizes the slowness of the input channel, i.e., fingers typing (or hunt and pecking) on a keyboard" [42 p. 412].

Because of these advantages, the MMI project uses menus to communicate with the operator.

2.8 SIMPLIFICATIONS

Certain simplifications were made to increase the portability [1 p. 26] of the software. Assumptions made about the hardware [1 p. 20] were minimized, thus increasing the possibility of being able to transport the software to other hardware.

By excluding concurrent processes [1 p. 13], it was possible to make use of a simpler language and ease programming. Excluding concurrent processing then excludes the possibility of overlapping operator tasks. However, it is not envisaged that the operator should ever be "time shared" among tasks, even if the system could allow it. Consequently the operator interface was modelled simply as an interactive application, running in a transaction mode. As Foot ta et al. describe it:

> "Transaction mode implies that each message sent to the (host computer) from a terminal must elicit a response before a subsequent message can be sent from that terminal. We consider each such message pair as one transaction." [44]

If an unsolicited message should arrive from the OMC computer it was decided to ignore it. This simplification avoided the need to either detect interrupts [1 p. 22], or to poll the remote port. Subsequent versions of the MMI program can incorporate polling as a means of detecting unsolicited messages.

2.9 SUMMARY OF REQUIREMENTS ANALYSIS

This chapter has given details of the method used to analyse the problem. It has outlined the MMI project to solve the problem, based on the real needs of the operators, derived from interviews. The amount of computing power has been estimated, and the literature on the use of computers and microcomputers as protective ware has been surveyed. Novel features critical to the MMI project were implemented first.

CHAPTER 3
FUNCTIONAL AND INTERFACE REQUIREMENTS

3. INTRODUCTION TO REQUIREMENTS

The microcomputer was required to function [1 p. 20] in either one of two modes.

The SAPO required, for security reasons, that in an emergency the microcomputer could work like a VDT. In this mode, the operation of the microcomputer would be transparent to the operator. This required a small program to emulate a dumb VDT. This program enabled basic communication to be established using a high level language. During development, this program helped to ensure that there were no critical low-level problems.

The second mode of operation is under the control of the MMI program. The MMI program works as follows. Upon switching on, the main menu is displayed. Dialogue with the OMC computer is conducted by transmitting commands to it and receiving solicited data from it. The microcomputer prompts the operators for their requirements (in English), using the telephone terminology that they are used to using. When sufficient information is available, the microcomputer translates it and relays it to the OMC

computer in the required format. Responses from the OMC computer are interpreted by the microcomputer, and presented in natural language to the operator.

3.1 REQUIRED PROCEDURES

Attention now focusses on the second mode of operation, the MMI program. A number of processes were required to support the main menu. Other processes were provided as necessary to satisfy the needs of the user, and to allow easy creation and maintenance of the system.

Since a form-based approach was indicated, it made sense to extend this approach to handle as much information as possible in the same way. Thus forms were used to:

1) present the menus to the operator,

2) present further explanatory information (help),

3) capture information from the operator.

In extending the form-based approach further, it was natural to maintain the same standard in storing information on diskette as in displaying it. By storing the image as a file of characters, a standard editor [1 p.17] could be used to create and change the text forms. The editor allows a developer or maintainer to:

1) name or call up the form,
2) lay out the frame of the form,
3) display the form,
4) insert headings, labels, prompts, data and other textual matter in the frame,
5) change the data in the form,
6) save the form.

To further conserve the main memory, the current exchange configuration, forms, menus and help screens are also resident in files on the diskette. When the microcomputer is switched on, a text file containing the current ETE configuration [1 p. 13] is read from the diskette. This contains e.g. the codes for the ETE's controlled. By editing this file it is possible to change the MMI system to match the ETE configuration.

However, for speed of access, all tables except error tables are resident in the main memory. During the running of the MMI program, help-screens are fetched from the diskette. Little processing time need be involved. It was discovered that help-screens could be quickly transferred in blocks [1 p. 10] from the diskette if a "brute-force" [45 p. 23] method was used in storing them. That is, the help files simply contain an image of what must be presented on the screen, including all the formatting characters such as spaces and form-drawing characters. The fast method developed to present help was also used to present forms and menus on the screen. The frame of the form or menu and the data are also transferred at the same time. The operating system that was chosen included the low-level functions BlockRead and UnitWrite, which permitted this fast reading and presentation of data to be accomplished. This technique [46 p. 479] has only been published for the Apple III microcomputer.

The microcomputer is required to print the work done by the operator. The printing process is simply the printing of variables from the memory of the microcomputer. The printing process is activated after the screen has been updated from the same variables.

The operator normally uses the translation mode. After gaining access to the appropriate form, the operator

enters information into it. The microcomputer reads the strings, check them, and the strings found in translation tables are sent to the OMC computer. Similiarly, in the reverse direction, a string from the OMC computer is looked up, and the string found in the table is presented to the operator in the appropriate place in the form currently being displayed. In order to keep the design open-ended, translation tables are stored in files (dictionaries) that can be edited for new applications. It was possible in many cases to create a set of strings whose meaning does not depend upon the context in which the strings are found. This reduced the number of dictionaries required.

While computer data [1 p. 12] is being awaited from the OMC, an activity indicator (i.e. an external representation of internal activity) is shown to the operator to indicate that something is happening. The possibility of the operator starting a new task has been excluded, in order not to overtax the ability of the operator.

The current state of the MMI system is always displayed prominently at the top of the screen so that the operator knows exactly where he/she is. A one-line "window" was reserved at the bottom of the screen for use during development, and subsequently for indication of OMC computer activity. The window allows the remaining memory size, the value of variables, and statements about the program state to be displayed. The actual characters being sent to, and being received from the OMC computer were displayed here during development. The level of detail displayed was controlled by the program developer. The version number of each package is displayed on the screen when the microcomputer is first switched on.

It is possible for the developer (or maintainer) of the system to modify the program, for a new application or

for changed circumstances. This is effected off-line, using the type of tool used to create the system in the first place.

3.2 INTERFACE REQUIREMENTS

This first half of this chapter has outlined the modes of operation of the microcomputer and the functions that it performs. The remainder of the chapter sketches the interface requirements [1 p. 22] that had to be met.

3.3 LOW-LEVEL FUNCTIONS

Some of the low-level functions (primitives) that were necessary for interfacing with the operator and peripheral devices are classified below:

1) interface to diskette
   - read a script file;
   - read a form from the diskette and display it;

2) interface to the screen
   - display the form from the diskette;
   - determine the location of the cursor;
   - move the cursor;
   - clear the screen;
   - display a string at a certain point on a form including centering a string in the form;
   - signal that an error has occurred.

3) interface to the keyboard
   - get a character from the operator;
   - read a string of characters from the operator;
   - check the digits and format of a telephone or equipment number;
   - verify that the operator has checked the data;

4) interface to modem
initialize card;
try out the link to the OMC computer;
send a command and parameters to the OMC computer;

5) interface to printer
send a character to the printer.

3.4 CLASS OF SATISFACTORY SYSTEMS

In sizing [1 p. 31] computers to decide which were satisfactory, experience led to the consideration of microcomputers with about 64 Kbyte of memory. At that time, in particular, the Apple IIe, the HP86, the IBM PC and the MicroEngine were considered. It was felt that the operating system had to be machine-independent in order to allow for the possibility of the hardware being discontinued.

Today, there are many more brands and models of microcomputers operating at higher speeds, making the selection difficult. With iconography and mice being available today as low cost output and input devices for microcomputers, it is tempting to design interfaces using them. Nevertheless, the basic method used to choose a computer remains unchanged:

1) Determine what the computer is going to be used for.

2) Find out how much money is available. (Roughly speaking, the more you pay the more you get.)

3a) Find ready-made software to fulfil the requirements, and then buy the hardware that supports the package. Alternatively,

3b) if no ready-made package will fulfil the requirements, purchase a development system based on sound software engineering principles, and buy as many software development tools as possible.

Two configurations of the terminal had been proposed:

1) The operator types on a terminal that is connected via a "black box" to the OMC computer.

2) The operator types on the microcomputer keyboard, and looks at its video display.

The second configuration was chosen because of its greater speed.

3.5 ANALYSIS OF MESSAGES

Three steps were taken to analyse the dialogue with the OMC computer:

1) Even though a description of the current transactions on the OMC computer was available, the transactions were checked manually. All options in the required area were exercised and the resulting log was kept. The log was marked to show the direction of all transmissions. This completed the specification of the existing dialogue with the OMC computer except for error messages, which were handled separately. The flow of conversation that constitutes a transaction does not always follow a linear path, but may be diverted one way or another depending upon the conditions valid in the OMC computer at the time. For example, the OMC computer reacts differently when a telephone number is non-existent to when it exists. A procedure was produced to cater for such cases.

2) The log was analysed into information elements ("data items" [47 p. 1944]), and the required action (such as scre n control) was determined.

3) A new display was presented for each key that the operator was allowed to select.

3.6 PASCAL AS A FORMAL DESIGN LANGUAGE

In identifying the original requirements that are specified for a project, and in maintaining these requirements, Heninger stresses the need for "completeness, precision, and clarity" [48 p. 2]. He also tenders the following good advice: "Be as formal as possible" [48 p. 4]. To follow this advice one needs either to create a new Formal Design Language [1 pp. 16, 20] (FDL), or to adopt an existing FDL. Chu describes the need for a FDL in these terms:

> "A software design language aims to describe the design of software so that other software engineers can understand it. It is developed to serve as a design tool to software engineers for communication and documentation. Thus, it must be highly descriptive and comprehensive. It should be precise and concise, but not at the expense of clarity. It is not a programming language which aims to help the programmer to conveniently and concisely write a correct program." [49 p. 297]

Pseudo-code [1 p. 28] could have been used as a FDL, but suffers from the disadvantage that it cannot be analysed and checked by computer. A technique introduced to the author by Hanrahan [50] was used instead: Pascal itself was adopted as the FDL. This is in line with Dijkstra's first description of a program leading to a top-down

design:

```
BEGIN
  "print first thousand prime numbers" [51]
END
```

By using an established language as the FDL, all the good attributes of the established language (such as strong typing [1 p. 33], structured nature etc.) were carried over into the FDL. Additionally, advantage was taken of the available Pascal environment:

1) The editor and cross-referencer for editing or cross-referencing Pascal was also suitable for editing or cross-referencing the formal design.

2) A Pascal program compiler was used to read the formal description. It automatically detected many typographical errors and checked the description for syntax errors. Subsequently, the implementation of the description in Pascal became trivial.

3.7 PECULIAR REQUIREMENTS

It is necessary to consider the peculiar requirements that make the project unique. It was a requirement of the SAPO that the PT80 printer be used, and so this particular printer was used as the device to log the tasks performed by the operator. Another peculiar requirement is considered below: the communication protocol.

3.8 OMC COMMUNICATION PROTOCOL

The physical connection between the OMC computer and the terminal is effected by means a pair of V24 modems and a data transmission path between the two modems that

carries the bit stream. The data link carries data in the form of characters. Data is coded according to the CCITT International Alphabet No. 5 [52]. The parity convention adopted is even parity. Transmission of data to and from the OMC computer is asynchronous at 300 baud or 1200 baud. Transmission to and from the microcomputer was chosen to be at 300 baud. To complicate matters, the end of a line sent to, or received from the OMC computer may or may not be preceded by the transmission of the tilde character (~) and seven delete (DEL) characters [52 p. 3] from the OMC computer. So, besides for CR LF, the sequence ~ DEL DEL DEL DEL DEL DEL DEL CR LF had to be anticipated.

A password is used to gain access to the OMC computer. The word is composed of three parts:

1) three characters (e.g. the initials of the operator),

2) a control character to improve security, and

3) three characters (e.g. three digits).

A list of control codes relevant to the project is shown in Table I.

Table I -
CONTROL CODES USED BY OMC COMPUTER

CONTROL-	DIRECTION	MEANING
A	to OMC	log-on character
D	to OMC	log-off character
F	to OMC	2nd part of password
J	to and from OMC	Line Feed
M	to and from OMC	Carriage Return

Characters sent to the OMC computer are normally echoed (full-duplex). An exception to this rule occurs after sending the second part of the password. The subsequent three characters are not echoed (for security reasons). Each terminal is connected on a one-to-one basis with an input/output port on the OMC computer. Thus, no switching network exists. Use may be made of a Point-to-point Virtual Circuit, where this is economic. A session is initiated by the operator sending the Start Of Heading character (SOH) [52 p. 4]. The OMC computer responds with an end of line sequence (see above) and the character "@".

The following data items are of primary importance. With the exception of the log-on character, the operator always types a line, and terminates the line with a Carriage Return, and a Line Feed is echoed. (The operator may correct errors that are noticed before the line is terminated.) There is NO single character (such as Carriage Return or ETX) used by the OMC computer to indicate the end of a message, as the OMC computer assumes a simple teleprinter is connected. In addition, the convention of the OMC computer is that the end-of-line sequence is transmitted BEFORE the text of the line. Thus, the last line transmitted will not be terminated by an end-of-line sequence. However, if the continuation mode is employed by the operator (":" instead of ";" just before Carriage Return), the response from the OMC computer will in most cases terminate with a new prompt "@" and advantage can be taken of this fact. The exceptions to this rule occur when the OMC computer prompts the operator within a transaction. These cases have to be anticipated individually.

The first command sent to the OMC computer during a session must be the password command, unless the tasks the operator is going to perform are not protected by a pass-

word. After this, any other commands may be used. The operator will conventionally log off with the command that shuts down the use of the password entered previously, but this is not mandatory. The telephone tasks are then performed by the operator typing command lines and (usually) parameter lines.

The OMC computer always transmits upper case characters, but will accept upper or lower case characters in commands. However, upper case characters are conventionally transmitted to the OMC computer.

The output from the OMC computer is buffered. No measurements were made on the size of this buffer. As characters may be sent almost continuously by the OMC computer, it is necessary to buffer the incoming characters in the microcomputer. The majority of the messages received from the OMC computer are short. A buffer size of 512 bytes [1 p. 11] was chosen for convenience, and covers most situations. However, if the operator uses certain commands and parameters without due caution (e.g. asks for a list of ALL lines connected to the exchange), the number of characters printed will be too great to carry in the memory of any microcomputer. The MMI program never precipitates such a situation because it avoids using such commands and parameters.

It is possible for a message ("TELEX") to be sent from one operator to another via the medium of the OMC computer. (In practice, little use is made of this medium.) Apart from the system startup message, "TELEX" messages are the only unsolicited messages that can occur, and some mechanism must exist for either ignoring or treating them. In the present version of the MMI program, it was decided to ignore all such messages as UCSD Pascal Version 1.1 did not allow easy access to the status register of the SY6551

Asynchronous Communication Interface Adapter. UCSD Pascal Version 1.2 provides access via a call to the function RemStatus [53 p. 57-58].

Two timeouts are imposed by the OMC computer. Both timeouts are based on inactivity of the keyboard. The first timeout is a short timeout which requires the operator to resubmit the original (or new) command. The MMI program avoids such timeouts by always assuming that a timeout has expired, and always starting ab initio. The second timeout is longer, requiring the operator to resubmit the password, followed by the original command. With level 7.4 of the OMC software, a timeout message is transmitted to the operator.

3.9 SUMMARY OF REQUIREMENTS

This chapter has outlined the functions to be performed by the microcomputer and sketched the required interface to the OMC computer. The description of the OMC protocol given above is not complete. For example, the exact characters that pass over the data link have not been specified (see the documentation [34] for details). From experience, even documentation tells only part of the story. In practice, the only way of obtaining a complete definition for an interface is to connect to the interface with an approximate model based on the information available, see if the model works, and then refine the model. This is not to belittle formal descriptions of interfaces, merely to point out that they have to be verified.

The MMI program filters out any unanticipated control characters. All known messages are intercepted and translated by the program before being presented to the operator. If an unknown message should surface (e.g. a

known message with a mutilated character, or a new format of message or new message from a new OMC software release), it is simply passed on to the operator. In the reverse direction, if the OMC computer does not receive the correct characters in the correct format, it will be quick to object by returning a message. This "known" message will be intercepted and translated by the MMI program in the usual way.

CHAPTER 4
DESIGN PHASE

4. INTRODUCTION TO THE DESIGN

This chapter explains the design [1 p. 16] of the MMI program and covers the design phase [1 p. 16] of the MMI project. The chapter first defines the main program of the Man-Machine Interface and then the coupling to the existing OMC computer. The main program is outlined at the highest level and explained. The existing syntax of the OMC computer and new MMI are contrasted by means of one example. The conventions adopted for data entry and display are explained. The sources of errors and the methods of handling them are treated. Finally the Architectural Design [1 p. 9] is laid out.

CHAPTER 4
DESIGN PHASE

4. INTRODUCTION TO THE DESIGN

This chapter explains the design [1 p. 16] of the MMI program and covers the design phase [1 p. 16] of the MMI project. The chapter first defines the main program of the Man-Machine Interface and then the coupling to the existing OMC computer. The main program is outlined at the highest level and explained. The existing syntax of the OMC computer and new MMI are contrasted by means of one example. The conventions adopted for data entry and display are explained. The sources of errors and the methods of handling them are treated. Finally the Architectural Design [1 p. 9] is laid out.

4.1 MAIN PROGRAM

The main MMI program is outlined in Fig. I, using Pascal as a FDL, and a description of the program follows. Stubs [1 p. 34] are shown for lower-level procedures.

The MMI program transfers the initial menu to the screen. The program then gets a character from the operator. The variable called "node", of type STRING, is used in a novel way to create a stack [1 p. 33], keeping track of where the operator is in the menu tree. Each time the operator moves in the tree, the appropriate ASCI file is read from diskette, and displayed on the screen. (An ASCI file is simply a UCSD Pascal text file with the two header blocks stripped off.) This process continues through screens of type "help" and "menu" until the current screen is a "data" entry form. The program then prompts the operator for the parameters required. If the operator confirms that the details are correct, then the command and parameters are sent to the OMC computer. As the MMI program will run in a dedicated mode, the EnterTree construct shown in Fig. I was used to keep the program looping forever.

```
PROGRAM MMI (* Man-Machine Interface *);

TYPE
  ScreenType = (help, menu, data);

VAR
  node: STRING;
  ch: CHAR;
  CurrentScreen: ScreenType;
```

```
(*******stub procedures follow*******)
procedure ReadAsciFile  (node: STRING);
    begin end;
procedure WriteScreen   (CurrentScreen: ScreenType);
    begin end;
procedure GetParam;
    begin end;
procedure Verify        (ch: CHAR);
    begin end;
procedure Send          (command, param: STRING);
    begin end;
procedure DeleteLastChar(node: STRING);
    begin end;
procedure Append        (ch: CHAR; node: STRING);
    begin end;
procedure GetChar;
    begin end;
procedure InitCards;
    begin end;
procedure TryLink;
    begin end;


PROCEDURE EnterTree;
(*enter menu tree*)


  PROCEDURE transfer(node: STRING);
  (*transfer to node specified and update screen*)

    BEGIN
      ReadAsciFile (*specified by*) (node);
      WriteScreen(currentscreen)
    END (*transfer*);
```

```
PROCEDURE GetAndSendData;
(*get data from operator, verify it and send to OMC*)

  VAR
    command, param: STRING;

  BEGIN
    REPEAT
      GetParam (*from operator*);
      verify(ch);
      IF ch = 'Y' THEN
        send(command, param);
    UNTIL ch = 'Y'
  END (*GetAndSendData*);


PROCEDURE GetForm;
(*get characters from operator until*)
(*a data form is reached            *)


PROCEDURE MoveInTree(CurrentScreen: ScreenType);
(*update node and transfer to current node*)

  BEGIN
    IF ch = 'Q'
    THEN (* back up tree *)
      DeleteLastChar (*from*) (node)
    ELSE (* go down tree *)
      append(ch, (*to*) node);
    transfer (*to*) (node)
  END (*MoveInTree*);
```

```
    BEGIN (*GetForm*)
      REPEAT
        GetChar (*from operator*);
        MoveInTree (*to*) (CurrentScreen);
      UNTIL (CurrentScreen = data)
    END (*GetForm*);


  BEGIN (*EnterTree*)
    transfer (* to initial menu *) (node);
    REPEAT
      GetForm;
      GetAndSendData;
    UNTIL false
  END (*EnterTree*);


BEGIN (*MMI*)
  InitCards (**initialize all cards in microcomputer**);
  TryLink (**try link to OMC computer**);
  EnterTree;
END (*MMI*).
```

Figure I -
FORMAL DESCRIPTION OF MAN-MACHINE INTERFACE PROGRAM

4.2 SYNTAX OF OMC COMPUTER AND OF NEW MMI

The syntax of a transaction with the OMC computer is best explained by means of a simple example. Other French commands implemented initially included ABOCR, ABOIL, ABOMO, ABOSU, PWOAR, PWOLA, TAXIN, these being the most commonly used commands. The example given in Fig. II shows the appearance of one transaction. To distinguish between the two directions of communication, the responses from the

OMC computer that the operator would have had to interpret have been recorded in lower case. The characters the operator would have had to have typed are shown in upper case. The same prompt character is used when the operator is in the command mode or in the parameter mode, so the operator must keep track of this mentally.

```
@ABOIL,CEN=1:                                    <--- COMMANDline
    cen=1/84-02-24/09 h 08 mn 52/sub characteristics list~
@ND=741009:                                     <--- PARAMETERline
    processing tglail acc~                       <--- ACCEPTEDline
      nd=741009          ne  =003-00-127  tax = 00030012~
           ty-kla+fla                            <--- 2nd DATAline
           cat=1st+na16+rvt+cof+caf2~            <--- 3rd DATAline
    processing tglail exe~                       <--- EXECUTEDline
@
```

Figure II -

EXAMPLE OF PREVIOUS METHOD OF PERFORMING A TRANSACTION

To accomplish the same task using the new MMI, the following steps take place:

1) From the main menu (Fig. IV), the operator selects the option "List", by pressing the key "L".

2) A form appears, prompting the operator to press either "D" for Directory number or "E" for equipment number.

3) The inside of the form is cleared, and the operator is prompted "Telephone number = ", together with the default exchange prefix. If the operator wishes to change the default prefix, the operator backspaces over the displayed prefix, and enters the full telephone

number.

4) The program checks that the operator has indicated a number that is reasonable, by checking for non-numerics, and by comparing the length of the number and the exchange prefix with the known set of permissible values.

5) The operator is given a last opportunity to change the number before it is sent off to the OMC computer. The mnemonics returned by the OMC computer are converted into English, displayed and printed as shown in Fig. III, for the example given in Fig. II.

Equipment No. = 003-00-127
Directory No. = 741009
Meter Reading = 00030012
Multi-Frequency phone
Recall Button
Non-metering Line
Short code dialling No. 16
Follow-me
Conference facility
Catastrophe facility No. 2

Figure III -
EXAMPLE OF NEW METHOD OF PRESENTING INFORMATION

If the operator has used the system previously that day, the last used telephone number is displayed. Often the operator will be performing tasks on similiar numbers (e.g. ascending sequences). If the telephone number differs from the last number used only in the last digits, the operator merely has to overtype the new digits. The probability of the operator performing more than one task

on the same number is high. To indicate that the number is unchanged, the operator simply presses the Carriage Return key.

When selecting an item from a menu, only a single alphabetic character is required. This is in line with the view of the designer interviewed by Hammond et al.:

> "Selecting items by numbers is bad because it is not something used outside computing. Keywords or abbreviations is better because this is closer to what the user is thinking about rather than having to scan the menu for a number" [54 p. 43].

Although selection by letter is more natural to the operator, many programmers still cling to selection by number. This may, in part, be caused by the lack of constructs such "IF ch IN ['I', 'G', 'K'] THEN" in the older programming languages. The UCSD Pascal environment uses selection by letter, but the options are squashed into a single prompt line. For the MMI, options fill the screen, and the option is selected by the operator pressing the first letter of the option required. The operator may use upper or lower case letters of the alphabet. An exception to this is that the operator presses the "?" key to select the help option and the menu is presented again if the operator presses the space bar.

The operator does not press the Carriage Return key. Requiring a single character with no Carriage Return results in the operator typing a minimum of keystrokes. However, when the operator is entering numeric data, such as a telephone number, the operator must type in the number and press the Carriage Return key to indicate the end of the data.

MAN - MACHINE MENU	
B	Begin by sending password
E	End by stopping use of password
D	Dismantle
L	List
M	Meter
N	New
R	Restore
S	Suspend
?	Press ? for help

Figure IV -
EXAMPLE OF A MENU (MAIN MENU)

METER FORM	
?	Help with Meter readings
Q	Quit Meter form

Figure V -
EXAMPLE OF A FORM (METER MENU)

METER READING HELP
To take a meter reading, you must first give either the Directory no., or the Equipment number and exchange prefix.
Press the Q key now to return to the Menu.

Figure VI -
EXAMPLE OF A HELP SCREEN

In the MMI program, all forms, i.e. menus, help screens and data entry forms that are displayed on the screen have a unity of style. (See Figs. IV, V and VI.) Every form is framed in a rectangle as large as the active screen area. The idea of this is to make the layout of the screen as attractive as possible. The characters used to draw the form have rounded corners in order to make the form less austere and enhance the appearance of the screen.

The work of Teitelbaum and Granda "demonstrates a clear performance advantage in searching for screen-presented information in a multiple screen interface, when that information is present in a positionally constant manner" [55 p. 153]. With this study in mind, the MMI program reserves positionally constant areas for information of the same type. At the top of the screen of each form is a title enclosed in a rectangle. This is provided

to orient the operator. At the bottom of every form is a similiar rectangle that contains information on how the operator can exit from the current form. One disadvantage of this approach is that if the intensity of the screen is left too high, it is possible for a permanent image to be burnt on the phosphor of the screen. This can be avoided by turning the screen off when it is not in use, or by periodically adjusting the vertical and horizontal controls slightly.

4.3 ERROR RECOVERY

The microcomputer used for the MMI has to recover gracefully from all errors. There are three possible sources of errors: the operator, the microcomputer or the OMC computer, and these are considered in turn.

If the operator presses an incorrect key when choosing an item from a menu, nothing happens: the screen remains unchanged. This decision was taken so as not to antagonize the operator. As long as the operator is navigating through the menus and help screens, the MMI program gets single characters from the operator. If the operator inadvertently presses a valid key, the wrong menu will be displayed, but the operator can always recover easily from this situation by pressing the "Q" key (for Quit). The previous menu will be reinstated.

If several micro-computers are going to be operated in the same office space it is possible to assign different beeps to different microcomputers to avoid confusion. If the operator has finished navigation, is entering data, and presses an unacceptable key, the microcomputer beeps to signal the error. This action has been taken because the consequences of entering incorrect data may be serious. The operator may correct errors until the Carriage Return

key is pressed, and it is expected that the operator check the line before depressing the Carriage Return key, as recovery from the situation can only be achieved by the operator undoing what was inadvertently done.

During an input-output operation with the micro-computer an error may occur. This is dealt with by immediately presenting a diagnostic message to the operator. These messages are stored in memory for immediate presentation. Problems may also occur in the OMC computer. These are dealt with by reading the message from the OMC computer, and looking up a translation from diskette. Storage on diskette was chosen to ease the task of maintaining the messages. The process which is used to look up the translation is slow and could be improved by means of a binary search. However, a binary search requires the dictionaries on the diskette to be sorted, and this may create practical problems during maintenance.

4.4 INTRODUCTION TO THE ARCHITECTURAL DESIGN

So far, this chapter has stated the design in a Pascal FDL. An example of the syntax of the Man-Machine Interface and the syntax of the existing OMC computer interface has been presented to show the data formats required. Errors made by the operator or error messages from the OMC computer are counted as being data formats that have to be treated in the normal course of affairs.

The remainder of this chapter defines the collection of hardware and software components [1 p. 12] and their interfaces, thereby establishing a framework for the development of the MMI. The remainder also describes the selection of a structured language, compiler and support environment for the project. It also describes the consideration which led to the purchase of a particular

microcomputer and mentions the need to customize the system purchased. The importance of designing with change in mind is remembered. The standard adopted for programming is included as well as a comparison of two microcomputers communicating with two different ETE's.

4.5 OPERATING SYSTEM AND ENVIRONMENT

Anderson and Shumate warn that "selecting a programming language, compiler and support environment is a difficult and important task" [56]. The selection should be based on sound considerations [57]. Young [58] suggests that languages used for real time [1 p. 29] work should be chosen on the basis of security [1 p. 30], readability, flexibility, simplicity, portability [1 p. 26] and efficiency [1 p. 17]. Considerations which applied to the MMI project were that the operating system [1 p. 25] be available "off the shelf", and that the application could be written using a structured programming language.

4.6 CHOICE OF STRUCTURED PROGRAMMING LANGUAGE AND HARDWARE

The advantages of structured programming [51] are well-known. Because of the greater availability of Pascal, it was selected in preference to other structured languages such as CCITT CHILL [59] or Ada [60], [61]. Many dialects of Pascal are available, of which one must be chosen. For example, Woteki and Sand have published a review of four implementations of Pascal [62 p. 352].

The University of California at San Diego (UCSD) dialect of Pascal was chosen for the MMI project because of its useful string extensions and associated environment. The UCSD Pascal environment includes a good screen-oriented editor that supports global find and replace operations, and a compiler that permits separate compilation of modules

[1 p. 24]. The UCSD System is available for a number of microcomputers in this country. The cheapest minimal system with UCSD Pascal was the Apple IIe. Thus one such microcomputer was purchased for development purposes, and two others for a field trial. Modula-2 [63] and [64] is also a good candidate, but was not chosen because of its newness.

The MMI program is a turn-key program. That is, a bootstrap [1 p. 10] starts the program automatically when the computer is switched on. A customized turn-key diskette, with a back-up [1 p. 10] copy is provided for each site. The diskettes can be customized by editing the menus to allow access to the allowed sets of options. The exchange configuration file must also be edited for each OMC by the maintainer of the system. The advantage of this approach is that no recompilation is necessary when changes have to be made.

The set of characters that can be displayed on the screen was changed to a custom set that allows the representation of simple forms. Patches [1 p. 26] to the operating system were necessary to prevent it from responding to control characters (e.g. CTRL-S, CTRL-F, CRTL-@), which could conceivably be typed by the operator.

It was thought initially that a modification to the hardware would be necessary to enable the microcomputer to be switched to either of the two required modes: dumb terminal or intelligent MMI. However, this was avoided by examining the software capabilities of the microcomputer more closely. It was discovered that switching under software control was possible if three conditions were met:

1) The characters being sent to the 80-column card were filtered to block the control characters for forward and

reverse scrolling from being sent to the screen. If the cursor is in the "home" position, it was also necessary to block a "backspace", as this precipitates a reverse scroll.

2) More than 23 lines must not be sent to the 40-column screen otherwise it will scroll, and certain pointers will be lost.

3) Before and after switching between 40-and 80-column modes, it is essential to send the cursor to the "home" position. By doing this, 40-and 80-column co-ordinates become synchronized.

In designing the MMI, the possibility of future changes to the design was borne in mind. Changes were considered under three headings: changes that would definitely occur some time in the future, changes that might occur, and changes that were ruled as being unlikely. It was correctly assumed that once the MMI had been tried out, changes would be requested to include improvements and additional features. The first such suggestion implemented was to have the program check to see if the link to the OMC computer was good, before progressing any further.

The changes to the design that were considered likely were:

1) The baud rate could be raised to 1 200 baud. This could be achieved by installing a speed-up card with a 6502 microprocessor running at 5,5 MHz. Alternatively, a polling scheme could be used in place of a transaction driven system. This is possible with UCSD Pascal version 1.2.

2) It may be necessary to accept unsolicited messages. This

implies that a polling scheme be used in place of a transaction driven system. The OMC computer will then be polled at frequent intervals to determine if an unsolicited message has arrived.

3) It may become necessary to couple the microcomputer to more than one OMC computer. This could be accomplished under software control by switching the communications port from one OMC computer to the another and would require some additional hardware.

It was assumed to be unlikely that changes 2) and 3) would occur together. Also, the overlapping of operator tasks was not envisaged because of the possibility of confusing the operator.

4.7 PROGRAMMING STANDARDS

The general programming standards [65] suggested by Aron were adhered to: top-down [1 p. 36], structured programming, using a standard higher level language and system library [1 p. 34] programs. The system library had to be built from scratch. The style advocated by Ledgard et al. [66] viz. the use of mnemonic names and careful commenting, was also followed. Their programming standards were adapted and adopted. For example, a general standard followed was that all programs be formatted by means of an automatic formatting program, and touched up manually where necessary. The formatting program automatically capitalizes reserved words, places one statement on a line, indents each statement or nested procedure correctly, and inserts the name of each procedure as a comment at the end of each procedure. Fig. 1 was formatted using this program.

4.8 PROTOTYPING OF THE FILE AND THE SCREEN STRUCTURE

A prototype system was constructed in order to learn from mistakes at a low cost. Gehani's comments [67 p. 480] about prototypes are very relevant: "In the engineering industry, unlike in the software industry, the construction of a product is generally preceded by the construction of a prototype." He advocates prototyping in the software industry too, and gives details: "Short cuts are taken to produce the prototype quickly at the expense of efficiency, robustness, generality and a good user interface." He points out too that "it is cheaper to correct misunderstandings and ambiguities in the prototype stage than after the final product has been built." (It is less embarrassing too.) This section describes how prototyping was used in designing the structure of the files and screen.

The restrictions [67 p. 481] that Gehani imposed were also imposed upon the MMI program:

"1. Instead of building a form definition translator, the form definitions would be hand translated.

2. A general terminal handling facility would not be provided ...

3. The simplest possible user interface (from a prototype builder's viewpoint) would be provided, e.g. no multiple page forms or multiple windows.

4. Only a simple error message scheme would be provided.

5. Use as many tools on the ... operating system as possible, paying regard to efficiency (or the lack of it) only when it really had a significant impact

on the system.

6. The system would not be implemented on a distributed system."

Gehani made the following decisions about file structure, and the MMI program followed his method:

> "It was decided to implement each form instance as one file. This was a natural and logical choice since this would simplify the implementation of several form operations since advantage could be taken of the operating system file operations." [67 p. 482]

The MMI project differs in that only one file was used per form, whereas Gehani uses seven files per form [67 p. 483]. The idea is to be able to transfer the file contents to the screen en bloc, and to store any information that the program might require in the form itself. Galitz points out, "screens should display only relevant information" [68 p. 8]. This makes it easier for the operator to find information on the screen, but the programmer must be forced into the discipline of not displaying extraneous material on the screen.

A decision was taken to limit the size of the screen. The 40-column mode of the Apple microcomputer was chosen in preference to the 80-column mode in order to present a smaller quantity of information on the screen. A packed array of fixed size was used to store the graphic characters and literal text which forms an exact image of the screen. The text contains help, a menu, or a data form. The packed array is stored as a file on diskette, in the final form ready for display on the screen. The size of the packed array was determined from the size of the screen and the size of the diskette blocks. The size of the

screen was 24 lines by 40 columns, and the size of the blocks was 512 characters. A screen image was therefore defined as being a packed array of 1024 characters, exactly two blocks big. The screen images are edited with the UCSD Pascal editor, avoiding the need to create a special-purpose editor.

When using the editor, the developer or maintainer sees the exact appearance of the image of the screen. This approach frees the programmer from the burden of having to count columns and rows, and allows "the programmer to specify the screen at a higher level" [69 p. 32]. The ease with which the screen images can be edited encouraged the building of "a prototype system ... to illustrate the feasibility of new ideas or design" [67 p. 480]. However, before use, the two-block header of the UCSD file, which contains information about the environment of the file, was stripped off the file in order to save space on diskette.

Upper case letters can be resolved at greater distances than lower case letters because of the greater angle subtended at the human eye. However, resolution is not of importance here, as all screens display characters that are sufficiently large for operators with near normal eyesight. What is of importance is readability. Real words (not random abbreviations) should be easily and quickly read. Words using lower case letters are more quickly assimilated because of the sensitivity of the human eye to the shape of the word. Consequently, the MMI project presents all text to the operator in lower case, with the usual capitalization rules of English being used when appropriate.

The program determines the type of each image (help, menu or form) by inspecting the text in the image. In each screen image is embedded the type of the screen in upper

case characters in the right hand half of the first line (context line) of text displayed on the screen (Fig. VI). These characters are used by the program to determine the ScreenType, each time the array is read from diskette. The left hand half contains the name of the node, so that the operator will know the present context. When the date and time are supplied by the OMC computer they are reformatted, and positioned in the top left and right corners of the screen, just inside the outer frame.

Down the left side of each screen image are the set of characters that the operator is currently allowed to type. See e.g. Fig IV. These characters are read out of the array and are subsequently used to check that the operator does not type an illegal character to move to a new frame. Thus the movements of the operator in the menu tree are strictly controlled by the contents of these packed arrays.

Provision was made for a rectangular portion of each form to be used in conversational [1 p. 14] mode, without disturbing the current context line or the escape options which are always displayed on the screen. The MMI program determines the position and size of the rectangle by examining the blank form (See e.g. Fig. V) each time it is read from the diskette.

This section has described how prototyping was useful in designing the structures of the files and the screen images. The section has explained how only relevant information is displayed. Help screens, menus and forms are kept i files in a format ready for display. The technique used allows easy specification of the screen at a high level. Lower case letters are chosen to display information to the operator. Only one file is required per form and the contents of the images have been indicated. The type of the screen image, the set of allowable

characters and (in the case of a form) the origin and dimensions of the conversational area are all stored as part of each image.

4.9 COMMUNICATING WITH SYSTEM X OR WITH THE OMC COMPUTER

A "user-friendly" man-machine language translator [5] successfully completed a trial [70] in the Cambridge Area. It was developed for the British Telecom System X, to allow administrative staff to exchange information more efficiently. The application of the translator to System X is identical to the MMI application. Similiarities include the application only for administrative staff, and not engineering staff; a "comfort indicator" or "activity indicator"; plain English output [5]; use of a 64K microprocessor with associated floppy diskette(s); programming in a structured high-level language; hierarchial menu structure; program completely resident in memory. Differences [70] include those listed in Table II.

Table II -
DIFFERENCES BETWEEN BRITISH TELECOM
AND SOUTH AFRICAN MAN-MACHINE LANGUAGE
TRANSLATORS

BRITISH TELECOM for SYSTEM X	S. A. POST OFFICE for SA128E
English mnemonics to English	French mnemonics to English
dual diskette (read & write)	single diskette (read only)
5 thousand pounds	<4 thousand Rand
Z80 microprocessor	6502 microprocessor
used to commissioning System X	not used for commissioning
written in CBASIC	written in Pascal
interpreted and optimized	compiled
interrupt driven	transaction driven

4.10 SUMMARY OF THE DESIGN OF THE SYSTEM

This chapter has described the design of the MMI, the interface to the operator, and the interface to the OMC computer. The Apple 2e and the UCSD Pascal environment was selected for the project. The chapter included the need to customize the Apple 2e, and continued with the need to design with change in mind, and adopt a definite programming standard. A prototype helped to settle the design of the file structure and screen structure. The designs of the microcomputers communicating with System X and with the OMC computer have been compared.

CHAPTER 5
IMPLEMENTATION

5. INTRODUCTION TO THE IMPLEMENTATION

This chapter deals with the conservation of the microcomputer resources. A prototype program to demonstrate the feasibility of communication between the microcomputer and the OMC computer is described. The data structures at the interfaces are examined, and this leads to a description of the key algorithm. The chapter includes detail of how the operator specifies data. The software development tools used are described briefly, as well as the development of the special character set. The chapter concludes with an explanation of the writing of the external documentation [1 p. 17] which helped the development of the program.

5.1 USE OF THE MICROCOMPUTER RESOURCES

Two constraints on processing were kept in mind: the amount of memory space available and the time available to execute real-time processing. It was found that the memory capacity was adequate and automatic diskette swapping was not necessary. This was mainly because textual information was kept as images on diskette.

The time available for the processing of data is limited. This limitation was solved by investigating the time-consuming statements, and avoiding the use of these during the reading of data from the OMC computer. In order to demonstrate the relative amount of time required to execute simple statements, a few typical Pascal statements were timed. The results obtained are shown in Table III. The results exclude the time taken by the driver [1 p. 17] program. As might be anticipated, the processing of data takes little time, but input-output processes take much longer. It is instructive to note the wide range of times measured, a factor that is usually ignored when programming in a higher order language.

Table III -
TIME TAKEN FOR 10 000 STATEMENTS

TIME	STATEMENT
1,4s	ch := 'A';
3,9s	L := length(st);
28,1s	insert('A', st, 1);
39,1s	st := copy('A', 1, 1);
41,9s	write('A');
61,4s	P := pos('A', 'A');
71,8s	st := concat('1', 'A');
146,6s	gotoxy(0,0);
159,5s	writeln;
2446,8s	page(output);

5.2 SPACE ON DISKETTE

In the final MMI program, the diskette is not used for recording, and is write protected for security. Additionally, the door of the drive may be sealed with masking tape

unless the microcomputer is to be used for other applications. The only time that the diskette is used is in booting the microcomputer and loading the turn-key program into memory. It is anticipated that this will be done once a day when the operator commences work. The MMI diskette contains files such as the bootstrap [1 p. 10], the operating system [1 p. 25], the system library [1 p. 34], the application program [1 p. 9] and configuration [1 p. 13] information. The space occupied by these have been analysed into classes in Table IV.

Table IV -
SPACE ALLOCATED TO CLASSES OF FILES
(IN BLOCKS OF 512 BYTES)

CLASS OF FILE	FILE SIZE
The bootstrap & directory	6 blocks
The operating system:	74
The system library:	63
The application program:	8
The help files:	28
The menus:	12
The forms:	28
The scripts:	18
The error messages:	16
Configuration text:	8
Spare:	19
Total:	280

Most of the object program [1 p. 25] appears in the system library, so that changes may be made to units without the need to recompile the complete application. The forms, menus and help files which constitute the bulk of

unless the microcomputer is to be used for other applications. The only time that the diskette is used is in booting the microcomputer and loading the turn-key program into memory. It is anticipated that this will be done once a day when the operator commences work. The MMI diskette contains files such as the bootstrap [1 p. 10], the operating system [1 p. 25], the system library [1 p. 34], the application program [1 p. 9] and configuration [1 p. 13] information. The space occupied by these have been analysed into classes in Table IV.

Table IV -
SPACE ALLOCATED TO CLASSES OF FILES
(IN BLOCKS OF 512 BYTES)

CLASS OF FILE	FILE SIZE
The bootstrap & directory	6 blocks
The operating system:	74
The system library:	63
The application program:	8
The help files:	28
The menus:	12
The forms:	28
The scripts:	18
The error messages:	16
Configuration text:	8
Spare:	19
Total:	280

Most of the object program [1 p. 25] appears in the system library, so that changes may be made to units without the need to recompile the complete application. The forms, menus and help files which constitute the bulk of

what the operator sees occupy a good proportion of the diskette. They are kept as files that can easily be changed, without the need for any compilation. A little space remains unused. If more space is required in the future, the available space can be increased by 8 blocks by using the 36th track, and another 32 blocks by placing the file SYSTEM.APPLE on the other side of the diskette.

5.3 COMPUTER-COMPUTER COMMUNICATION

A prototype program was written to show that the Apple II could communicate with the OMC computer in real time [1 p. 29]. The program demonstrated a critical feature of the final system, viz. that computer-computer communication was possible. Some subtle aspects of the interface were revealed at this stage. For example, it was realized that provided the operator was forced to stay in the continuation mode, a prompt would be received. This could then be used to delimit the end of a message from the host. Also at this stage, possible difficulties were identified. In particular, it was determined that no diskette activity, and little output to the screen could take place while a message was being received from the OMC computer.

If a "digital tape recorder" were available, the correct transaction between the terminal and the OMC computer could be "recorded" for subsequent study. Instead, the prototype program included the ability to "record" [35 p. 34] the transaction on diskette. Subsequent "playback" of the diskette and analysis of the recording faithfully reproduced the vocabulary and grammar of the transaction with the OMC computer.

The prototype program was used to record the required dialogue whilst normal transactions with the OMC computer were conducted. The microcomputer acted as a terminal, and

the program logged the transaction taking place on diskette. Thus the actual values of data items were captured on diskette. Possible misinterpretation of documentation by the programmer was thereby prevented. For example, the number of blank spaces between two fields is difficult to determine from the documentation, but could be determined precisely from the recording.

The recorded transactions formed the specification for the computer-computer interface. Each transaction was stored as a file of characters, called a script, with upper-case and lower-case letters being used to indicate the direction of transmission. See Fig. II. Error messages were catered for separately. Bits and pieces from the prototype program were used subsequently in the building up of the required translation mode (MMI). Once the prototype program had demonstrated the concept was viable, the hardware was ordered for the trial.

The following method proved to be a quick method of establishing computer-computer communication.

1) One microcomputer was used to write as much of the terminal software as was possible.

2) A second microcomputer of the same type was used temporarily to establish computer-computer communication at the lowest possible level (dumb terminal to dumb terminal). By using microcomputers of the same type, compatibility was more probable, and only one manufacturer's literature had to be studied. At this stage, communication card options were made identical, and a modem eliminator was used. By having the microcomputers adjacent, problems of human communication and telecommunication were eliminated. Documentation problems were sorted out at this stage. For example, the method

of turning the microcomputer into a dumb terminal is not correctly documented in the manual [71].

3) Software was installed in one microcomputer, and the second remained a dumb terminal. By typing the characters required by the OMC computer, the software was tested.

4) Refinements that suggested themselves were incorporated at this stage.

5) The software was then tested on site, without the OMC computer being emulated. At this stage, it was discovered that including too many features resulted in incoming data being lost. For example, logging to both the printer and to diskette caused data to be lost, but by logging only to the diskette, the crucial problem of communicating with the OMC computer was overcome.

5.4 DATA STRUCTURES

Simplistically, the MMI project can be regarded as consisting of two well defined interfaces, (the Computer-Computer Interface, and the Man-Machine Interface) joined by the software that is necessary to enable the two interfaces to be connected to each other. Both interfaces were defined algorithmically and from the data structure [1 p. 15] point of view, and the main program had to knit the two interfaces together.

The data structure of the existing computer-computer interface was studied. From this the algorithms necessary to handle the structure in a sequenced manner grew naturally. In order to determine what data structures were being used, the work of the operators was studied in detail. The correct method of performing this work was determined from

of turning the microcomputer into a dumb terminal is not correctly documented in the manual [71].

3) Software was installed in one microcomputer, and the second remained a dumb terminal. By typing the characters required by the OMC computer, the software was tested.

4) Refinements that suggested themselves were incorporated at this stage.

5) The software was then tested on site, without the OMC computer being emulated. At this stage, it was discovered that including too many features resulted in incoming data being lost. For example, logging to both the printer and to diskette caused data to be lost, but by logging only to the diskette, the crucial problem of communicating with the OMC computer was overcome.

5.4 DATA STRUCTURES

Simplistically, the MMi project can be regarded as consisting of two well defined interfaces, (the Computer-Computer Interface, and the Man-Machine Interface) joined by the software that is necessary to enable the two interfaces to be connected to each other. Both interfaces were defined algorithmically and from the data structure [1 p. 15] point of view, and the main program had to knit the two interfaces together.

The data structure of the existing computer-computer interface was studied. From this the algorithms necessary to handle the structure in a sequenced manner grew naturally. In order to determine what data structures were being used, the work of the operators was studied in detail. The correct method of performing this work was determined from

the staff working in the exchange who had the necessary practical experience to perceive the complex interlocking of the commands, and the subtleties of their use. The variables used were all dealt with as being Pascal strings, so that advantage could be taken of the string intrinsics of UCSD Pascal.

The recorded transactions were analysed using software tools as follows. (Each string of text that was separated from another by a new line or delimiter: "=", "<", ">", "-", "," " ", "+", will be termed a token.) All characters that delimit were converted to Carriage Returns. This had the effect of placing every token on a new line. This list was then sorted, and identical entries were cast out, yielding a sorted list of the unique tokens. It must be pointed out that all digits were examples taken from the recording, and that other values would appear in practice.

5.5 KEY ALGORITHM

The key algorithm of the final program was derived from the principal data structures. A statement of the algorithm [1 p. 18] follows:

1) Present a menu to the operator and get the choice from the operator.

2) Transmit the appropriate COMMANDline (first line of a transaction -- see Fig. II).

3) Show confirmatory text at a certain position on the screen.

4) Expect to receive a string (VERIFIEDline) from the OMC.

5) Determine if the VERIFIEDline contains an error message

by means of examining the first character ("*" for error),

6) Analyse the VERIFIEDline and split it into tokens, using "/" as a delimiter. Call further procedures to handle each of the tokens in turn.

7) Transmit the appropriate PARAMETERline (See Fig. II).

8) Show confirmatory text at a certain position on the screen.

9) Expect to receive a string from the OMC computer.

10) Determine if the ACCEPTEDline (See Fig. II) contains an error message b means of examining the first character ("*" for error). If it is an error, analyse the line using an error procedure and split it into tokens, using "/" as a delimiter. Otherwise analyse the ACCEPTEDline into tokens, using " ". Call utility procedures to treat each of the tokens in turn.

11) Expect to receive a string from the host computer.

12) Determine if the DATAline (See Fig. II) contains an error message by examining the first character ("*" for an error), then analyse the DATAline into tokens, using "/" as a delimiter for error messages, and " " for other messages. Treat each of the tokens in turn.

13) Expect to receive a string from the OMC computer.

14) Determine if the DATAline is an EXECUTEDline (See Fig. II). If it is, leave the procedure. Otherwise repeat from 11) to get subsequent DATAline(s).

5.6 CONVERSATIONAL MODE

When a form has been presented on the screen, the microcomputer slips into a conversational [1 p. 14] mode to:

1) Determine whether the operator wishes to specify a telephone number or an equipment number.

2) Ask the operator for the directory or equipment number of the line or telephone required. The operator types only the number, followed by a Carriage Return. If the operator makes a mistake in entering the number, backspacing enables single characters to be corrected.

3) Determine from the OMC computer if it is currently possible to deal with the number, and if not, display the reason on the screen (e.g. "Directory number does not exist").

4) Present the results of the transaction to the operator.

5) Offer the operator the possibility of repeating the transaction.

5.7 INTRODUCTION TO PROGRAM DEVELOPMENT

So far, this chapter has covered the conservation of the microcomputer resources such as memory, time and diskette space. A prototype program was used successfully to demonstrate communication between the microcomputer and the OMC computer, and to record transactions. Data structures at the interfaces were analysed and led to a description of the key algorithm. The last section included detail of how the operator specifies a line or a telephone number.

The chapter now continues with a brief description of the software development tools that proved to be useful, a description of the development of the special character set, and concludes with an explanation of the method of writing of the external documentation.

5.8 SOFTWARE DEVELOPMENT TOOLS

The UCSD Pascal environment (operating system, editor, compiler, assembler, linker, librarian and filer) were extensively used during the development of the software and in the preparation of the documentation. In addition, a set of software tools that had been integrated by the author proved to be most useful during the development of the software and documentation. These are described briefly below:

1) ARCHIVE was used to convert a text file to an ASCI file (a file containing only characters), and vice versa.

2) PRINT was used to make a printed copy of a series of text files, numbering the pages of each file and interpreting the embedded control codes for the printer.

3) TRANSLATE was used to convert specified characters in a file to other characters (e.g. upper case to lower case, or spaces to Carriage Returns).

4) LOCATE was used to print the lines in a text file that contained a specified pattern.

5) REPLACE was used to change occurrences of an specified pattern in a file with another pattern.

6) CAPITAL was used to capitalize the reserved words in each Pascal program.

7) FORMAT was used to format each Pascal program. It speeded up the preparation of programs and also proved useful in tracking down non-matching begin-end pairs.

8) SORT was used to sort text files into alphabetic order.

9) UNIQUE was used to filter out all occurrences of identical adjacent lines, passing only one instance of the lines. It was used to determine the unique words in a list, and was particularly useful after having used SORT.

10) VERIFY is a comparator [1 p. 12] that compares two text files for any differences. It was used to verify that two files were identical, and if not, to list any discrepancies e.g. to determine the differences between two versions of a file.

11) XREF was used to produce a cross reference list of the variables and procedures in a Pascal program. The list was examined e.g. for unused variables.

5.9 SPECIAL CHARACTER SET IN FIRMWARE

The character generator ROM in the Apple 2+ had been successfully replaced by a 2716 EPROM in the TRACE FAULTS project. Initial development, performed on an available Apple 2+ model, demonstrated that a form-drawing character set showed promise. Some changes to firmware would be necessary to the newer Apple 2e models to allow the drawing of forms on the screen. (No changes to the software were required to accommodate the conversion from an Apple 2+ to an Apple 2e. The program runs on either model. However

6) CAPITAL was used to capitalize the reserved words in each Pascal program.

7) FORMAT was used to format each Pascal program. It speeded up the preparation of programs and also proved useful in tracking down non-matching begin-end pairs.

8) SORT was used to sort text files into alphabetic order.

9) UNIQUE was used to filter out all occurrences of identical adjacent lines, passing only one instance of the lines. It was used to determine the unique words in a list, and was particularly useful after having used SORT.

10) VERIFY is a comparator [1 p. 12] that compares two text files for any differences. It was used to verify that two files were identical, and if not, to list any discrepancies e.g. to determine the differences between two versions of a file.

11) XREF was used to produce a cross reference list of the variables and procedures in a Pascal program. The list was examined e.g. for unused variables.

5.9 SPECIAL CHARACTER SET IN FIRMWARE

The character generator ROM in the Apple 2+ had been successfully replaced by a 2716 EPROM in the TRACE FAULTS project. Initial development, performed on an available Apple 2+ model, demonstrated that a form-drawing character set showed promise. Some changes to firmware would be necessary to the newer Apple 2e models to allow the drawing of forms on the screen. (No changes to the software were required to accommodate the conversion from an Apple 2+ to an Apple 2e. The program runs on either model. However

some changes were performed to allow the use of the newer Apple Super Serial Card, in place of the older Serial Interface Card.) Because the VIDEO ROM for the English version of the Apple 2e is not documented, it was necessary to dete mine experimentally if it too could be replaced by an EPROM.

The function of the VIDEO ROM in the Apple IIe, and the number of leads suggested that it could be replaced by a 2764 type EPROM. Inspection of the voltages on the ROM leads in a working Apple by means of an oscilloscope showed no conflicts with the pin-out of the 2764 EPROM. On the outside of the international version of the Apple 2e is a switch to select an alternate character set. This switch allows either the American National Standard Code for Information Interchange (ASCII) character set, or a national character set to be displayed. It was also discovered that this switch controls the most significant bit of the addressing of the VIDEO ROM, and that the BASIC commands INVERSE and NORMAL control the next most significant bit. Encouraged by these discoveries, the VIDEO ROM was removed from the Apple and an attempt was made to read the contents into an EPROM programmer. The bit patterns forming the characters were read successfully.

The bit patterns were analysed, modified as required, then written into 2764 EPROM's for insertion in the target machines [1 p. 35]. Although not used in the MMI project, inverse characters were also modified in case these are required in the future. The special graphics character set needed for the MMI project was designed to fit in with a copy of most of the existing ASCII characters. Only the alternate character set was modified, allowing the Apple to function conventionally if the character-set switch is set to the normal position.

The allocation of the graphic characters was performed in such a manner that Pascal programs would still be readable, as some of the initial development used an Apple II+ without the switch. This extension of the ASCII character set by substitution of the G0 set [72 p. 10] is recognized by the American National Standards Institute as being a possibility. The special graphics character set replaces the characters reserved for national use or characters that are infrequently used by the characters to draw forms. For example, Tee-pieces that point left and right are substituted for the square brackets in the ASCII set. The substitution occurs when the switch is operated and works for both the 40 column and the 80 column mode of the Apple 2e. The substituted symbols are given in Table V.

Table V -
ALLOCATION OF SPECIAL GRAPHIC CHARACTERS

ASCII CHARACTER	CORRESPONDING GRAPHIC CHARACTER
@ Commercial at	Cross piece
[Left square bracket	Tee-piece with leg pointing left
\ Reverse solidus	Tee-piece with leg pointing down
] Right Square bracket	Tee-piece with leg pointing right
^ Circumflex	Tee-piece with leg pointing up
{ Left curly bracket	Top left corner of a frame
} Right curly bracket	Top right corner of a frame
~ Tilde	Bottom right corner of a frame
` Grave accent	Bottom left corner of a frame
| Vertical line	Vertical line
_ Underline	Horizontal line

Forms created with the special character set were described as looking "very professional", possibly as a result of the angles in each character being intentionally rounded. The teletext standard makes provision for similiar symbols in a line drawing character set [73].

5.10 DOCUMENTATION

Brooks, cited by Meyers [74] is a strong advocate of writing the user manual first. It may seem unusual to write preliminary manuals before coding the programs. However, this approach was found to be of great value for the MMI user documentation [1 p. 36]. The unusual approach exposed open issues, crystallized issues that had been fluid, and laid down the majority of the content of the final documents.

The draft manuals that were written included an Installation Manual, a User Manual and a Maintenance Manual. System documentation [1 p. 34] consisted of interim reports at regular intervals on the progress of the MMI project. As Ingrassia points out, "one of the most common problems encountered in software management is the lack of visibility into the progress being made by each individual on a project" [75 p. 171]. Interim reports combatted this problem by making the progress of the project visible and formed the basis for this dissertation, which is now the definitive document on the project.

This remainder of this chapter explains the method used to write the manuals. First the need to determine the audience of the manuals is stressed. The ultimate purpose of the manuals is analysed, and the prior experience of the operators is examined. The knowledge of the operators after reading the manual is decided upon. A similiar analysis is performed for the staff who must install the terminals. Feedback on the manuals was encouraged in order that improvements could be made. Problems were investigated under the headings of clarity, terminology and emphasis.

Documentation for the operator was written with the

advice of Pakin in mind:

"To begin, identify the audience of the material:

* What are the audience functions?

* How will they use this material?

* What is their experience level with data processing?

* What are they expected to know after they have read it?" [76 p. 75]

Maynard recommends that "manuals should be aimed at users according to their function" [77 p. 16]. Accordingly, separate manuals were written for the operators and the installers of the terminal.

How the manual would be used was considered carefully. The operators use their manual as a means of instructing themselves in the use of the terminal, and later as a refer nce. They were assumed to have no prior knowledge of data processing. After reading the manual, the operator is expected to be able to:

1) test that the terminal is functioning correctly;

2) use the terminal to perform the administrative and service functions of the ETE which are required by the SAPO.

On the other hand, the function of the installers is to install the terminal for the operators. Their manual is used as a means of instructing themselves in the setting up of the terminal, and later as a reference. They too were assumed to have no prior knowledge of data processing.

After reading the manual, the installer is expected be able to:

1) unpack and install the terminal;

2) test that the terminal is functioning correctly.

User documentation was reviewed using the advice of Gudknecht: "Personnel who haven't been exposed to the equipment or documentation should review the material" [78 p. 116]. A preliminary version of the Installation Manual, and the Operator's Manual were shown to the Senior Technician and an operator at a SCC.

From observing the Senior Technician reading the manual, it became clear that it is necessary to have all figures next to the relevant text. It was noted that the term "monitor" has another meaning in telephony, and so the term "screen" had to be used instead. The Senior Technician experienced difficulty in identifying the diskettes and the diskette drive (he assumed this to be a modem). The diskettes are therefore clearly labelled and the drive is described in the manual. As the terminal may be used in an environment that is far from gentle and antiseptic, diskettes are supplied in a strong dustproof container. Each diskette is clearly marked with handling precautions and a backup diskette must be supplied. A label was designed to be attached to the diskettes. The text of the label starts:

"I AM A DISKETTE. Take care of me please..."

Ultimately it will be advisable to provide an English diskette and an Afrikaans diskette. It is possible to change languages at any stage by interchanging diskettes, without turning the terminal off, as the operating system

and MMI program are language independent. A possible problem could arise in the translation of the application to Afrikaans. Afrikaans uses four diacritical marks: the acute accent (´), the grave accent (`), the circumflex accent (^), and the diaeresis (¨). It was found that it is necessary to allow one line for the diacritical marks, one line for the text, and a blank line in order that the diacritical marks be associated with the correct line. Because diacritical marks occur infrequently in Afrikaans words, little problem is anticipated with their representation and interpretation.

A preliminary manual was written on the maintenance of the files for the terminal. The Maintenance Manual was written for a person assumed to have some prior knowledge of data processing. After reading the manual, the person will be able to maintain the data bases for the operation of the terminal. The person will not be expected to program in Pascal or change the fundamental operation of the terminal but is expected to be responsible for all routine maintenance of the files for the terminal.

To improve the clarity, terminology and emphasis of the Maintenance Manual, a preliminary version of the Maintenance Manual was shown to technical staff. Files are easily captured and edited with the software development tools, so the maintainer need only specify the input and output data items, how they are related and how they should be linked together. Only if major changes are required will it be necessary to recompile the source, since the data which is likely to change is contained in files.

5.11 SUMMARY OF IMPLEMENTATION

This chapter has described the microcomputer resources, prototype program, key data structures and

algorithms. It has listed the useful software development tools. The development of the special graphics character set has been described. Early writing of the external documentation served to clarify many cloudy issues.

algorithms. It has listed the useful software development tools. The development of the special graphics character set has been described. Early writing of the external documentation served to clarify many cloudy issues.

CHAPTER 6
PROGRAM CREATION

6. INTRODUCTION TO PROGRAM CREATION

This chapter considers the concept of program generators, and concludes that the MMI project does not display the economy of scale required to make the use of such a concept viable. Instead, the benefits derived from the use of the principles of program families, stepwise refinement [1 p. 33], and modules [1 p. 24] (usually units that are contained in a system library) are described. The units hide [1 p. 21] detailed information and provide some of the benefits cited for program generators. By using the same syntax for design and programming, the need to write a program to translate from design language to programming language was avoided. The man-machine language interpreter [1 p. 22] and the three parsers [1 p. 26] that are incorporated into the MMI program are introduced in this chapter. The flow of control [1 p. 19] by the main program and segmentation [1 p. 30] are mentioned.

6.1 REUSABILITY AND MAINTENANCE

A major concern during the program creation phase was the saving of effort in the program creation as well as in

CHAPTER 6
PROGRAM CREATION

6. INTRODUCTION TO PROGRAM CREATION

This chapter considers the concept of program generators, and concludes that the MMI project does not display the economy of scale required to make the use of such a concept viable. Instead, the benefits derived from the use of the principles of program families, stepwise refinement [1 p. 33], and modules [1 p. 24] (usually units that are contained in a system library) are described. The units hide [1 p. 21] detailed information and provide some of the benefits cited for program generators. By using the same syntax for design and programming, the need to write a program to translate from design language to programming language was avoided. The man-machine language interpreter [1 p. 22] and the three parsers [1 p. 26] that are incorporated into the MMI program are introduced in this chapter. The flow of control [1 p. 19] by the main program and segmentation [1 p. 30] are mentioned.

6.1 REUSABILITY AND MAINTENANCE

A major concern during the program creation phase was the saving of effort in the program creation as well as in

the maintenance phase. Batz et al draw attention to the reusability [1 p. 30] problem:

> "Each new software system is developed as if no system like it had ever been developed before, and no system like it will ever be developed again." [79 p. 78]

In view of this problem, the MMI project reused whatever it could. However, because of the lack of a precedent, all that the MMI project could reuse was the definition of the interface of an existing a screen-control unit [80].

The quote above implies also that a generally careless attitude towards future maintainence of systems exists. Although software does not wear out, it does still require maintenance. Maintenance is required to remove programming errors, to improve the efficiency of the program or the people using it, to adapt the program to new operating standards, or real working environments and to provide additional features [81 p. 403].

Maintenance costs are large and increasing. Since a large proportion of the life of a software product is devoted to the maintenance of the product, it makes sense to make every effort to create a product that is easy to maintain. Thus techniques are needed to reduce the level of expertise required of the user, reduce the work involved, enhance the efficiency of the program, improve the confidence of the user, and provide documentation for future use. Four techniques are available: customizable packages, very high level languages, programming by evolution and programming by examples [82]. A method midway between a good programming practice, and an automatic programming system was used in the MMI project.

Managing of complexity is a fundamental issue in all engineering disciplines. To manage complexity, the MMI project clearly separated input and output data items into files according to their function. Any data that might conceivably have to be changed was stored in a file. These files contain knowledge about the domain of the application. Each of these files can be manipulated with the UCSD editor and filer. This made it unnecessary to create an editor and filer.

The possibility of forming a program generator to create the application program was considered. A program generator appeared to have the following advantages: A generator would ease the task of writing the application programs, improve their accuracy and quality, and make subsequent maintenance easier.

Clarke has written a tutorial article on program generators [83]. He advocates program generation as being a better method of programming. It is means of constraining staff "to a narrow (and partly arbitrary) discipline". He defines a program generator as a "parameter driven utility program". That is, it is a useful program to which the programmer merely supplies parameters. It generates as its output "a high level language program". However, sections of the program may still need to be written directly in the high level language and inserted into the final program.

Program generators find their application where there is "considerable repetition of effort" [83 p. 49]. (Repetition is also one reason for the widespread use of libraries.) Beside redundancy in the algorithmic part of programs, there is also "structural repetition" [83 p. 49] in most programs. As Clarke puts it "the breakeven point will be of the order of only two or three uses per skeleton

-- a point reached or reachable in almost any single new application" [83 p. 49]. Unfortunately, the MMI project did not have the economy of scale to justify creating a program generator.

Clarke also points out that "multiple languages at different levels of abstraction (cause) language translation problems". In particular, "the interface between the design and the programming syntaxes" requires the support of "a powerful macro language" [83 p. 48]. For example, Ginsparg and Gordon [84] use Situation, User Action, and Effect triples in a Formal Design Language and translate these using a high level compiler to produce a program in the C language. By using the same syntax for design and programming, the MMI project avoided the need for this translation. The MMI project tried to make the procedures that appear in the interface between design and programming phases as powerful as possible. This simplified communication between the design and the program phases.

The easiest and quickest method of creating programs is to select a program which bears some resemblance to the new one and copy the parts which seem to be relevant and helpful. The advantage of this method is that the experience embodied in the program is explicitly transferred. Naturally this means that the program is always limited to a certain class of applications, because of the limited knowledge that can be incorporated into the program. However, the dangers of this method are detailed by Parnas [85 p. 97]. He warns: "one never modifies a completed program to get a new family member". On the contrary, it is "the well-developed but still incomplete representation that is offered as a contribution to the work of others." He advocates a style of writing programs "as if the operators and operands were 'built in' the language."

Using this methodology, the MMI program was built from a number of powerful units.

6.2 INTERPRETER AND PARSERS

The MMI program contains a man-machine language interpreter. This gets a character from the operator in order to determine what image (menu, help or form) to present to the operator, or it gets a string of characters from the operator into a form. he interpreter checks the character or string for conformity with the rules applying to such a character or string. In addition to the interpreter, the MMI program contains three parsers. The first parser analyses images that are stored on diskette to determine the type of the image, the set of allowable replies from the operator and (in the case of a form) the origin and size of the area in the form available for conversation with the operator. The second parser analyses a file containing the script (recorded dialogue) of the computer-computer transactions to be followed. Each line is analysed to determine its origin (operator or OMC computer). The third parser reads lines from the OMC computer, and determines what must be presented to the operator.

6.3 FLOW OF CONTROL

The flow of control of the program is under the control of the operator. After a help screen or a menu has been presented to the operator, SCGetCCh is used to get the reply in the form of a single letter from the operator. After an input form has been displayed, the procedure SCGetField is ultimately used to obtain a number from the operator.

The MMI program make use of precompiled units in order

to speed up compilation. Additionally, short sections of code that may have to be changed are kept in source form or in tabular form in files which are either included into the main program and used at compile time or read from diskette during the running of the program. In particular, the initialization section could have been swapped out of memory when its work was completed, but it was found that this was not necessary.

6.4 DECISIONS MADE DURING CODING

The following decisions were taken during coding.

1) Wherever possible, tables of data about the OMC computer are read from diskette to make changes possible without recompilation. (If new data types are introduced, it is necessary to add to the program and recompile.)

2) Previously the operator had to know the exchange (CEN) number of each exchange. Now the operator is asked simply for the exchange prefix (two or three digits), and the system looks up a table on behalf of the operator.

3) The first two (or three) digits of any telephone number entered by the operator are checked for validity, using the same table.

4) Entries that are obviously too low or high are trapped, using either length or range as a criterion.

5) Where possible, the range of permissible values is shown.

6) Equipment numbers are entered by component (Unit, Shelf and Port), to encourage the operator to attach some

meaning to the components.

7) Flexibility is allowed in the size of the screen area used for input-output (operator dialogue).

8) Little flexibility is allowed in the presentation of the line indicating the current context. (Signposts are standardized.)

9) Data is centralized on the screen for emphasis.

10) Except where both are required, the operator can specify equipment by giving either the telephone number or the equipment number. Because decision 6) is incompatible with the system deciding which number the operator has typed, the operator must first specify which number will be given.

In addition, the following simplifications were made during the coding phase to resolve the lack of a firm requirement specification.

1) Testing of subscribers' lines is not supported because work is currently in progress on the subscriber's line test equipment and interfaces.

2) The inclusion of multiple changes (to a maximum of 8 subscriber's lines) is not supported because it is likely that the response time by the OMC computer to such a request will be unacceptably long. Changes to single lines are supported, and will possibly avoid confusing the operator.

3) The operators will not be allowed to update schedules as this is considered to be the responsibility of the OMC staff.

6.5 SUMMARY OF PROGRAM CREATION

This chapter has defined the concept of program generators, considered their the field of application, their problems and dangers, and concluded that in the MMI program there is no economy of scale to make such a concept viable. Instead, use is made of the principles of program families, stepwise refinement, and units contained in a library. The units, their interfaces and their procedures are covered in the next chapter. These units hide information and provide some of the benefits cited for program generators. Reuse of the definition of one interface resulted in some economy of effort. By using the same syntax for design and programming, the need to write a program to translate from design language to programming language was avoided. The interpreter and three parsers created for the MMI program were introduced in this chapter. The flow of control by the main program and the lack of segmentation were mentioned, and the decisions and simplifications made during the coding phase were listed.

CHAPTER 7
DECOMPOSITION OF MMI INTO UNITS

7. INTRODUCTION TO DECOMPOSITION INTO UNITS

In order to decompose the program into units, the method of data flow analysis leading to a structure diagram [81 pp. 169,170] was found to be very helpful. The principle of information hiding was also used: common-use functions [47 p. 1949] were identified and hidden in units.

The system was decomposed according to the requirements. The MMI program presents the menus to the operator, and uses precompiled units to get characters from the operator, enter the menu tree, send the command or parameters to the OMC computer and deal with the replies.

At the last count, the MMI program contained more than 5420 lines of code (including comments and interfaces, but excluding help files, menus, forms, scripts and error message tables). Table VI gives the distribution of these lines of code to indicate the relative importance of the units.

TABLE VI -
NUMBER OF LINES OF CODE

TE	1333
SC	1191
DP	>850
GP	835
AN	559
MA	408
Main	>244
Total	>5240

The MMI program is required to perform six major functions, consequently, units were created to be the modules fulfilling these functions (See Fig. VII), and are first presented using a top-down [1 p. 36] approach:

MA: The main unit, MA, allows the operator to move around in a menu tree. The operator selects which form needs to be filled in by traversing the tree to an extremity (terminating node). At such an extremity, a form is presented for the operator to fill in.

AN: For each transaction required by the OMC computer, the analysis unit, AN reads an example or model script from a file. From this file, the AN unit determines what parameters must be sent. It then prompts the operator (in good English) for the parameters.

TE: All telecommunication with the OMC computer, such as the transmission of parameters, is performed by unit TE.

DP: The response from the OMC computer is analysed, and the unit DP displays the parameters to the operator in English.

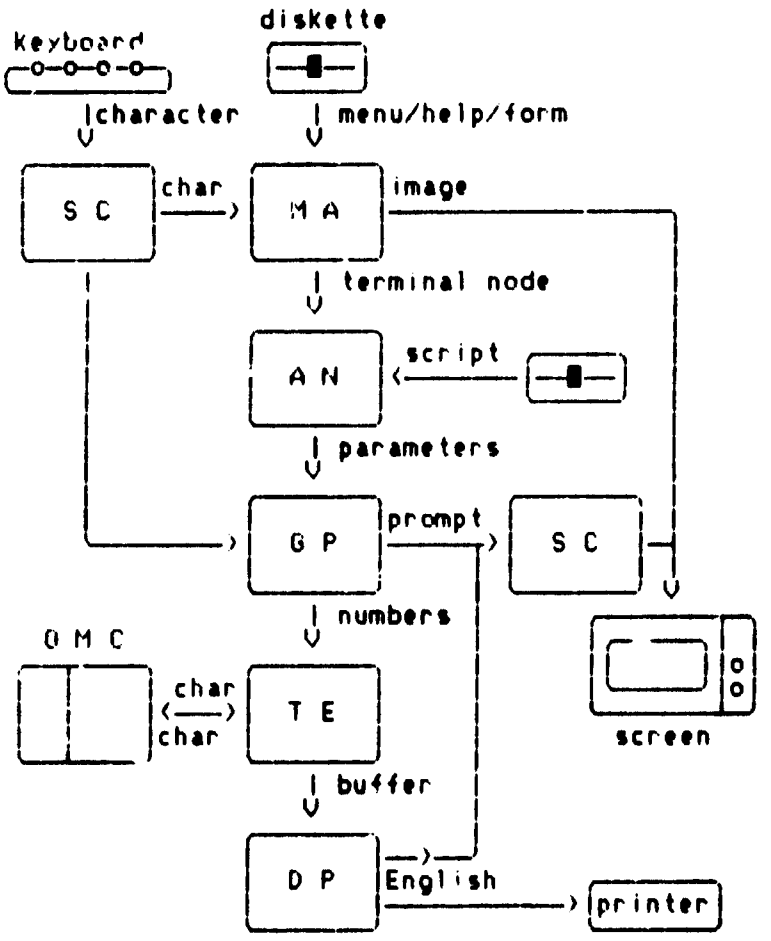

FIGURE VII -
FLOW OF DATA BETWEEN UNITS

GP: The unit called GP gets the parameters specific to telecommunication from the operator.

SC: The screen control unit, SC is responsible for advanced control of the screen and keyboard.

AS: All the above units were written in Pascal. In addition to these units, an assembly language [1 p. 9] unit called A3, provides a few niceties such as disabling control characters typed by the operator, and displaying characters in inverse lettering (black letters on a green or white background).

In creating the interfaces between the MMI program and the units, two approaches proved profitable. The first approach incorporated a large number of small procedures, to augment the available Pascal vocabulary. The second approach was to have one unit embodying one large procedure with a small interface. Typically, one of the parsers would constitute a single, large procedure. The philosophy in building units was to make them as powerful as possible, in order to provide useful constructs to the MMI program. The powerful units control the man-machine interface and the computer-computer interface, and the main program is responsible for overall control. Thus the coupling [1 p. 15] between the man-machine interface, the OMC computer, and the flow of control was minimized. With the exception of the unit AS, details of the units will be presented using a bottom-up [1 p. 11] approach.

7.1 SCREEN CONTROL UNIT, SC

Instead of specifying a new screen control unit, the specification for an existing interface [80] was used, and modified to meet the requirements. This approach took advantage of the thought that had gone into the initial

GP: The unit called GP gets the parameters specific to telecommunication from the operator.

SC: The screen control unit, SC is responsible for advanced control of the screen and keyboard.

AS: All the above units were written in Pascal. In addition to these units, an assembly language [1 p. 9] unit called AS, provides a few niceties such as disabling control characters typed by the operator, and displaying characters in inverse lettering (black letters on a green or white background,.

In creating the interfaces between the MMI program and the units, two approaches proved profitable. The first approach incorporated a large number of small procedures, to augment the available Pascal vocabulary. The second approach was to have one unit embodying one large procedure with a small interface. Typically, one of the parsers would constitute a single, large procedure. The philosophy in building units was to make them as powerful as possible, in order to provide useful constructs to the MMI program. The powerful units control the man-machine interface and the computer-computer interface, and the main program is responsible for overall control. Thus the coupling [1 p. 15] between the man-machine interface, the OMC computer, and the flow of control was minimized. With the exception of the unit AS, details of the units will be presented using a bottom-up [1 p. 11] approach.

7.1 SCREEN CONTROL UNIT, SC

Instead of specifying a new screen control unit, the specification for an existing interface [80] was used, and modified to meet the requirements. This approach took advantage of the thought that had gone into the initial

design of the unit, without copying or buying the module. The disadvantage of this approach was that the specification for the interface was ambiguous in parts, and decisions had to be made one way or the other to resolve the ambiguities. Ideas were also gleaned from other references [86].

The Screen Control unit, SC, moves the cursor and displays text only in a specified rectangular display area, called a port. The SC unit is responsible for advanced control of the screen and keyboard. All information on the screen and from the operator ultimately passes through the unit. The SC unit could be reused in any application, e.g. a non-telecommunication application, but in the MMI project it is used to provide support to the telecommunication oriented General Purpose unit, GP.

The SC unit allows the programmer to move the cursor one space up, down, left or right. It allows the cursor to be moved to any position that is specified relative to the port, or specifically onto the start of the next line, or to the top left corner. Strings of characters can be written on the current line and the unit does not allow the cursor to move outside the current port at any time during any of these operations, but saturates at the boundary. The current port may be changed or inspected under program control. The remainder of the current line, the whole of the current line or any specified line can be erased. The remainder of the port or the complete port can be cleared.

Finally, the SC unit boasts the following features:

1) The SC unit is general enough to allow information to the operator to be presented in any rectangular portion of the screen.

2) In keeping with the principle of information hiding, the SC unit keeps track of the cursor position. This implies that all communications to the operator must go through this unit. For this reason the additional procedures SCWrite and SCWriteln were written. Other additional procedures that were written are SCPeek and SCBell.

3) All output to the screen is through a UNITWRITE procedure in order to communicate speedily with the operator.

4) The screen control unit will not permit attempts to be made to access the screen area outside the PORT. (The information is shown inside the port, on the closest edge.)

5) The UCSD file SYSTEM.MISCINFO is used to allow for the possible use of certain external terminals or 80-column printed circuit boards.

6) Prefixed external terminals are not catered for. (These terminals require two characters e.g. to clear the screen.)

7) The possibility of allowing the programmer to change or switch off audible warnings was borne in mind.

8) Default values can be passed to the SC unit. For example, in a form, the previously specified value is used as a default. This is useful where the operator has to take a number of actions on the same exchange or telephone number.

9) The SC unit also contains the two keyboard interpreters, SCGetCCH (Screen Control's Get Checked Character), and

SCGetField (Screen Control's Get a Field), which get a character or a string of characters from the operator.

The procedure SCGetCCh repeatedly gets a character from the operator. The character is converted to upper case, and if it is not in the specified set of characters, the procedure beeps, otherwise it displays the character on the screen. The procedure returns the upper case character to the program. If more than one character is required, then the procedure SCGetField is used in place of SCGetCCh. The procedure SCGetField reads a string from the operator, one character at a time (using SCGetCCh), guaranteeing that the string is less than a specified size, and does not contain any invalid characters. It beeps if any character typed is not in the set ValidChars, and displays only printable characters. A default field can be passed into this procedure.

Three modes are available for the entry of a character or characters: DisplayMode, NewMode, and EditMode. DisplayMode is used by the programmer to display a string to the operator if no information is wanted from the operator. NewMode is used when no default value is available to show the operator. The prompt string is displayed, the maximum length of the reply is indicated, and the reply is accepted. EditMode is used to display a string to the operator, display a default value, and either accept the default value, or accept some other value typed in by the operator. If EditMode is chosen, the old field will be displayed, and can be updated. If the NewMode or EditMode is chosen, then the maximum permitted size for the field will be indicated by dots. If a reply from the operator is expected, the string that was typed may be corrected by means of backspacing until the Carriage Return key is pressed.

Timeouts are applied by the OMC computer in order to protect the use of its recources. The OMC computer times out a command after about 5 minutes of keyboard inactivity, and it becomes necessary for the operator to restart the command. If there is no keyboard activity after about 30 minutes, the session times out additionally, and it becomes necessary for the operator to log-on again. A counter can be set to an appropriate value to timeout if there is no activity from the operator, and thereby protect the OMC computer, or match similiar timeouts already present there. This provision was not used in practice because the MMI program works only in complete transactions, and restarts a new transaction every time.

7.2 PARSER TO DISPLAY PARAMETERS FROM THE OMC, DP

A large procedure called "DisplayParam" was written to parse the various French mnemonics that could possibly be returned by the OMC computer, and display them in expanded form in English. The procedure receives a buffer containing a maximum of 255 characters, writes the translation to the screen and printer, and returns a scalar to indicate the result (e.g. fail to find any parameter, error message found etc.). Except for the translation of error messages from the OMC computer, all translations are held in memory for quick access. The first step in the method of interpretation adopted was to quickly analyse the buffer for significant characters, and subsequently to treat each case separately. Wherever economic, translations are kept in separate files that are included at compile time. See Appendix 1 for examples.

The procedure "DisplayParam" is encapsulated in a unit called DP, and is the only procedure in the unit. "DisplayParam" itself contains several procedures. The procedures "DPWrite" and "DPWriteln" display a string

or a new line in the current port, and also on the printer. "FindSymbol" accepts a set of characters, and a buffer. It scans th. buffer, and returns an integer to indicate the first position where a character in the set is found in the buffer. A procedure called "AddChar" adds a character on the end of the buffer.

A procedure "WriteNum" was written to extract the numeric characters from a buffer. "DPWrite" is then called upon to display and print the extract. "Writ" is the name of a procedure which right-justifies text on the screen, based on the position of the right margin which must be supplied to it. If the string is too long to fit into the allotted space, it is first truncated. The procedures "WriteE" and "WriteG" call on "Writ", in order to show text and place an equals sign or a greater-than sign on the right margin, in preparation for the display of the value of a parameter.

"ShowErr" is a procedure which handles error messages from the OMC computer. It incorporates another procedure called "UseTable", which is used to read the files containing the English translations of the French error messages. The speed of looking up the messages can be improved, but the method used does not demand that the translations be kept sorted on diskette. To speed up the average time taken to translate a message, the translations have been partitioned into two files. "ShowErr" also incorporates the procedure "ReadNCompare" which reads a line containing an error number and the French message, and compares them with those received from the OMC computer. When both number and message match those received, a Boolean variable is set to "true". Unfortunately each French error was not allocated a unique number. This means that both number and message must be checked, and a backtracking algorithm was designed for the purpose.

A parser called "ShowValues" was created to decompose the French parameters into their components, viz. variables and their values. This parser incorporates a procedure "ShowNums" which accepts a buffer.

"ShowNums" incorporates a procedure called "Xlate", which translates French values to English values. "Xlate" in turn incorporates a procedure called "Cpy" which copies just the digits that exist in a buffer to another buffer by calling upon the procedure "AddChar". "Xlate" functions by searching the French string for the French mnemonic. If the mnemonic is found then "Xlate" continues to search the French string for an equals sign, and uses "WriteE" to show the English. "FindSymbol" is again invoked, this time to find the next non-space character. "Cpy" is used to determine the value of the French parameter, and "DPWriteln" is used to display and print it. "ShowNums" works by calling "Xlate" for as many times as there are French mnemonics. See the entries for "Xlate" in the Appendix 1.

"ShowValues" also incorporates calls to the procedure "ShowCOSes" which translates, displays and prints classes of service. It works in the same manner as "ShowNums". The complication here is that some classes of service are numbered, and this number can be given a meaning. In the MMI the meaning is displayed rather than just a number. Failing that, the number is simply displayed. The possibility of a class of service being numbered is noted in the dictionary by storing the suffix "n" at the end of the French mnemonic for the class of service. The translations relevant to "ShowCOSes" appear in Appendix 1 where calls to the procedure "translate" occur. The tremendous improvement in the intelligibility that the MMI brings can be gauged by comparing the text for the old and the new method of showing classes of service in Appendix 1. In order once

again to affect an improvement in speed, the dictionary containing classes of service has been divided into two volumes.

Some procedures specific to the SSC were developed, but not used because other projects are now in progress on the subscriber's line test equipment and interfaces.

7.3 THE COMPUTER-COMPUTER INTERFACE UNIT, TE

The protocol of the OMC computer was compared with the OSI model of the ISO [87], but too many layers of the OSI model were degenerate or missing for the model to be of use. The method of handling data from the OMC computer that was adopted was to place each character as it comes into a buffer. This continues until a complete message has been received before any processing is performed. This method posesses the advantage of performing a minimum of work while characters are being received, and is essential if no characters are to be lost.

Two data structures to hold the responses from the OMC computer were considered, but rejected:

1) A series of lines was discarded because the last line from the OMC computer does not terminate with a Carriage Return (the OMC computer supplies a Carriage Return BEFORE each new line).

2) A two dimensional array of characters was discarded because of the time taken to index in two dimensions, particularly as it might later be required to receive the characters at 1200 baud.

A one dimensional array of characters was considered, and chosen as the final data structure.

The computer-computer interface is different to the man-machine interface. In particular, the possible error situations that can arise there are different. Errors are detected at two levels: firstly, hardware errors are caught during the reading of characters from the OMC computer. Secondly, logical errors are trapped when the response from the OMC computer is something not anticipated in the normal course of affairs, such as an error message stating that a file of the ETE is full.

Hardware errors include parity errors, framing errors and overrun. These errors are commonly caused by faults such as noise on the data line or breaks in the connection. These may occur in either or both directions of transmission. Noise will usually affect individual bits, and this is detected by parity checking. Noise could affect individual characters when an even number of bits in a character are affected, and a plausibility check is applied to detect most occurrences of such errors. Breaks of short duration could cause characters to be dropped. These breaks are detected by checking that the reply from the OMC computer exists in the vocabulary. A long break could cause a complete reply to be lost, and this could be protected against by means of a timeout in a future version. The timeout could also cater for the cases where the data line goes faulty in the direction towards the OMC computer.

In the case of a hardware error, the program looks up a table of error numbers in memory, displays the corresponding message on the screen, asks the operator to press the space bar, and exits the transaction. In the case of a logical error, the program looks up a table of error messages to translate the message into terms that can be understood by the operator, displays the translation on the screen, asks the operator to press the space bar, and exits

the transaction. Every attempt has been made to create error messages that are specific, precise and meaningful. Obscure or cryptic messages have been replaced by friendly guidance that is user-centred.

The computer-computer interface was conveniently dealt with using a single unit called TE (for TElecommunication), containing a number of primitive procedures necessary to establish and maintain meaningful communication with the OMC computer. The following procedures were required for telecommunication between either an intelligent or a dumb terminal and the OMC computer. The names of the procedures give some indication of their function: activity, message, monitor, monitorln, error, WriteToHost, WritelnToHost, StringIt, InitCards, dialog, FixPassword, PromptSeen, LogOn, ReadlnFromBuffer, clock, ChangeSessionConnection, verify, SendCommand, and SendParameter. Details of these procedures have been omitted from this dissertation in the interests of brevity.

7.4 GENERAL PURPOSE UNIT, GP

The General Purpose unit, GP is oriented towards the beginning of the telecommunication application. It provides some additional services that SC overlooks, and also gets telephone oriented strings from the operator. The GP unit contains procedures that get four types of strings from the operator: GetTel gets a valid telephone number; GetA gets a part of an equipment number and checks it; GetU gets a valid Unit number; and GetEqt gets a valid equipment number from the operator. In order to obtain the information required for the checking of numbers, the GP unit has to initially get a file containing configuration information from the diskette.

The additional services that the GP unit provides are:

1) the ability to change the port,

2) move the cursor to the top left corner of the port and then clear the port,

3) shuffle the cursor out of the way so that it cannot be seen by the operator,

4) display an apology from the programmer when something goes wrong, and

5) display an error message if the operator enters a number incorrectly.

7.5 THE PARSER TO ANALYSE PARAMETERS IN THE SCRIPT, AN

The knowledge of what parameters must be sent to the OMC computer resides in the files of model dialogues that have been previously recorded. When the MMI program wishes to determine what parameters must be sent, it calls upon the ANalysis unit, AN. The program gives the name of the (terminal) node to the AN unit, and AN obtains the appropriate parameters from the operator.

The AN unit first opens the model file associated with the node. It then proceeds to parse the sample dialogue (script) stored in the file to determine what parameters must be sent. Finally, the unit calls upon the relevant procedures in the GP unit to get the required parameters from the operator.

The only procedure in the MMI program that was optimized for speed of operation was the procedure called "OpenScript", that opens the script file. This procedure

is called every time the operator has to fill in a form. The first version of it was slow, so refinements were made to decrease the time to execute the procedure. This was done in five stages:

1) The first version (with efficient initialization) was written and it was found to take 4,7s.

2) The critical code was re-written to be more efficient. The procedure now took 4,2s.

3) Range checking of subscripts was removed, reducing the time to 3,7s.

4) A built-in procedure was used in one location resulting in a better time of 3,1s.

5) Finally, a built-in procedure was used in another location resulting an acceptable time of 2,3s.

This illustrates that using UCSD Pascal it is possible to trade-off development time for speed of operation without having to resort to assembly language. In optimizing the procedure, the low-level procedure FillChar and the low-level function BlockRead built into UCSD Pascal were used. Original statements that proved to be too slow after testing were retained in the procedure as comments to explain the intention of the design. The "OpenScript" procedure accepts a string of characters containing the (terminal) node, and returns an array of 1024 characters containing the model dialogue requested. A convention was adopted that the name of the node, with the suffix "&.ASCI" would be the name of the file containing the script. The procedure "OpenScript" first appends this suffix to the name of the node, before attempting to open the file. The procedure then transfers one or two blocks of 512

characters from the diskette into the array.

A procedure named "analyser" subsequently accepts the array of characters from "OpenScript", and returns a Boolean array to indicate the parameters found. The procedure parses the array by using known landmarks such as the prompt character and continuation mode character which must exist in the array. Because of the large size of the array of characters, the author's technique of using a "LongString" as a buffer could not be used here, and the very low-level UCSD function "scan" had to be used, making it necessary to insert many comments into the procedure to explain it. The "analyser" procedure determines the command and parameter lines, and the parameter line is presented as a string to another procedure called "AnalyseParam". In "AnalyseParam", the parameter string is scanned for the presence of each of the possible French mnemonics, and if they exist, this fact is recorded in the Boolean array.

At this stage, the unit AN calls upon the procedure "GetValidSet" to determine what set of options is available to the operator. "ChangePort" is invoked to switch to the smaller display port used for conversation with the operator. Based upon the Boolean array previously determined, the operator is prompted for all the parameters required. For example, a procedure called "DnE" is used to obtain the directory AND equipment number, and a procedure called "DrE" is used to obtain either the directory OR the equipment number from the operator.

In order to reduce the impact of future changes in the interface to the OMC computer, the program makes no assumptions about the absolute position of any field from the OMC computer.

7.6 THE MENU PARSER UNIT, MA

The menu unit, MA, contains essentially two procedures, "transfer" and "MoveInTree". "MoveInTree" uses the character obtained from the operator. If the character is a "Q" (for Quit), the "Q" is removed from the end of the string called "node" to ensure that the previous image on the screen will be reinstated. Otherwise, the character is added onto the end of "node" to ensure that the next image will be presented. "MoveInTree" then calls upon "transfer" to effect the presentation of the correct image.

The procedure "transfer" simply allows the appropriate image, be it a menu, help screen or form to be transferred to the screen. However, it contains a parser to extract information embedded in the image. Initially, "transfer" opens the ASCI file specified by the string "node" supplied to it. If the file cannot be found, "transfer" assumes that the node has not yet been included on the diskette, calls upon "sorry" to apologise that the option is not yet ready, and removes the last character from the node in order that the previous image may subsequently be restored. Otherwise the image is read from diskette and written lock, stock and barrel to the screen, using the procedure "WriteScreen".

The procedure "WriteScreen" receives the image in the form of a packed array of characters, which it uses to update the full screen. Additionally, "WriteScreen" parses. It invokes the procedure "DetermineType" to find the type of the current screen image from the screen image itself, and it also calls upon "GetValidSet" to determine, from the image, the legitimate characters that the operator may type. If the image contains a form, "WriteScreen" additionally calls upon "GetIOWindow" to determine the size and origin of the new port.

"DetermineType" functions by inspecting the right half of the line containing the current context, and expects to find either the word "MENU", "FORM" or "HELP" there. The word may be spaced out thus: "M E N U". "GetValidSet" is more complex, and works by inspecting the left column of the image where the English mnemonics are kept (first letter of the option). For this procedure to work, every line in the file must contain the UCSD compression code DLE, and display 40 characters on the screen. The mnemonics are kept in the 6th column.

"GetIOWindow" is even more complicated and searches for the top left corner and bottom right corner of the blank rectangle that must be part of the image of every form. From this search it can calculate the position and size of the area reserved for conversational input from the operator. Naturally, "GetIOWindow" has to assume that the special graphic character set is in use.

7.7 ASSEMBLY LANGUAGE UNIT, PATCHES TO OPERATING SYSTEM

It was considered expedient to include 3 patches to the operating system in order to improve its integrity, and make it more suitable for the industrial environment in which it is to work.

7.7.1 PATCH 1 -- DISABLE CONTROL KEYS

Operators must be prevented from pressing control keys that interrupt the functioning of the program. This was effected by calling an 6502 assembly-language routine [88] that adds 128 to the value of the character intercepted from the keyboard. Calls to this routine take place at the beginning of the MMI program. At the same time, another 6502 assembly-language routine [89] was included to allow flashing characters to appear on the screen, in case this

ever became a requirement.

7.7.2 PATCH 2 -- AVOID A RETURN TO THE COMMAND LEVEL

The UCSD operating system returns control to the command level after a program has executed. The operating system was patched to repeat the MMI program forever and avoid showing the UCSD command line. This patch was devised by the author, and has not been published elsewhere before. Details appear in Appendix 2.

7.7.3 PATCH 3 -- AVOID CONTINUOUS SEARCHING FOR A DISKETTE

If the diskette used to boot the UCSD operating system is subsequently removed, the operating system searches continuously for it. To rectify this, the operating system was permanently patched to wait for the operator to respond instead of repeatedly checking for the presence of a diskette. Block 15 of SYSTEM.PASCAL was patched to replace the loop which checks the boot drive, with a call to the space-wait procedure instead. The latest version (1.2) of Apple Pascal also rectifies the problem.

7.8 SUMMARY OF ORGANIZATION INTO UNITS

Two different approaches to designing interfaces between the MMI program and the seven units proved fruitful.

The first approach, adopted with the screen control unit, involved the incorporation of a large number of small procedures to augment the available Pascal vocabulary. The second approach was to have one unit embodying a large procedure, with a small interface. Typically, this large procedure would be one of the parsers. The philosophy in building units was to make them as powerful as possible,

providing useful constructs to the main program. Coupling between the Man-Machine Interface, the computer-computer interface, and the main flow of control was minimized by dividing these functions among the units.

The MMI was partitioned firstly into the two interfaces (computer-computer or man-machine) and then into the two directions of transmission, and then, where necessary, into levels of service (powerful or primitive function).

Novel approaches used in the units included the use of a flashing character as an activity indicator, the employment of a LongString as a buffer, and the introduction of ASCII files to store complete screen images and scripts. Data useful to the program was stored in the screen images so that both the operator and program could use it. This means that the data need be updated in only one place.

CHAPTER 8
SYSTEM TEST PLAN

8. TEST-PLAN INTRODUCTION

Initially it was thought that testing [1 p. 36] could be performed once during the software life cycle, but it was realized later that testing has to be a continuous process during software development: "Verification and validation is now practiced over the full software life cycle" [90]. Some ideas for the test plan [1 p. 35] to test the MMI program were derived from the IEEE Std 829-1983 [91 p. 10].

8.1 DEBUGGING AND TESTING

Some advice from Weinberg [92] on debugging proved useful. By stripping out all commands from a module, attention was focussed on the source code and not the comments which can be misleading if incorrect. Also, the advice of Sand was followed during testing and debugging. He recommends testing a program "as it is being built in order to catch mistakes, bugs and design flaws as soon as possible". He puts forward the benefits of incremental testing:

1) "It is easier to test a small part of a program than to test the whole thing."

2) "Debugging is easier when it is spread out over the the program-development process."

3) "Design flaws should be detected as early as possible."

4) "The program 'works' at an early stage." [93]

Modules were tested by means of a driving program. Although some errors did creep into the driving programs, an equal number of errors were detected in the modules being tested. Time was saved overall because errors could only occur in one of two places (the driving program or the module being tested). Where both of these were small, it did not take long to determine the cause of the errors. Procedures were debugged immediately after testing so that they could be safely called by other procedures. Modules were integrated as soon as possible after debugging, to determine if any mismatches occurred.

Initial testing of the telecommunication primitives was performed by connecting two Apple microcomputers, with one of them emulating [1 p. 18] the OMC computer. The simulation could have been driven by means of a program to emulate the OMC computer, but the emulation was performed by typing on the computer. The introduction of a human into the loop introduced accidental variability which tested error recovery more fully.

Subsequent testing was performed at the OMC computer, with direct coupling to the OMC computer. The first application that was tested was the taking of a reading of a subscriber's meter. This test showed that when a character

was sent to the OMC computer, too much time was spent by the program before reading the character returned from the OMC computer, causing an overrun error message. This was corrected by moving time-consuming code, in this area of the program, elsewhere to a less critical position. In fact, this is the only time-critical area in the program.

8.2 TEST DATA

The test data [1 p. 35] for the MMI project can be viewed at two interfaces: the man-machine interface, where the requirements of the operators must be satisfied, and the computer-computer interface, where the OMC computer requirements must be met. The former interface was tested by selecting each item of each menu to ensure that each option functioned correctly. This exercised the latter interface because through the computer-computer interface, the MMI program is capable of communicating all aspects of the commands required for the application. The commands and sub-commands (parameters) at the computer-computer interface can be extracted from the documentation [34] and [94]. The documentation on these commands was verified experimentally by exercising each command and recording the resultant dialogue.

In the real time environment it is not possible to create error messages from the OMC computer at will, so the reaction to error messages was tested by means of emulating the OMC computer. For the same reason, the documentation on error messages [95] was not verified experimentally.

The microcomputer determines at every point whether it has received a positive or a negative acknowledgement from the OMC computer. For example, at certain points, the OMC computer indicates this by transmitting either the string "PROCESSING TXXXX EXC" or "PROCESSING TXXXXX REF"

respectively (XXXX or XXXXX being four or five characters indicating the task that is being invoked in the OMC computer).

Other error messages can occur routinely, depending upon the command being given, and the state of the ETE. It was assumed that an error message exists for every erroneous entry that can occur. The microcomputer (when programmed correctly) cannot make syntax errors, thus reducing the number of errors messages that need to be anticipated. Precautions were taken to anticipate and gracefully handle all errors messages. One example from the OMC computer is:

```
    PROCESSING TXXXX ACC
*   #R1216/C031-0000
    NOT AVAILABLE
    XX-X-XXX
```

This means simply that this equipment number has already been allocated, and that the operator should choose another equipment number. Appendix 3 lists the error messages and their translations by the MMI.

8.3 FEATURES TESTED

As Jensen et al. put it, "It is convenient to identify two kinds of users in the system -- the ultimate user, who sees the necessity of creating or purchasing a software product to solve a problem, and the operators of a system, who will use the software product in their day-to-day work" [81 p. 99-100]. For the MMI project there are two sets of users whose requirements had to be satisfied: the SAPO (for functionality) and the operators (for convenience). The features required by the SAPO we e the operational functions for subscriber administration. The features

required by the operators are contained in the thirteen points arising out of the interviews with the operators in Chapter 2.

8.4 APPROACH TO TESTING

Myers believes "Testing is the process of executing a program with the intent of finding errors." [96 p. 5] This was the approach taken during the testing of the MMI program. The advice of Branstad et al. proved invaluable: "Test pieces and then aggregate them." [97 p. 30] During the development of the MMI, each module and option was tested as exhaustively as was possible.

As the MMI program must run in a endless loop, there is no easy way of stopping a test prematurely. This was solved by including an escape mechanism in the error handling procedure. When the operator makes an error, an error procedure is invoked. It is then possible for the operator to correct the error. By including the ability to exit the program by pressing the ESCAPE key at this point, it became easy to leave the program when testing it: a deliberate mistake invokes the error procedure and pressing ESCAPE ends the test. Thus this technique provides an easy way of interrupting the program during testing.

A minimal useful subset [47 pp. 1963-1964] was developed initially. The path required to be complete before a minimal subset could be tested was identified from the data flow diagram and priority was given to completing this path. Priority was also given to completing the software necessary to allow the system to perform one task with generality. Other tasks were subsequently added by increasing tables, and by adding scripts.

8.5 SYSTEM TEST CASES

The system testing [1 p. 35] required the use of a port on the OMC computer, modems, a data line to the OMC computer, the documentation, and the microcomputer. The terminal was prepared according to the Installation Manual and all operations described in the Users Manual were performed. All options available to the operator were exercised by the test cases. Each option was entered in turn by pressing the first letter of the command required. (However, the help option is selected by pressing "?".) When a form was reached, all the numbers requested by the program were entered into a form on the screen. The data items comprising the display were checked to see that each was displayed correctly. Other tests were performed to check the integrity of the system.

The first working version of the Man-Machine Interface program was demonstrated to an OMC specialist, and representatives of the company supplying the SA128E ETE. The specialist suggested that the link to the OMC be tested for continuity at the beginning of the program. This and other refinements such as the removal of dead wood, removal of timeouts, the correction of bugs, more efficient use of diskette space and computer time are being incorporated into the next version of the program.

8.6 TEST CATEGORIES

Several considerations need to be taken into account during testing [96 p. 112-118]. These include the testing of the features, the usability, the security, the performance, the storage requirements, the configurations, the compatibility, the installability, the documentation and the lack of clashes with the previous operating procedures. The questions below can be asked after the installation

phase:

1) What individual features or facilities are not provided ?

2) Usability test -- Check the following human factors:

2.1) Is the user interface appropriate to the level of the operators ?

2.2) Does all the input mean something to the operator ?

2.3) Are all error messages straightforward ?

2.4) Are there inconsistencies in: Input and output syntax, conventions, semantics, format, style or abbreviations ?

2.5) Are there a minimum of unused options ?

2.6) Is there always an immediate acknowledgement ?

2.7) Is the system easy to use ?

3) Security testing -- are all control characters trapped ?

4) Performance testing -- does the system perform satisfactorily at 300 baud ?

5) Storage testing -- Is there an adequate reserve of

5.1) space on the diskette ?

5.2) memory space ?

6) Configuration testing -- Can the configuration be changed for different ETE's ?

7) Compatibility testing -- What hardware and software incompatibilities with the OMC computer can be found ?

8) Installability testing -- What hitches were experienced during installation?

9) Reliability testing -- For how long can the system run continuously without being reset ?

10) Recovery testing -- Does the system recover from errors?

11) Serviceability testing - How long does it take to make a modification to the system ?

12) Documentation testing

12.1) What inaccuracies can be found in the documentation ?

12.2) What contradictions can be found in the documentation ?

12.3) Where is the documentation unclear ?

12.4) What has been omitted from the documentation ?

13) Procedure testing -- What clashes with current operating procedures can been found ?

14) Test procedure -- What troubles were experienced during testing ?

8.7 TEST PLAN SUMMARY

It was found that testing has to be a continuous process during software development. The data that had to be tested was conveniently viewed at the Man-Machine Interface and the computer-computer interface. Simulated errors were injected into the computer-computer interface to test the error response. The operation of the MMI program was tested for the administration of subscribers. The features required by the operators were derived from the interviews with the operators.

The approach of building pieces, testing them and then joining them together proved most fruitful, as did the approach of first building a minimal useful subset. All options available to the operator were exercised by the test cases and the numbers requested by the program were entered in order to test the MMI exhaustively. Several considerations that need to be taken into account after installation were listed. These include testing of the features, usability, security, performance, storage requirements, different configurations, compatibility, installability, documentation lack of clashes with the previous operating procedures

8.7 TEST PLAN SUMMARY

It was found that testing has to be a continuous process during software development. The data that had to be tested was conveniently viewed at the Man-Machine Interface and the computer-computer interface. Simulated errors were injected into the computer-computer interface to test the error response. The operation of the MMI program was tested for the administration of subscribers. The features required by the operators were derived from the interviews with the operators.

The approach of building pieces, testing them and then joining them together proved most fruitful, as did the approach of first building a minimal useful subset. All options available to the operator were exercised by the test cases and the numbers requested by the program were entered in order to test the MMI exhaustively. Several considerations that need to be taken into account after installation were listed. These include testing of the features, usability, security, performance, storage requirements, different configurations, compatibility, installability, documentation and lack of clashes with the previous operating procedures.

CHAPTER 9
CONCLUSIONS

9. SUMMARY OF PROJECT

A survey of the literature has shown a general need for work in the fields of business and telecommunications to produce better man-machine interfaces. Guidelines for designers working in these fields have been brought together. Following these guidelines, interviews were held, revealing the real needs of operators. A plan was conceived and implemented to improve a particular man-machine interface in all practical ways.

In implementing the plan, some discoveries were made. Amongst other things, it was discovered that two-way communication could be established between a microcomputer and the OMC computer. Help information can be stored on diskette, and yet can be quickly transferred to the screen of the microcomputer. Forms can be easily represented on the screen by installing firmware containing the graphics character set to draw forms. A flexible tree-structure can be used to access help-screens, forms and menu-screens in a uniform way.

The MMI project has shown that local problems can be

corrected or avoided: Inflexible terminals can be replaced, and problems with the operating system of the OMC computer can be circumvented. No longer will it be necessary (for administrative staff of the SAPO) to experience difficulty with the mnemonics that were required to command the SA18E ETE.

Cryptic mnemonics and error messages returned from the OMC computer can be intercepted and presented in a user-friendly fashion.

9.1 GOALS MET

The goals of the project were met, within the limits noted below. The MMI for the operators:

1) presents information in a format that is easily understood;

2) tells them clearly what to do when something goes wrong;

3) tells them directly what has been done successfully;

4) is well documented, in a place where the information cannot be lost;

5) can be modified to suit the users requirements;

6) is menu driven;

7) is in English;

8) spells everything out in full but is still brief and to the point;

9) holds the intelligence required to make complex tasks

simple;

10) is rapid locally, but still bound by the response time of the OMC computer;

11) is lucid;

12) is brief and to the point;

13) requires a minimum of keystrokes to achieve an end.

9.2 FUTURE WORK

Further improvements can be made to the MMI using Apple Pascal Version 1.2, which is now available. This version includes improvements to overcome bugs that existed in version 1.1. For example, the initialization sections of nested units were (incorrectly) executed in the reverse order [53 p. 77]. An additional feature of version 1.2 is the return of the status of the remote port [53 p. 57]. This means that the program need no longer wait if the Super Serial Card is busy, and this opens the gate to allow the program to check if the link to the OMC computer has been broken. Version 1.2 also runs on the Apple IIc model [98].

The project was not translated to Afrikaans, but this is a purely mechanical process, and will be simplified by the fact that the text is stored in separate files which can be readily edited. In addition, operator commands that were excluded from the initial version such as those dealing with e.g. PABX's can be incorporated in subsequent versions. Additional features can also be incorporated on request. The number of keystrokes required has been drastically reduced and this must improve the speed of operation. However, the particular approach taken could

never effect any improvement to the processing time taken by the OMC computer, but some further improvement can be effected by upgrading the operation to 1200 baud.

At the OMC, while the microcomputer was not being used for debugging purposes, the exchange staff made use of it. They found it to be a great timesaver when a large number of changes had to be made to the files of the OMC computer to cater for a change in a large number of telephone numbers. The critical line in the BASIC program was PRINT "AFCMD=TR, FICH=EQU(26), NUME="; IX; CHR$ (58); CHR$ (13) where IX was being incremented up to 1023. The typing of thousands of characters was thus avoided, and the change-over was speeded up. This suggests that further work in providing a number of tools to help exchange staff to accept and maintain the ETE could be of benefit.

In the wider context, the cognitive issues in man-machine interactions still remain unexplored [18 p. 333], and much work is needed to validate the new user interfaces being built using graphic symbols and mice.

9.3 FINAL REMARKS

A major problem in the SAPO is lack of trained staff. Anything such as the MMI to help to relieve the situation would be welcome. Staff welcomed the new terminal and wanted to know when it would be installed.

In conclusion, this research project has shown that a plan to improve a section of the existing man-machine interface by using a microcomputer in place of a VDT is reasonable and practical.

In future, may we always regard "the user as a human being" [99].

APPENDIX 1 - LIST OF MAJOR TRANSLATIONS

```
Xlate('NE',     'Equipment No.');
Xlate('ND',     'Directory No.');
Xlate('TAX',    'Meter reading ');
Xlate('NROB',   'Remote Test Unit No.');
Xlate('AFUR',   'U.R. No.');
Xlate('NBFAU',  'No. faulty items ');
Xlate('TFAU',   'No. of faulty calls ');
Xlate('NEQPT',  'No. items of equipment ');
Xlate('NDG',    'Group No.');

XlateCOS('CAT',   'CLASS OF SERVICE Category');
XlateCOS('TY',    'CLASS OF SERVICE Type');
XlateCOS('MAR',   'CLASS OF SERVICE Mark');
XlateCOS('STAT',  'STATE OF PHONE Line');

translate('ABS',    'Subscriber absent');
translate('ART',    'Follow-me');
translate('ARV',    'Wake-me service');
translate('ATT',    'Camp on busy');
translate('CAD',    'Protea/Disa phone');
translate('CAFn',   'Catastrophe facility');
translate('CAMn',   'Malicious call tracing');
translate('CNI',    'Changed Number Interception');
translate('COF',    'Conference facility');
translate('CTn',    'Metering Category');
translate('DFn',    'Spare Class Of Service');
translate('DOP',    'Operator line');
translate('EMG',    'Emergency line');
translate('ESS',    'Circuit Testing Equipment line');
translate('FDn',    'Detailed billing');
translate('FLA',    'Recall Button');
translate('FXA',    'Line is P.G.');
translate('GABAn',  'Test threshold');
translate('IAM',    'Malicious call tracing on');
```

```
translate('INT',    'Service Interception');
translate('KLA',    'Multi-Frequency phone');
translate('LAI',    'Hot-line');
translate('LIBR',   'Line Free');
translate('LFL',    'Line Faulty');
translate('LNP',    'Non-preferential line');
translate('LOP',    'Operator''s line');
translate('LSS',    'Priority line');
translate('LST',    'Non-metering line');
translate('MIX',    'Bothway line');
translate('MNP',    'Permanent monitoring');
translate('MNT',    'Temporary monitoring');
translate('MOB',    'Meter Obs.');
translate('MOD',    'Changes to services allowed');
translate('NAn',    'Short code dialling');
translate('ODDA',   'Normal sub Obs. I/C');
translate('ODDD',   'Normal sub Obs. O/G');
translate('ODGA',   'PABX group Obs. I/C');
translate('ODGD',   'PABX group Obs. O/G');
translate('OLDX',   'High traffic normal sub. Obs.');
translate('OLGX',   'High traffic PABX group. Obs.');
translate('OSDA',   'Normal sub meter Obs. I/C');
translate('OSDD',   'Normal sub meter Obs. O/G');
translate('OSGA',   'PABX group meter Obs. I/C');
translate('OSGD',   'PABX group meter Obs. O/G');
translate('PPR',    'Coin phone');
translate('RVT',    'Follow-me');
translate('SPA',    'Outgoing calls only');
translate('SPB',    'Incoming calls only');
translate('SPS',    'Special subscriber');
translate('SR0',    'Unrestricted Service');
translate('SR1',    'Follow me restriction');
translate('SR2',    'Local calls only');
translate('SR3',    'No national or international calls');
translate('SUSn',   'Service Suspended');
translate('TLC',    'Concentrator line');
translate('TTX',    'Subscriber''s private meter');
translate('ZGn',    'Geographic Zone');
```

APPENDIX 2 - PATCH 2 TO SYSTEM.PASCAL

The source for System.Pascal must be similiar to the following:

```
page(OUTPUT);
write(
'Command: E(dit, R(un, F(ile, C(omp, L(ink,/
 X(ecute, A(ssem, D(ebug,?[1.1]');
CASE
  Getch OF
    'E': Runsys('Editor');
    'F': Runsys('Filer');
    'L': Runsys('Linker');
         .
         .
         .
END (*case*);
```

If the Pascal source program were available, it would be possible to avoid the prompt at the command level by patching the system as follows:

```
(***page(OUTPUT);
write(
'Command: E(dit, R(un, F(ile, C(omp, L(ink,/
 X(ecute, A(ssem, D(ebug,?[1.1]');
```

```
CASE
  Getch OF
    'E': Runsys('Editor');
    'F': Runsys('Filer');
    'L':  ***)

     Runsys('Linker');
       (***.
        .
        .
END ***);
```

The appropriate point in the file system.pascal was determined by looking for the command line string:
Command: E(dit, R(un, F(ile, C(omp, L(ink,/
X(ecute, A(ssem, D(ebug,? [1.1]

The old P-code at this point in system.pascal follows:

```
123 (205) CXP (0, 39)  ; CALL EXTERNAL - (PROMPTLINE)
126 (236) SLD05 (0, 0) ; SHORT LOAD GLOBAL - (BADCMD)
129 (205) CXP (0, 41)  ; CALL EXTERNAL - (GETCHAR)
```

It is possible to avoid the prompt at the command level by patching the P.code as follows:

```
123 (215) NOP (215) NOP (215) NOP  ; NO OPERATION
126 (236) NOP (215) NOP (215) NOP  ; NO OPERATION
129 (205) NOP (215) NOP (69) 'L'   ; FOR L(INKER
```

If the main program is installed as "system.linker", it will not be possible to exit from this program, or to halt the system.

APPENDIX 3 - LIST OF ERROR TRANSLATIONS

OLD MESSAGE	NEW MESSAGE
0022 various	Link busy: phone OMC
0025 SUBSCRIBER IN PERMANENT CONDITION	Line PG: send faultsman
0025 NOT KEYWORD SUBSCRIBER	Not a MF phone: check number
0025 NOT CALL BOX SUBSCRIBER	Not a coin phone: check number
0025 NOT REMOTE CHARGING SUBSCRIBER	No private meter: check number
0026 RESPONDER ERROR	Test Unit fault: phone OMC
0055 various	Translator error: phone OMC
0080 001/EXCHANGE ERROR	Subs meter pulse lost: repeat/OMC
0080 002/TELEPHONE NOT LIFTED	Phone is now on hook: call again
0195 01/ACCESS TO FILE INCORRECT	Exchange fault: phone OMC
0195 02/SATURATED FILE	Exchange file full: phone OMC
0257 various	OMC error: phone OMC
0258 various	Password error: phone OMC
0700 NAC	Equipment faulty: phone OMC
0734 01/UNEQUIPPED RESPONDER	No Test Unit for this no: phone OMC
0734 02/BUSY RESPONDER	Test Unit busy: wait or reset it

0734 03/RESPONDER ERROR
Test Unit fault: reset it

0734 04/BUSY SUBSCRIBER
Sub's Line is busy: try later

0734 05/CONNECTION ERROR
Faulty test connection: try again

0734 06/NO FREE CHANNEL
Test connections busy: try later

0735 ND=xxxxxxxx UNEQUIPPED
No such number: check number

0736 NE=xxx-x-xxx UNEQUIPPED
No such equipment: check number

0737 01/NROB=xx/UNEQU) RESPONDER
No such Test Unit: check number

0737 02/NROB=xx/BUSY RESPONDER
Test Unit busy: wait or reset it

0737 03/NROB=xx/.. PONDER ERROR
Test Unit fault: reset it

0742 01/NROB=xx/RESPONDER ERROR
Test Unit fault: reset it

0742 02/NROB=xx/INCORRECT CALIBRAT
Test Unit fault: phone OMC

0743 AFUR=xxx/NAC
Error while testing: phone OMC

0744 various
OMC error during test: phone OMC

0746 various
No equipment for this no.: check

0747 02/AFUR=xxx/INCORRECT UR TYPE
1st 3 digits wrong: check

0748 01/STOP OF SYSTEMATIC TESTS ON
check results

0748 02/TESTS INTERRUPTED.....
Error during test: phone OMC

0749 01/BUSY SUBSCRIBER
Test this one later

0749 02/UR NAC
Unable to test: phone OMC

0749 03/CONNECTION ERROR
Unable to test: phone OMC

0749 04/NC FREE CHANNEL
Unable to test: phone OMC

0749 05/SUBSCRIBER IN PERMANENT COND.
Unable to test: phone OMC

0749 06/OC INTERCHANGE ERROR
Unable to test: phone OMC

0749 07/INCOMPLETE MEASUREMENT
Unable to test: phone OMC

0751 various
No equipment for this no.: repeat

0752 various
Exchange error: phone OMC

0808 ABORT OF PROCESS WITH RECOVERY
ERROR: repeat last work/phone OMC

0808 PROCESS ABORT WITHOUT RECOVERY
ERROR: repeat last work/phone OMC

1102 various
Translator full: phone OMC

1104 TOO MANY MODIFIED ELEMENTS
Not so many: repeat with less no's.

1216 005/various
Incompatible choice: try another

1216 006/various
Inconsistent choice: try another

1216 007/various
Incorrect choice: try another

1216 008/various
Choice not allowed: try another

1216 009/various
Choice already used: try another

1216 010/WITHDRAWN DISCRIM. MISSING
This class doesn't exist: check

1216	011/ADDED DISCRIM. PRESENT	This class already exists: check
1216	022/various	This doesn't exist: check
1216	023/various	This already exists: check
1216	031/various	This is not available: check
1216	032/various	This is not assigned: check
1216	047/NON-CALLABLE LINE WITH COS	This class can't be called: check
1216	048/NON-CALLABLE LINE	This line can't be called: check
1216	052/various	Inconsistency: check
1216	053/various	Inconsistency: check
1259	018/SECTOR NUMBER OVERFLOW	Translator error: phone OMC
1267	various	Inconsistent choice: try another
1274	various	Inconsistent choice: try another
1278	various	Inconsistent choice: try another

1216 011/ADDED DISCRIM. PRESENT
This class already exists: check

1216 022/various
This doesn't exist: check

1216 023/various
This already exists: check

1216 031/various
This is not available: check

1216 032/various
This is not assigned: check

1216 047/NON-CALLABLE LINE WITH COS
This class can't be called: check

1216 048/NON-CALLABLE LINE
This line can't be called: check

1216 052/various
Inconsistency: check

1216 053/various
Inconsistency: check

1259 018/SECTOR NUMBER OVERFLOW
Translator error: phone OMC

1267 various
Inconsistent choice: try another

1274 various
Inconsistent choice: try another

1278 various
Inconsistent choice: try another

REFERENCES

[1] IEEE, "IEEE Standard glossary of software engineering terminology," New York: IEEE, Std 729-1983, Feb. 1983.

[2] CCITT, Study Group XI, "Report on the meeting held in Geneva from 6 to 16 April 1981," (Report no. R 1.), Geneva: Int. Telecommun. Union. CCITT, May 1981.

[3] L. SLY and W. HOUGH (Ed.), "Potelin's APLARM for SA128E's," (Postal), SAPO staff newspaper, vol. 15, no. 3, p. 10, Nov. 1984.

[4] M. BENEDETTI and C. LANZINI, "Toward a new man-machine interaction in SPC switching systems," in (1982 International Zurich Seminar on Digital Communications. Man-Machine Interaction), Zurich, Switzerland, New York: IEEE, 1982, IEEE Catalog no. 82CH1735-0, pp. 269-274, 9-11 Mar. 1982.

[5] T. D. NATHAN and D. JOHNSON, "Communicating with System X," (British Telecom J.), ISSN 0260-1532. pp. 2-3, Winter 1982/83.

[6] -------- , "A communication program for on-line systems and its use in man-machine language translation work," (Br. Telecommun. Eng.), ISSN 0262-401X. pp. 198-201, vol. 2, no. 3, Oct. 1983.

[7] L. TESLER, "Enlisting user help in software design," (SIGCHI Bulletin), ACM Special Interest Group on Computer and Human Interaction. vol. 14, no. 3, pp. 5-9, Jan. 1983.

[8] H. SIMPSON, "A human-factors style guide for program design," (Byte), ISSN 0360-5280, vol. 7, no. 4, pp. 108-132, Apr. 1982.

[9] T. CAREY, "User differences in interface design," (Computer), ISSN 0018-9162, pp. 14-19, Nov. 1982.

[10] K. LANTZ, "Uniform interfaces for distributed systems," (Ph. D. thesis), The University of Rochester, Dept. of Computer Science, 1980.

[11] R. W. SWEZEY R. W. and E. G. DAVIS, "A case study of human factors guidelines in computer graphics," (IEEE Computer Graphics and Applications), ISSN 0272-1716, vol. 3, no. 8, pp. 21-30, Nov. 1983.

[12] B. K. HILL, "Applied ergonomics -- guidelines for the design and layout of Visual Display Terminals, workspace and physical work environment," in (Visual Display Terminals: The Human Factor), Conf. on Visual Display Terminals: The Human Factor, Pretoria: S.A. Bureau of Standards, Nov. 30 1983.

[13] E. B. JAMES, "The user interface: how we may compute," in (Computing Skills and the User Interface), M. J. COOMBS and J. L. ALTY, Ed. London: Academic, chap. 10, pp. 337-371, 1981.

[14] P. HAYES, E. BALL and R. REDDY. "Breaking the man-machine communication barrier " (Computer), ISSN 0018-916[illegible], pp. 19-30, Mar. 1981.

[15] J. L. WOODWARD, "What makes business programming hard?", (Byte), ISSN 0360-5280, pp. 68-76, Oct. 1982.

[16] A. L. PAXTON and E. J. TURNER, "The application of human factors to the needs of the novice computer user," (Int. J. Man-Machine Studies), vol. 20, pp. 137-156, 1984.

[17] D. SMITH, C. IRBY, R. KIMBALL and B. VERPLANK, "Designing the Star user interface," (Byte), vol. 7, no. 4, pp. 242-282, Apr. 1982.

[18] R. B. ALLEN, "Cognitive factors in the use of menus and trees: an experiment," (IEEE J. on Selected Areas in Communications), vol. SAC-1, no. 2, pp. 333-336, Feb. 1983.

[19] D. MORELAND, "Human factors guidelines for terminal interface design," (Communications of the ACM), vol. 26. no. 7. pp. 484-494, July 1983.

[20] H. E. HANRAHAN and I. G. KENNEDY, "Software for SPC computers," in Electronic Telephone Switching, Continuing Engineering Education Course, Johannesburg: University of the Witwatersrand, Chap. 9, Feb. 1985.

[21] H. AOYAGI, N. UBUKATA, "Interactive data generator for Centennial II PABX," (Conf.: GLOBECOM '82. IEEE Global Telecommunications Conf.), New York: IEEE. IEEE Catalog no. CH 1819-2/82-0000-1013, vol. 3, pp. 1013-1017, 1982.

[22] CCITT Yellow Book, "Man-Machine Language MMI," ISBN 92-61-01111-X, Geneva: Int. Telecommun. Union. CCITT, Recommendations Z.311-Z.341, 1981.

[23] T. K. BREUNING, "Implementation of CCITT Man Machine Language in EWSD," (Int. Conf. on Software Engineering for Telecommunication Switching Systems [4th]), London: IEE Conf. Publication, ISBN 0-85296-242-8, pp. 209-213, 1981.

[24] B. H. HORNBACH, "MML: CCITT Man-Machine Language," (IEEE Trans. Commun.), ISSN 0090-6778, vol. COM-30, no. 6, pp. 1329-1336, June 1982.

[25] B. B. BERNSTEIN and A. S. KASHAR, "Intelligent terminals: functions, specifications and applications," Wellesley, Mass.: QED Information Sciences, ISBN 0-89435-021-8, p. 1-1, 1978.

[26] J. MARTIN, "Design of man-computer dialogues," Englewood Cliffs, NJ: Prentice-Hall, ISBN 0-13-201251-0, p. 123, 1973.

[27] N. C. SHU, V. Y. LUM, F. C. TUNG and C.L. CHANG, "Specification of forms processing and business procedures for office automation," (IEEE Trans. on Software Engineering), ISSN 0098-5589, vol. SE-8, no. 5, p. 499, Sep. 1982.

[28] R. J. BROCKMAN, "Reviews: care and attention finally came to screen format design," ACM SIGDOC, vol. 10, no. 1, p. 13. Apr. 1984. (A review of W. O. GALITZ's "Handbook of screen format design," Wellesley, Mass.: Q.E.D. Information Sciences Inc., ISBN 0-89435-052-8, 2nd printing, Mar. 1982.)

[29] CCITT Study Group XI, "Report on the meeting held in Geneva from 7 to 17 December 1981," (Report no. R 8.), Geneva: Int. Telecommun. Union. CCITT, pp. 133-138, Feb. 1982.

[30] -------- , "Report on the meeting held in Geneva from 21 June to 1 July 1982," (Report no. R 14.), Geneva: Int. Telecommun. Union. CCITT, p. 156, July 1982.

[31] J. P. GATEFAIT, P. SURLEAU, and J. L. KONRAT, "Execution mechanisms for administration programs in the E10.S system," (Int. Conf. on Software Engineering for Telecommun. Switching Systems [4th]), London: IEE Conf. Publication, ISBN 0-85296-242-8, pp. 130-131, 1981.

[32] C. S. MAGERS, "An experimental evaluation of on-line HELP for non-programmers," in (Proc. CHI'83 Human Factors in Computing Systems), ACM, New York, ISBN 0-89791-121-0, pp. 277-281, 1983.

[33] R. W. ROOT and S. DRAPER, "Questionnaires as a software evaluation tool," in (Proc. CHI'83 Human Factors in Computing Systems), ACM, New York, ISBN 0-89791-121-0, pp. 83-87, 1983.

[34] CIT-ALCATEL, "Operating manual DFN 0140 and DFN 0416," Lannion France: CIT-Alcatel GID, vol. 2.1, 1981.

[35] N. H. WEIDERMAN, "Personalizing large computers," (S.I.G. PC Notes), ACM Special Interest Group on Personal Computers, vol. 1, no. 4, pp. 33-35, Spring 1979.

[36] R. H. PERROTT (Ed.), "Symposium on software engineering," London: Academic, ISBN 0-12-551450-6, pp. 9, 39, 41-42, 1977.

[37] E. B. JAMES and D. IRELAND, "Microcomputers as protective interfaces in computing networks," (Software -- Practice and Experience), ISSN 0038-0644, vol. 10, no. 12, pp. 953-958. Dec. 1980.

[38] M. JORDAN, "A tool system for software development," (Ph. D. thesis), The University of Cambridge, Dec. 1981.

[39] A. J. WALKER, "Computer technology: science or engineering?," (Electron), Official Journal of the SAIEE., vol. 1, no. 7, pp. 46-50, July 1984.

[40] N. R. MCBURNEY II, "The personal computer as an interface to a corporate management information system," (Byte), ISSN 0360-5280, vol. 7, no. 10, pp. 315-358, Oct. 1982.

[41] J. VAN DEN BOS, M. J. PLASMEIJER and P. H. HARTEL, "Input-Output tools: a language facility for interactive and real-time Systems," (IEEE Trans. on Software Engineering), ISSN 0098-5589, vol. SE-9. no. 3. pp. 247-259, May 1983.

[42] J. W. BROWN, "Controlling the complexity of menu networks," (Commun. of the ACM), vol. 25, no. 7, pp. 412-418, July 1982.

[43] E. C. BERNICE and D. DASARATHY, "Modelling and validating the man-machine interface," (Software -- Practic and Experience), ISSN 0038-0644, vol. 12, no. 6, pp. 557-569, June 1982.

[44] T. W. DOLOTTA, J. S. LICWINKO, R. E. MENNINGER & W. D. ROOME, "The LEAP load and test driver," (Proc. Second Int. Conf. on Software Engineering), IEEE Cat. no. 76CH1125-4 C, p. 182, 13-15 Oct. 1976.

[45] B. W. LAMPSON, "Hints for computer system design," (IEEE Software), ISSN 0740-7459, vol. 1, no. 1, pp. 11-28, Jan. 1984.

[46] A. EVANS, "Help in Apple III Pascal," (Byte), ISSN 0360-5280, vol. 8, no. 8, pp. 477-482, Aug. 1983.

[47] S. D. HESTER, D. L. PARNAS, and D. F. UTTER, "Using documentation as a software design medium," (Bell Syst. Tech. J.), vol. 60, pp. 1941-1977, Oct. 1981.

[48] K. L. HENINGER, "Specifying software requirements for complex systems: new techniques and their application," (IEEE Trans. on Software Engineering), ISSN 0098-5589, vol. SE-6. no. 1. pp. 2-12, Jan. 1980.

[49] Y. CHU, "Introducing a Software Design Language," (Proc. Second Int. Conf. on Software Engineering), IEEE Cat. no. 76CH1125-4 C, pp. 297-304, 13-15 Oct. 1976.

[50] H. HANRAHAN, Personal communication, 1983.

[51] O. J. DAHL, E. W. DIJKSTRA and C. A. R. HOARE, "Structured programming," London: Academic, ISBN 0-12-200550-3, p. 27, 1972.

[52] CCITT White Book, "Data transmission," Geneva: Int. Telecommun. Union. CCITT, Vol. VIII, Recommendation V.3, pp. 1-10, 1969.

[53] APPLE, "Apple Pascal 1.2 update manual," Cupertino, California: Apple Computer Inc., Product 030-0602-A, 1983.

[54] N. HAMMOND, A. JORGENSEN, A. MACLEAN, P. BARNARD & J. LONG, "Design practice and interface usability: evidence from interviews with designers," in (Proc. CHI'83 Human Factors in Computing Systems), ACM, New York, ISBN 0-89791-121-0, pp. 40-44, 1983.

[55] R. C. TEITELBAUM and R. E. GRANDA, "The effects of positional constancy on searching menus for information," in (Proc. CHI'83 Human Factors in Computing Systems), ACM, New York, ISBN 0-89791-121-0, pp. 150-153, 1983.

[56] G. E. ANDERSON and K. E. SHUMATE, "Letters to the editor," (Computer), ISSN 0018-9162, vol. 15, no. 11, p. 7. Nov. 1982.

[57] G. E. ANDERSON, "Selecting a programming language, compiler, and support environment: method and example," (Computer), ISSN 0018-9162, pp. 29-36, Aug. 1982.

[58] S. J. YOUNG, "Real time languages: design and development," Chichester: Ellis Horwood, chap. 1, pp. 15-22, 1982.

[59] CCITT, Yellow Book, "CCITT high level language (CHILL)," ISBN 92-61-01121-7, Geneva: Int. Telecommun. Union. CCITT, Recommendation Z.200, 1981.

[60] R. F. BRENDER and I. R. NASSI, "What is Ada?", (Computer), ISSN 0018-9162, pp. 17-24, June 1981.

[61] R. T. BOUTE and M. I. JACKSOSN, "A Joint evaluation of the programming languages Ada and CHILL," Int. Conf. on Software Engineering for Telecommunication Switching Systems (4th), London: IEE Conf. Publication, ISBN 0-85296-242-8, pp. 214-220, 1981.

[62] T. H. WOTEKI and P. A. SAND, "Four implementations of Pascal," (Byte), ISSN 0360-5280, vol. 7, no. 3, pp. 316-356, Mar. 1982.

[63] M. C. JOHNSON and A. MUNRO, "Pascal's design flaws -- Modula-2 solutions and Pascal patches," (Byte), ISSN 0360-5280, vol. 9, no. 3, pp. 371-388, Mar. 1984.

[64] E. ELDRED, "Volition Systems' Modula-2," (Byte), ISSN 0360-5280, vol. 9, no. 6, pp. 353-364, June 1984.

[65] J. D. ARON, "The program development process part 1 -- the individual programmer," Reading, Mass.: Addison-Wesley, ISBN 0-201-14451-4, p. 16, 1974.

[66] H. F. LEDGARD, P. A. NAGIN and J. F. HUERAS, "Pascal with style: programming proverbs," Rochelle Park, New Jersey: Hayden, ISBN 0-8104-5124-7, pp. 161-164, 1979.

[67] N. H. GEHANI, "An electronic form system -- an experience in prototyping," (Software -- Practice and Experience), ISSN 0038-0644, vol. 13, no. 6, pp. 479-486, June 1983.

[68] W. GALITZ, "Human engineering in screen design," (J. of Systems Management), pp. 6-11, May 1983.

[69] L. A. ROWE and K. A. SHOENS "Programming language constructs for screen definition," (IEEE Trans. on Software Engineering), ISSN 0098-5589, vol. SE-9, no. 1, pp. 31-39, Jan. 1983.

[70] T. D. NATHAN and D. JOHNSON, personal communications, 10 and 16 May 1983.

[71] APPLE, "Apple super serial card installation and operating manual," Cupertino, California: Apple Computer Inc., Product # A2L0044, p. 28, 1981.

[72] ANSI, "Code extension techniques for use with the 7-bit coded character set of American national Standard Code for Information Interchange," New York: American National Standards Institute, ANSI X3.41-1974, p. 10, (C) 1975.

[73] WORLD SYSTEM, "The World System Teletext Technical Specification and an Introduction to Videotext," Information Technology Division, Dept. Trade and Industry, London: Fig. 8, Jan. 1984.

[74] W. MYERS, "Can software development process improve -- drastically?" (IEEE Software), ISSN 0740-7459, vol. 1, no. 3, pp. 101-102, July 1984

[75] F. S. INGRASSIA, "Combating the '90% complete' syndrome," (Datamation), pp. 171-176, Jan. 1978.

[76] S. PAKIN, "Evaluate user documentation before you buy the software," (IEEE Trans. on Prof. Commun.), ISSN 0361-1434, vol. PC-24, no. 2, pp. 75-78, June 1981.

[77] J. MAYNARD, "A user-driven approach to better user manuals," (IEEE Trans. on Prof. Commun.), ISSN 0361-1434, vol. PC-25, no. 1, pp. 16-19, Mar. 1982.

[78] A. R. GUDKNECHT, "Well-written documentation leaves nothing to imagination," (IEEE Trans. on Prof. Commun.), ISSN 0361-1434, vol. PC-25, no. 3, pp. 112-119, Sep. 1982.

[79] J. C. BATZ, P. M. COHEN, S. T. REDWINE and J. R. RICE, "The application-specific task area," (Computer), ISSN 0018-9162, vol. 16, no. 11, pp. 78-85, Nov. 1983.

[80] R. CLARK (Ed.), "UCSD p-system internal architecture guide," San Diego: SofTech Microsystems, First Edition, pp. 127-131, Mar. 1981.

[81] R. W. JENSEN and C. C. TONIES, "Software engineering," Englewood Cliffs, New Jersey: Prentice-Hall, ISBN 0-13-822130-8, 1979.

[82] A. LAMSWEERDE, "Automatisation de la production de logicel d'application: quelques approaches 3e partie, Computer aided application programming: some insights 3rd part," (Technique et Science Informatiques), vol. 2, no. 2, pp. 81-93, 1983.

[83] R. CLARKE, "A background to program generators for commercial applications," (Aust. Comput. J.), vol. 14, no. 2, pp. 48-55, May 1982.

[84] J. M. GINSPARG and R.D. GORDON, "Automatic programming of communications software via nonprocedural descriptions," (IEEE Trans. Commun.), ISSN 0090-6778, vol. COM-30, no. 6, pp. 1343-1347, June 1982.

[85] D. L. PARNAS, "On the design and development of program families," (IEEE Trans. Software Eng.), ISSN 0098-5589, vol. SE-2, no. 1, pp. 1-9, Mar. 1976.

[86] APPLE, "Apple II Apple Pascal language reference manual," Cupertino: Apple Computer, pp. 128-130, 1980.

[87] INTERNATIONAL STANDARDS ORGANISATION, "Reference model of Open Systems Interconnection for CCITT applications," Draft Recommendation X.200, Annex (to circular letter No. 58), Dec. 1982.

[88] R. DEGROAT, "Pascal memory utility," (Call-Apple), vol. 4, no. 6, p. 45, July/Aug 1981.

[89] -------- , "Inverse and flashing modes for Apple Pascal text display," (Call-Apple), vol. 3, no. 6, July/Aug. 1980.

[90] M. S. DEUTSCH, "Software verification and validation," Englewood Cliffs, New Jersey: Prentice-Hall, ISBN 0-13-822072-7, p. 4, 1982.

[91] IEEE, "IEEE standard for software test documentation," New York: IEEE, Std 829-1983, Feb. 1983.

[92] G. M. WEINBERG, "The psychology of computer programming," New York: Van Nostrand, ISBN 0-442-29264-3, pp. 266, 267, 1971.

[93] P. A. SAND, "Advanced Pascal programming techniques," Berkeley, California: Osborne/McGraw-Hill, ISBN 0-88134-105-3, pp. 270, 271, 1984.

[94] CIT-ALCATEL, "Operating manual 00032 Aa-00252 Aa," Lannion France: CIT-Alcatel GID, vol. 2, July 1982.

[95] -------- , "Malfunction report message dictionary DCD 00726," Lannion: CIT-Alcatel GID, Iss. 1, May 1982.

[96] G. J. MYERS, "The art of software testing," New York.: John Wiley & Sons, ISBN 0-471-04328-1, 1979.

[97] M. A. BRANSTAD, J. C. CHERNIAVSKY and W. R. ADRION, "Validation, verification, and testing for the individual programmer," (Computer), ISSN 0018-9162, Dec. 1980.

[98] .. MARKOFF, "The Apple IIc personal computer," (Byte), ISSN 0360-5280, vol. 9, no. 5, pp. 276-284, May 1984.

[99] W. DEHNING, H. ESSIG and S. MAASS, "The adaptation of virtual man-computer interfaces to user requirements in dialogs," Berlin: Springer-Verlag, ISBN 3-540-10826-2, p. 24, 1981.